מבוא לאלגוריתמים

תומאס ה. קורמן
צ'ארלס א. לייזרסון
רונאלד ל. ריבסט

1321007

מבוא לאלגוריתמים

כרך ב

אוניברסיטת בן-גוריון בנגב
הספרייה

כרך א:
I יסודות מתמטיים
II מיון וערכי מיקום
III מבני נתונים
IV שיטות מתקדמות לתכנון ולניתוח

כרך ב:
V מבני נתונים מתקדמים
VI אלגוריתמים על גרפים
VII נושאים נבחרים

QA
76.6
.C66212
1995
VOL.2
copy12

Introduction to Algorithms

Thomas H. Cormen
Charles E. Leiserson
Ronald L. Rivest

Original English edition copyright © 1990 by the Massachusetts Institute of Technology.
The MIT Press, Cambridge, 1990.

אחראי אקדמי: פרופ' דוד הראל

ריכוז הפיתוח: פרופ' יהודית גל-עזר

תרגום: תמר אלמוג-אלטון
עריכה: תלמה מוקדי

גרפיקה ועיצוב עטיפה: גילה ברון
התקנה למקינטוש: רותי אפלברג
סדר מקינטוש: אילנה רימון, מנוחה סליגמן-מרביץ

20407
20417
22910

מסת"ב ISBN 965-06-0415-4

הדפסה שנייה

© תשנ"ט — 1999. כל הזכויות שמורות לאוניברסיטה הפתוחה.
בית ההוצאה לאור של האוניברסיטה הפתוחה, רח' קלאוזנר 16, רמת-אביב, ת"ד 39328, תל-אביב 61392.
The Open University of Israel, 16 Klausner St. Ramat-Aviv, P.O.Box 39328, Tel-Aviv, 61392.
Printed in Israel.
סודר ועומד במערכת DTP של האוניברסיטה הפתוחה.

תוכן העניינים

V מבני נתונים מתקדמים

מבוא 3

19 עצי B 5
19.1 הגדרת עצי B 8
19.2 פעולות בסיסיות על עצי B 10
19.3 מחיקת מפתח מעץ B 16

20 ערימות בינומיות 21
20.1 עצים בינומיים וערימות בינומיות 22
20.2 פעולות על ערימות בינומיות 26

21 ערימות פיבונצ'י 38
21.1 המבנה של ערימות פיבונצ'י 39
21.2 פעולות על ערימות מתמזגות 41
21.3 הקטנת ערכו של מפתח ומחיקת צומת 48
21.4 חסימת הדרגה המקסימלית 52

22 מבני נתונים לקבוצות זרות 56
22.1 פעולות על אוסף קבוצות זרות 56
22.2 ייצוג קבוצות זרות על-ידי רשימות מקושרות 58
22.3 יערות של קבוצות זרות 62
* 22.4 ניתוח של איחוד על פי דרגה יחד עם כיווץ מסלולים 65

VI אלגוריתמים על גרפים

מבוא 79

23 אלגוריתמים בסיסיים על גרפים 80
23.1 ייצוגים של גרפים 80
23.2 חיפוש לרוחב 83
23.3 חיפוש לעומק 90
23.4 מיון טופולוגי 97
23.5 רכיבים קשירים היטב 100

24 עצים פורשים מינימליים 108
24.1 "הצמחתי" עץ פורש מינימלי 109
24.2 האלגוריתמים של קרוסקל ושל פרים 113

25 מסלולים קצרים ביותר ממקור יחיד 120
25.1 מסלולים קצרים ביותר וטכניקת ההקלה 124
25.2 האלגוריתם של דייקסטרה 131
25.3 אלגוריתם בלמן־פורד 136
25.4 מסלולים קצרים ביותר ממקור יחיד בגרף מכוון ללא מעגלים 139
25.5 אילוצי הפרשים ומסלולים קצרים ביותר 142

26 מסלולים קצרים ביותר בין כל הזוגות 152
26.1 מסלולים קצרים ביותר וכפל מטריצות 154
26.2 אלגוריתם פלויד־וורשאל 159
26.3 האלגוריתם של ג'ונסון עבור גרפים דלילים 166
* 26.4 מסגרת כללית לפתרון בעיות מסלולים בגרפים מכוונים 170

27 זרימה מקסימלית 178
27.1 רשתות זרימה 179
27.2 שיטת פורד־פולקרסון 185
27.3 זיווג מקסימלי בגרף דו־צדדי 196
* 27.4 אלגוריתמי דחיפת־קדם־זרימה 200
* 27.5 אלגוריתם הרם־וקדם 209

VII נושאים נבחרים

מבוא 225

28 רשתות מיון 227
28.1 רשתות השוואה 227
28.2 עקרון ה־0-1 231
28.3 רשת מיון ביטונית 234
28.4 רשת מיזוג 237
28.5 רשת מיון 239

29 מעגלים אריתמטיים 244
29.1 מעגלים צירופיים 244
29.2 מעגלי חיבור 249
29.3 מעגלי כפל 259
29.4 מעגלים מבוקרי־שעון 265

30 אלגוריתמים עבור מחשבים מקביליים 274
30.1 הקפצת מצביעים 277
30.2 אלגוריתמי CRCW לעומת אלגוריתמי EREW 285
30.3 משפט ברנט ויעילות־בעבודה 292
* 30.4 חישוב רישות מקבילי יעיל־בעבודה 296
30.5 שבירת סימטריה באופן דטרמיניסטי 301

31 פעולות על מטריצות 311
31.1 תכונות של מטריצות 311
31.2 האלגוריתם של שטראסן לכפל מטריצות 320
* 31.3 מערכות משפרים אלגבריות וכפל מטריצות בוליאניות 326
31.4 פתרון מערכת משוואות לינאריות 330
31.5 היפוך מטריצות 342
31.6 מטריצות סימטריות חיוביות־לחלוטין וקירוב על־ידי ריבועים פחותים 346

32 פולינומים והתמרת פוריה המהירה (FFT) 356
32.1 ייצוג פולינומים 358
32.2 DFT ו־FFT 363
32.3 מימושים יעילים של ה־FFT 370

33 אלגוריתמים בתורת המספרים 379
33.1 מושגים בסיסיים בתורת המספרים 380
33.2 מחלק משותף מקסימלי 385
33.3 אריתמטיקה מודולרית 390
33.4 פתרון משוואות לינאריות מודולריות 396
33.5 משפט השאריות הסיני 399
33.6 חזקות של איבר 402
33.7 מערכת ההצפנה RSA — הצפנה באמצעות מפתח ציבורי 406
* 33.8 בדיקת ראשוניות 411
* 33.9 פירוק לגורמים של מספרים שלמים 418

34 התאמת מחרוזות תווים 426
34.1 האלגוריתם הנאיבי להתאמת מחרוזות 428
34.2 אלגוריתם רבין־קארפ 430
34.3 התאמת מחרוזות באמצעות אוטומטים סופיים 434
34.4 אלגוריתם קנות־מוריס־פראט 440
* 34.5 אלגוריתם בויאר־מור 447

35 גיאומטריה חישובית 454
35.1 תכונות של קטעי ישרים 454
35.2 כיצד קובעים אם שני קטעים כלשהם נחתכים 459
35.3 מציאת הקמור 464
35.4 מציאת זוג הנקודות הקרובות ביותר 473

36 שלמות ב־NP 480
36.1 זמן פולינומיאלי 481
36.2 אימות בזמן פולינומיאלי 487
36.3 שלמות ב־NP ורדוקציות 491
36.4 הוכחות שלמות ב־NP 499
36.5 בעיות NP־שלמות 506

37 אלגוריתמי קירוב 521
37.1 בעיית הכיסוי על־ידי קדקודים 523
37.2 בעיית הסוכן הנוסע 525
37.3 בעיית כיסוי הקבוצה 529
37.4 בעיית סכום התת־קבוצה 533

ביבליוגרפיה 541

מפתח מונחים עברי־אנגלי 551
מפתח מונחים אנגלי־עברי 597

V מבני נתונים מתקדמים

משאיר בקופם המשכה, ובמשכה עכבה של $\alpha(m, n)$. אינו עולה על 4. כמובן שמספר המתחנים n (n, m) הוא פונקציה שקצב גידולה אטי למדיים — כל עוד n אינו עולה על מספר האטומים בעולם הנראה לעין, שהוא בערך 10^{80}. אפשר שמשום כך יוצגו כל הקבועות שנתקלים בהם בניתוח אלגוריתמים בפונקציה $\alpha(m, n)$ ולא בקבוע כלשהו, אך ההוכחות שנציג בפרק 22 מראות שקבועות אלה אכן מתקיימות.

לבסוף, בפרק 22 מראים כמה קשה לחשב את $\alpha(m, n)$ על ידי הגדרת פונקציה פשוטה יחסית $A(i, j)$ שגדלה מהר מאוד כאשר i ו־j גדלים. אנו מגדירים את $\alpha(m, n)$ להיות הערך הקטן ביותר של i שעבורו $A(i, j)$ גדולה יותר מ־n כאשר j שווה ל־$\lfloor m/n \rfloor$. אנו מראים שהפונקציה $A(i, j)$ גדלה מהר מאוד כאשר i גדל, ולכן הפונקציה $\alpha(m, n)$ קטנה מאוד.

מבוא

בחלק זה נשוב ונבחן מבני נתונים ותמכים בפעולות על קבוצות דינמיות, אולם ברמה מתקדמת יותר מאשר בחלק III. לדוגמה, בשניים מן הפרקים נעשה שימוש נרחב בשיטות הניתוח לשיעורין שראינו בפרק 18.

פרק 19 נציג עצי־B, שהם עצי חיפוש מאוזנים המתוכננים לעבוד אחסון בדיסקים מגנטיים. בפרק 20 נראה דוגמאות לכך שהעצים יישארו מאוזנים.

על עצי־B דואגות לכך שהעצים יישארו מאוזנים.

הגישה לדיסק, עבור כל פעולה על עצי־B, מספר הגישות לדיסק גדל עם גובה העץ, והפעולות במפרקים 20 ו־21 נציג ממשים של ערימות ממזגות, התומכות בפעולות MINIMUM, INSERT, UNION ו־EXTRACT-MIN. הפעולה UNION מאחדת, או ממזגת, שתי ערימות. מבני הנתונים המוצגים במפרקים אלה תומכים גם בפעולות DELETE ו־DECREASE-KEY.

ערימות בינומיות, המופיעות במפרק 20, תומכות בכל אחת מן הפעולות האלה בזמן $O(\lg n)$ במקרה הגרוע, כאשר n הוא המספר הכולל של איברים בערימה או בשתי ערימות מקלט (UNION). מתגלה עדיפותן של ערימות פיבונאצי, המוצגות בפרק 21. מהותן שיפור של ערימות בינומיות. לפחות במובן התיאורטי. אנו משתמשים בחסמים של זמן לשערוך כדי למדוד את ביצועיהן. הזמן המשוערך של פעולה הוא זמן הנדרש לביצוע מספר כלשהו של פעולות, כאשר לכל פעולה שמבוצעת יש עלות משוערכת. לעלות המשוערכת של פעולה יכול להיות גודל שונה מגודל העלות האמיתית של הפעולה. אם עלויות משוערכות חסומות מלעיל על־ידי $O(\lg n)$ לכל n פעולות, אזי גם העלות האמיתית של n פעולות כאלה חסומה מלעיל על־ידי $O(n \lg n)$. העלות המשוערכת של פעולה חסומה מלעיל על־ידי $O(\lg n)$ אף־על־פי שפעולה יחידה עשויה לקחת זמן $\Omega(n)$. בערימות פיבונאצי, הזמן המשוערך של הפעולות INSERT ו־UNION הוא $O(1)$. העלות המשוערכת של EXTRACT-MIN ו־DELETE היא $O(\lg n)$. החשיבות של ערימות פיבונאצי היא בכך שהעלות המשוערכת של DECREASE-KEY היא $O(1)$. מכיוון שפרוצדורות שמבצעות פעולות DECREASE-KEY מופעלות בזמן לעתים קרובות יותר מאשר הפרוצדורות המבצעות פעולות אחרות, הזמן המשוערך של פעולה זו לכל ערך n הוא מן הסתם הקריטי ביותר בקביעת הביצועים של האלגוריתמים המתבססים על ערימות פיבונאצי. הן לכם לכם של כמה מן האלגוריתמים המהירים ביותר אסימפטוטית המוכרים לנו כיום למבנים בעזרת ערימות פיבונאצי.

כן מידת מורכבותו של הניתוח לשיעורין המוכיח חסם זמן זה. בפרק 22 נוכיח חסם מעניין, אם כי פשוט קצת יותר, על זמן הריצה.

הנושאים הנסקרים בחלק זה אינם בשום פנים ואופן הדוגמאות היחידות למבני נתונים "מתקדמים". להלן כמה מבני נתונים מתקדמים אחרים:

- מבנה נתונים שהומצא על־ידי van Emde Boas [194] תומך בפעולות MINIMUM, MAXIMUM, INSERT, DELETE, SEARCH, EXTRACT-MIN, EXTRACT-MAX, PREDECESSOR ו־SUCCESSOR בזמן $O(\lg\lg n)$ במקרה הגרוע, זאת בתנאי שהקבוצה האוניברסלית של המפתחות היא הקבוצה $\{1, 2, \ldots, n\}$.

- **עצים דינמיים** (dynamic trees), שהוצעו על־ידי Sleator & Tarjan [177] ונדונים אצל Tarjan [188], מתחזקים יער של עצים מושרשים זרים. לכל קשת בכל עץ יש עלות שהיא מספר ממשי. עצים דינמיים תומכים בשאילתות למציאת אבות, שורשים, עלויות של קשתות, ועלות הקשתות המינימלית של מסלול מצומת לשורש. הפעולות המותרות על עצים דינמיים כוללות חיתוך קשתות, עדכון כל עלויות הקשתות על מסלול מצומת לשורש, קישור שורש לעץ אחר והפיכת צומת לשורש העץ שבו הוא מופיע. באחד המימושים של עצים דינמיים, כל אחת מן הפעולות ניתנת לביצוע בזמן לשיעורין החסום על־ידי $O(\lg n)$; מימוש מורכב יותר נותן חסם של $O(\lg n)$ במקרה הגרוע.

- **עצי splay**, שפותחו בידי Sleator & Tarjan [178] ונדונים אצל Tarjan [188], הם סוג של עצי חיפוש בינריים שבהם פעולות החיפוש הרגילות רצות בזמן לשיעורין של $O(\lg n)$. אחד היישומים של עצי splay הוא פישוט של עצים דינמיים.

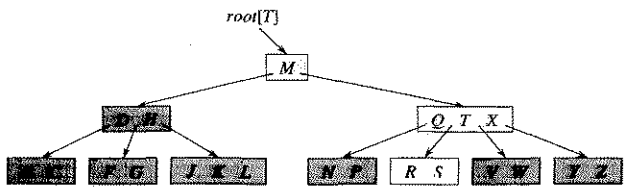
- מבני נתונים **מתמידים** (persistent) מאפשרים שאילתות, ולעתים גם עדכונים, על גרסאות קודמות של מבני נתונים. Driscoll, Sarnak, Sleator & Tarjan [59] מציגים טכניקות להפיכת מבני נתונים מקושרים למתמידים בעלות נמוכה של זמן ומקום אחסון. בעיה 14-1 מביאה דוגמה פשוטה לקבוצה דינמית מתמדת.

19 עצי־B

עצי־B הם עצי חיפוש מאוזנים המתוכננים כך שיפעלו היטב על דיסקים מגנטיים או התקני אחסון משניים אחרים הפועלים בשיטת הגישה הישירה. עצי־B דומים לעצים אדומים־שחורים (פרק 14), אולם הם טובים יותר במזעור מספר פעולות הקלט/פלט על דיסקים.

עצי־B שונים באופן משמעותי מעצים אדומים־שחורים בכך שלצומתיו של עץ־B יכולים להיות בנים רבים, מבנים ספורים ועד לכמה אלפים. כלומר, "גורם ההסתעפות" (branching factor) של עץ־B עשוי להיות גדול למדי, אם כי הוא נקבע בדרך כלל על פי מאפייניה של יחידת הדיסק שמשתמשים בה. עצי־B דומים לעצים אדומים־שחורים בכך שגובהו של כל עץ־B n־צומתי הוא $O(\lg n)$, אף כי עץ־B עשוי להיות נמוך במידה משמעותית מעץ אדום־שחור, משום שגורם ההסתעפות שלו עשוי להיות גדול בהרבה. משום כך ניתן להשתמש בעצי־B גם כדי לממש פעולות רבות על קבוצה דינמית בזמן $O(\lg n)$.

עצי־B הם הכללה טבעית של עצי חיפוש בינריים, באיור 19.1 מוצג עץ־B פשוט. אם צומת x של עץ־B מכיל $n[x]$ מפתחות, אזי לצומת x יש $n[x] + 1$ בנים. המפתחות בצומת x משמשים כנקודות הפרדה המחלקות את תחום המפתחות ש־x מטפל בהם ל־$n[x] + 1$ תת־תחומים שכל אחד מהם מטופל על־ידי אחד הבנים של x. כאשר מחפשים מפתח בעץ־B, מקבלים החלטה $(n[x] + 1)$־כיוונית המתבססת על השוואות עם $n[x]$ המפתחות המאוחסנים בצומת x.



איור 19.1 עץ־B שמפתחותיו הם העיצורים בשפה האנגלית. לצומת פנימי x המכיל $n[x]$ מפתחות יש $n[x] + 1$ בנים. כל העלים בעץ הם בעלי אותו עומק. הצמתים הבהירים הם הצמתים הנבחנים במהלך חיפוש אחר האות R.

סעיף 19.1 מביא הגדרה מדויקת של עצי־B ומוכיח כי גובהו של עץ־B גדל באופן לוגריתמי בלבד עם מספר הצמתים שהוא מכיל. סעיף 19.2 מתאר כיצד מחפשים מפתח בעץ־B וכיצד מכניסים מפתח לעץ־B, וסעיף 19.3 דן במחיקה. אולם לפני שנמשיך, עלינו לשאול מדוע ביצועיהם של מבני נתונים המתוכננים לפעול על דיסקים מגנטיים נמדדים באופן שונה מאשר אלה של מבני נתונים הבנויים לפעול על זיכרון גישה־אקראית ראשי.

מבני נתונים על התקני אחסון משניים

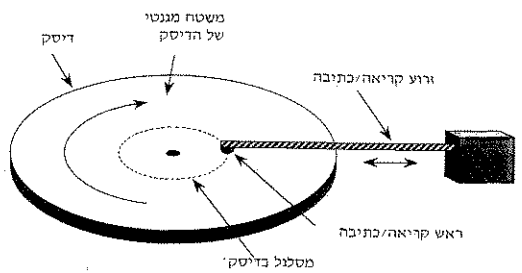
קיימות טכניקות רבות ושונות שבאמצעותן ניתן לספק קיבולת זיכרון במערכת מחשב. **הזיכרון הראשי** (primary memory) או **העיקרי** במערכת מחשב מורכב בדרך־כלל משבבי זיכרון עשויים צורן (סיליקון), שכל אחד מהם מסוגל להכיל מיליון סיביות נתונים. העלות לסיבית של טכנולוגיה זו גבוהה מזו של טכנולוגיית אחסון מגנטי, כגון סרטים או דיסקים. מערכת מחשב טיפוסית מכילה **התקני אחסון משניים** (secondary storage) המבוססים על דיסקים מגנטיים. קיבולת הזיכרון המשני עולה לעתים קרובות בסדרי גודל אחדים על קיבולת הזיכרון הראשי.

באיור 19.2 מוצג כונן דיסקים טיפוסי. שטח פני הדיסק מכוסה בחומר הניתן למגנוט. ראש הקריאה/כתיבה יכול לקרוא או לכתוב נתונים באופן מגנטי על פני הדיסק המסתובב. זרוע הקריאה/כתיבה יכולה להציב את הראש במרחקים שונים ממרכז הדיסק. כאשר הראש קבוע במקומו, המשטח העובר מתחתיו נקרא **מסלול** (track). המידע המאוחסן בכל מסלול מחולק לעתים קרובות למספר קבוע של **דפים** (pages) שווים בגודלם; בדיסק טיפוסי, אורכו של דף עשוי להיות 2048 בתים. יחידת המידע הבסיסית לאחסון ולאחזור היא בדרך כלל דף מידע — כלומר, הקריאה והכתיבה על הדיסק הן של דפים שלמים. **זמן הגישה** (access time) — הזמן הדרוש כדי למקם את ראש הקריאה/כתיבה ולחכות עד שדף המידע הנדרש יעבור מתחת לראש — עלול להיות גדול (לדוגמה, 20 מילי־שניות), ואילו זמן הקריאה או הכתיבה של דף, לאחר שכבר התבצעה הגישה אליו, הוא קטן. המחיר שמשלמים עבור עלותן הנמוכה של טכניקות האחסון המגנטי הוא אפוא הזמן הארוך יחסית הנדרש לגישה לנתונים. מכיוון שקל הרבה יותר להניע אלקטרונים מאשר להניע עצמים גדולים (או אפילו קטנים), זמן הגישה להתקני אחסון אלקטרוניים לחלוטין, כגון שבבי זיכרון מצורן, קטן בהרבה מזמן הגישה להתקני אחסון שיש בהם חלקים נעים, כגון כונני דיסקים מגנטיים. עם זאת, ברגע שהכל מוצב במקומו, קריאה מדיסק מגנטי או כתיבה עליו מתבצעות באופן אלקטרוני לחלוטין (מלבד התנועה הסיבובית של הדיסק), כך שניתן לקרוא או לכתוב במהירות כמויות גדולות של נתונים.

לעתים קרובות, הגישה לדף מידע וקריאתו מהדיסק צורכות זמן ארוך יותר מהזמן הדרוש למחשב כדי לבחון את כל המידע שנקרא. זוהי הסיבה לכך שבפרק זה נבחן בנפרד את שני המרכיבים העיקריים של זמן הריצה:

- מספר הגישות לדיסק;
- זמן היעי"מ (CPU time).

מספר הגישות לדיסק נמדד במונחי מספר דפי המידע שיש לקרוא מן הדיסק או לכתוב עליו. יש לציין שזמן הגישה לדיסק אינו קבוע — הוא תלוי במרחק בין המסלול הנוכחי למסלול המבוקש וכן במצבו ההתחלתי של הדיסק מבחינת תנועתו הסיבובית. אף על פי כן נשתמש במספר הדפים הנקראים או הנכתבים כקירוב גס מסדר ראשון של הזמן הכולל שאורכת הגישה לדיסק.



איור 19.2 כונן דיסקים טיפוסי.

ביישום טיפוסי של עץ־B, כמות הנתונים שמטפלים בהם כה גדולה עד שלא ניתן להכניס את כל הנתונים בבת אחת לזיכרון הראשי. האלגוריתמים עבור עצי־B מעתיקים דפים נבחרים מן הדיסק אל הזיכרון הראשי לפי הצורך, וכותבים בחזרה על הדיסק דפים שחלו בהם שינויים. מאחר שהאלגוריתמים עבור עצי־B זקוקים רק למספר קבוע של דפים בזיכרון הראשי בכל רגע נתון, גודל הזיכרון הראשי אינו מגביל את גודלם של עצי־B שאפשר לנהל.

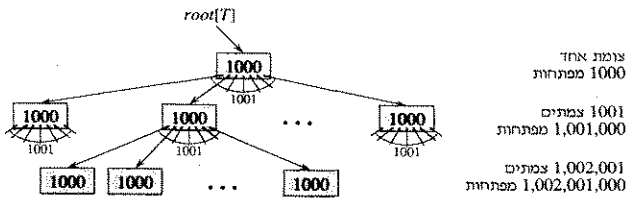
בפסאודו־קוד שלנו נשתמש במודל הבא של פעולות על דיסק. יהי x מצביע לעצם. אם במצב הנוכחי העצם נמצא בזיכרון הראשי של המחשב, אזי ניתן לפנות לשדותיו באופן הרגיל; לדוגמה, $key[x]$. אולם אם העצם שעליו מצביע x נמצא על דיסק, אזי יש לבצע את הפעולה DISK-READ(x) כדי לקרוא את העצם x אל הזיכרון הראשי לפני שניתן לפנות לשדותיו. (אנו מניחים שאם x כבר נמצא בזיכרון הראשי, אזי DISK-READ(x) אינה מבצעת גישות לדיסק; ניתן לומר שהיא מתבצעת "באופן ריק".) באופן דומה, הפעולה DISK-WRITE(x) משמשת לשמירת השינויים שנעשו בשדותיו של העצם x. כלומר, תבנית העבודה הטיפוסית עם עצם היא כדלקמן:

```
1  . . .
2  x ← a pointer to some object
3  DISK-READ(x)
4  operations that access and/or modify the fields of x
5  DISK-READ(x)          ▷ Omitted if no fields of x were changed.
6  other operations that access but do not modify fields of x
7  . . .
```

המערכת מסוגלת להחזיק בזיכרון הראשי רק מספר מוגבל של דפים בכל זמן נתון. אנו נניח שהמערכת דואגת לסלק מן הזיכרון הראשי דפים שאינם עוד בשימוש; האלגוריתמים שלנו המטפלים בעצי־B יתעלמו מעניין זה.

מאחר שבמרבית המערכות זמן הריצה של אלגוריתם עבור עץ־B נקבע בעיקר על־ידי מספר הפעולות DISK-READ ו־DISK-WRITE שהוא מבצע, הרי שהדבר ההגיוני הוא לעשות שימוש ממצה בפעולות אלה, דהיינו, לקרוא או לכתוב בכל פעם כמות גדולה ככל האפשר של מידע. לפיכך, גודלו של צומת בעץ־B הוא בדרך כלל כגודל דף שלם בדיסק. מספר הבנים שיכולים להיות לצומת בעץ־B מוגבל אפוא על־ידי גודלו של דף בדיסק.

עבור עץ־B גדול המאוחסן בדיסק, משתמשים לעתים קרובות בגורמי הסתעפות הנעים בין 50 ל־2000; הדבר תלוי בגודל המפתח ביחס לגודל הדף. גורם הסתעפות גדול מקטין בצורה דרמטית את גובה העץ ואת מספר הגישות לדיסק הנדרשות כדי למצוא מפתח כלשהו. באיור 19.3 מוצג עץ־B בעל גורם ההסתעפות 1001 וגובה 2, שיכול להכיל יותר ממיליארד מפתחות; למרות זאת, מכיוון שאת צומת השורש ניתן לשמור באופן קבוע בזיכרון הראשי, יידרשו רק *שתי* גישות לדיסק לכל היותר כדי למצוא מפתח כלשהו בעץ זה.



איור 19.3 עץ־B בגובה 2 המכיל יותר ממיליארד מפתחות. כל צומת פנימי וכל עלה מכילים 1000 מפתחות. בעומק 1 יש 1001 צמתים, ובעומק 2 — יותר ממיליון עלים. בכל צומת x רשום $n[x]$ — מספר המפתחות ב־x.

19.1 הגדרת עצי־B

מטעמי פשטות נניח, כפי שעשינו עבור עצי חיפוש בינריים ועצים אדומים־שחורים, שכל "הנתונים הנלווים" למפתח מסוים מאוחסנים באותו צומת שבו מאוחסן המפתח. ביישומים מעשיים אפשר למעשה לאחסן עם כל מפתח רק מצביע לדף אחר בדיסק, המכיל את הנתונים הנלווים למפתח זה. הפסאודו־קוד שבפרק זה מניח במובלע שהנתונים הנלווים למפתח מסוים, או מצביע לנתונים אלה, נודדים עם המפתח בכל עת שהוא מועבר מצומת לצומת. דרך שכיחה אחרת לארגון נתונים בעץ־B היא לאחסן את כל הנתונים הנלווים בעלים ולאחסן רק מפתחות ומצביעים לבנים בצמתים הפנימיים; בדרך זו אפשר להגדיל עד למקסימום את גורם ההסתעפות של הצמתים הפנימיים.

עץ־B T הוא עץ מושרש (ששורשו $root[T]$) המקיים את התכונות הבאות.

1. כל צומת x מכיל את השדות הבאים:
   א. $n[x]$ — מספר המפתחות המאוחסנים ברגע הנוכחי בצומת x;
   ב. $n[x]$ המפתחות עצמם, מאוחסנים בסדר לא יורד:
   $$key_1[x] \le key_2[x] \le \cdots \le key_{n[x]}[x]$$;
   ג. $leaf[x]$ — שדה בוליאני שערכו TRUE אם x הוא עלה ו־FALSE אם x הוא צומת פנימי.
2. אם x הוא צומת פנימי, הוא מכיל גם $n[x]+1$ שדות של מצביעים לבניו, $c_1[x], c_2[x], \ldots, c_{n[x]+1}[x]$. לצמתים שהם עלים אין בנים, ולפיכך השדות c_i שלהם אינם מוגדרים.
3. המפתחות $key_i[x]$ מפרידים בין תחומי המפתחות המאוחסנים בכל תת־עץ: אם k_i הוא מפתח כלשהו המאוחסן בתת־עץ המושרש ב־$c_i[x]$, אזי:
   $$k_1 \le key_1[x] \le k_2 \le key_2[x] \le \cdots \le key_{n[x]}[x] \le key_{n[x]+1}$$
4. לכל העלים יש אותו עומק, שהוא גובה העץ h.
5. קיימים חסמים תחתונים ועליונים על מספר המפתחות שצומת יכול להכיל. את החסמים האלה אפשר לבטא במונחי קבוע שלם $t \ge 2$ הנקרא **דרגת המינימום** (minimum degree) של העץ־B:
   א. כל צומת שאינו השורש חייב להכיל לפחות $t-1$ מפתחות. לכל צומת פנימי שאינו השורש יש אפוא לפחות t בנים. אם העץ אינו ריק, השורש חייב להכיל לפחות מפתח אחד.
   ב. כל צומת יכול להכיל $2t-1$ מפתחות לכל היותר. לכן צומת פנימי יכול להכיל $2t$ בנים לכל היותר. אם צומת מכיל בדיוק $2t-1$ מפתחות נאמר שהוא **מלא** (full).

עץ־B הפשוט ביותר מתקבל כאשר $t=2$. במקרה זה, לכל צומת פנימי יש 2, 3 או 4 בנים, והעץ המתקבל נקרא **עץ 2-3-4** (2-3-4 tree). אולם ביישומים מעשיים משתמשים בדרך כלל בערכי t גדולים בהרבה.

גובהו של עץ־B

מספר הגישות לדיסק הדרושות עבור רוב הפעולות על עץ־B נמצא ביחס ישר לגובהו של העץ. נבחן עתה את גובהו של עץ־B במקרה הגרוע ביותר.

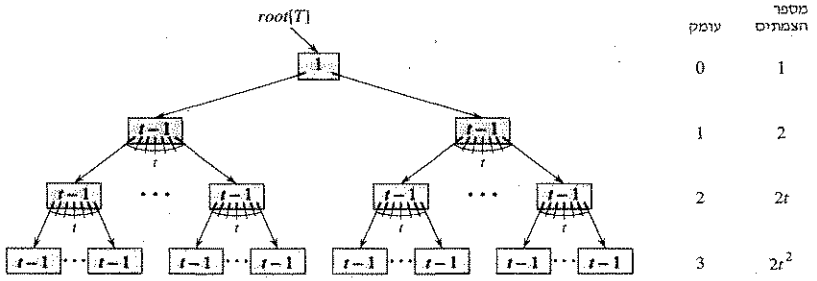
משפט 19.1
אם $n \ge 1$, אזי עבור כל עץ־B T בן n מפתחות שגובהו h ודרגת המינימום שלו $t \ge 2$, מתקיים:

$$h \le \log_t \frac{n+1}{2}$$

הוכחה עבור עץ־B שגובהו h, המספר המינימלי של צמתים מתקבל כאשר השורש מכיל מפתח אחד וכל שאר הצמתים מכילים $t-1$ מפתחות. במקרה זה, ישנם 2 צמתים בעומק 1, $2t$ צמתים בעומק 2, $2t^2$ צמתים בעומק 3, וכן הלאה, עד שבעומק h ישנם $2t^{h-1}$ צמתים. באיור 19.4 מוצג עץ כזה עבור $h=3$. מספר המפתחות n מקיים אפוא את האי־שוויון הבא:

$$\begin{aligned} n &\ge 1 + (t-1)\sum_{i=1}^{h} 2t^{i-1} \\ &= 1 + 2(t-1)\left(\frac{t^h-1}{t-1}\right) \\ &= 2t^h - 1 \end{aligned}$$

ומכאן נובעת ישירות טענת המשפט. ■



איור 19.4 עץ־B בגובה 3 המכיל את המספר המינימלי האפשרי של מפתחות. בכל צומת x רשום $n[x]$.

אנו רואים כאן את עוצמתם של עצי־B בהשוואה לעצים אדומים־שחורים. אף כי בשני המקרים גובהו של העץ גדל כמו $O(\lg n)$ (כזכור, t קבוע), הרי שעבור עצי־B, בסיס הלוגריתם יכול להיות גדול פי כמה. אם כן, עבור רוב הפעולות על עצים, מספר הצמתים הנבחנים בעצי־B קטן פי גורם של $\lg t$ בערך בהשוואה לעצים אדומים־שחורים. מכיוון שכדי לבחון צומת כלשהו בעץ דרושה בדרך כלל גישה לדיסק, מספר הגישות לדיסק מצטמצם במידה משמעותית.

תרגילים

19.1-1
מדוע איננו מרשים דרגת מינימום של $t = 1$?

19.1-2
עבור אילו ערכי t, העץ המוצג באיור 19.1 הוא עץ־B חוקי?

19.1-3
הצג את כל עצי־B החוקיים בעלי דרגת מינימום 2 המייצגים את $\{1, 2, 3, 4, 5\}$.

19.1-4
מצא חסם עליון הדוק על מספר המפתחות שניתן לאחסן בעץ־B שגובהו h כפונקציה של דרגת המינימום t.

19.1-5
תאר את מבנה הנתונים שיתקבל אם כל צומת שחור בעץ אדום־שחור "יבלע" את בניו האדומים וישלב את בניהם עם בניו שלו.

19.2 פעולות בסיסיות על עצי־B

בסעיף זה נציג בפירוט את הפעולות B-TREE-SEARCH, B-TREE-CREATE ו־B-TREE-INSERT. בשגרות המבצעות פעולות אלה נאמץ שתי מוסכמות:

- שורשו של עץ־B נמצא תמיד בזיכרון הראשי, כך שלעולם לא נדרשת פעולת DISK-READ על השורש; אולם יש לבצע DISK-WRITE של השורש כאשר מכניסים שינויים בצומת השורש.
- כל הצמתים שמועברים כפרמטרים חייבים להיות כאלה שכבר התבצעה עליהם קודם לכן פעולת DISK-READ.

השגרות שנציג מבצעות כולן מעבר יחיד מן השורש במורד העץ ללא צורך לחזור אחורה.

חיפוש בעץ־B

חיפוש בעץ־B דומה בעיקרו לחיפוש בעץ חיפוש בינרי, אלא שבמקום לערוך בכל צומת בחירה בינרית, או "דו־כיוונית", באיזה תת־עץ מאלה המסתעפים מן הצומת להמשיך את החיפוש, אנו מבצעים בחירה רב־כיוונית — בהתאם למספר בניו של הצומת. ביתר דיוק, בכל צומת פנימי x אנו מבצעים בחירה $(n[x] + 1)$־כיוונית.

השגרה B-TREE-SEARCH היא הכללה ישירה של השגרה TREE-SEARCH שהוגדרה עבור עצי חיפוש בינריים. B-TREE-SEARCH מקבלת כקלט מצביע לצומת השורש x של תת־עץ ומפתח k שאותו יש לחפש בתת־עץ זה. חיפוש בעץ כולו מתבצע על־ידי הקריאה B-TREE-SEARCH$(root[T], k)$. אם k נמצא בעץ־B זה, B-TREE-SEARCH מחזירה את הזוג הסדור (y, i) המורכב מצומת y ומאינדקס i המקיימים $key_i[y] = k$. אחרת, השגרה מחזירה את הערך NIL.


```
B-TREE-SEARCH(x, k)
1  i ← 1
2  while i ≤ n[x] and k > key_i[x]
3       do i ← i + 1
4  if i ≤ n[x] and k = key_i[x]
5       then return (x, i)
6  if leaf[x]
7       then return NIL
8       else DISK-READ(c_i[x])
9            return B-TREE-SEARCH(c_i[x], k)
```

בשורות 1-3 מוצאים באמצעות חיפוש לינארי את ה־i הקטן ביותר שעבורו $k \le key_i[x]$; אם אין כזה, i מקבל את הערך $n[x] + 1$. בשורות 4-5 בודקים אם כבר נמצא המפתח, ואם כן, השגרה חוזרת. בשורות 6-9 מסיימים את השגרה בכישלון (אם x הוא עלה), או שמבצעים קריאה רקורסיבית להמשך החיפוש בתת־עץ המתאים של x, לאחר ביצוע פעולת DISK-READ הדרושה על בן זה.

באיור 19.1 מדגמת פעולתה של B-TREE-SEARCH; הצמתים הבהירים הם אלה הנבחנים במהלך חיפוש אחר המפתח R.

כמו בשגרה TREE-SEARCH עבור עצי חיפוש בינריים, הצמתים הנבדקים במהלך הרקורסיה יוצרים מסלול מן השורש במורד העץ. מספר דפי הדיסק שהשגרה ניגשת אליהם הוא אפוא $\Theta(h) = \Theta(\log_t n)$, כאשר h הוא גובה העץ־B ו־n הוא מספר המפתחות בעץ. מאחר ש־$n[x] < 2t$, הרי שהזמן לביצוע לולאת ה־**while** בשורות 2-3 עבור כל צומת הוא $O(t)$, וזמן היעי"מ הכולל הוא $O(th) = O(t \log_t n)$.

יצירת עץ־B ריק

כדי לבנות עץ־B T, אנו משתמשים תחילה בשגרה B-TREE-CREATE כדי ליצור שורש ריק, ואז קוראים לשגרה B-TREE-INSERT כדי להוסיף לעץ מפתחות חדשים. שתי השגרות הללו משתמשות בשגרת עזר, ALLOCATE-NODE, המקצה בזמן $O(1)$ דף דיסק אחד שישמש כצומת חדש. נוכל להניח שהצומת הנוצר על־ידי ALLOCATE-NODE אינו דורש פעולת DISK-READ, שכן דף הדיסק עבור צומת זה עדיין אינו מכיל מידע שימושי.

```
B-TREE-CREATE(T)
1  x ← ALLOCATE-NODE()
2  leaf[x] ← TRUE
3  n[x] ← 0
4  DISK-WRITE(x)
5  root[T] ← x
```

ביצועה של B-TREE-CREATE דורש $O(1)$ פעולות על דיסק וזמן יעי"מ של $O(1)$.

פיצול צומת בעץ־B

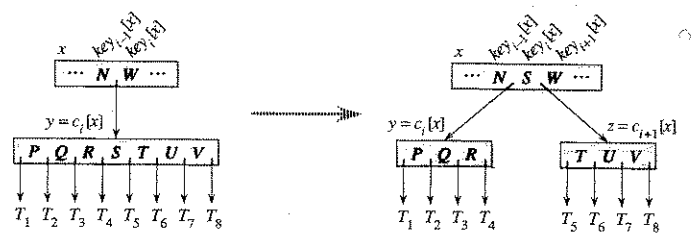
הכנסת מפתח לעץ־B היא פעולה מסובכת בהרבה מהכנסת מפתח לעץ חיפוש בינרי. פעולה בסיסית המתבצעת בעת הכנסה היא **פיצול** של צומת מלא y (המכיל $2t - 1$ מפתחות) סביב **המפתח החציון** (median key) שלו $key_t[y]$ לשני צמתים שכל אחד מהם מכיל $t - 1$ מפתחות.

המפתח החציון מועבר לאב של y — אשר חייב להיות לא מלא לפני פיצול y — כדי לזהות את נקודת ההפרדה בין שני העצים החדשים; אם ל־y אין אב, גובהו של העץ גדל ב־1. פיצול הוא אפוא הדרך שבה גדל עץ־B.

השגרה B-TREE-SPLIT-CHILD מקבלת כקלט צומת פנימי x *לא מלא* (שאנו מניחים כי הוא נמצא בזיכרון הראשי), אינדקס i, וצומת y כך ש־$y = c_i[x]$ הוא בן *מלא* של x. השגרה מפצלת בן זה לשניים, ועורכת את ההתאמות הדרושות ב־x כך שעתה יש לו בן נוסף.

איור 19.5 מדגים תהליך זה. הצומת המלא y מפוצל סביב המפתח החציון שלו S, שמועבר למעלה אל הצומת x — אביו של y. אותם מפתחות ב־y שהם גדולים מהחציון מוצבים בצומת חדש z, שהופך להיות בן חדש של x.

```
B-TREE-SPLIT-CHILD(x, i, y)
 1  z ← ALLOCATE-NODE( )
 2  leaf[z] ← leaf[y]
 3  n[z] ← t − 1
 4  for j ← 1 to t − 1
 5       do key_j[z] ← key_{j+t}[y]
 6  if not leaf[y]
 7    then for j ← 1 to t
 8              do c_j[z] ← c_{j+t}[y]
 9  n[y] ← t − 1
10  for j ← n[x] + 1 downto i + 1
11       do c_{j+1}[x] ← c_j[x]
12  c_{i+1}[x] ← z
13  for j ← n[x] downto i
14       do key_{j+1}[x] ← key_j[x]
15  key_i[x] ← key_t[y]
16  n[x] ← n[x] + 1
17  DISK-WRITE(y)
18  DISK-WRITE(z)
19  DISK-WRITE(x)
```



איור 19.5 פיצול צומת עם $t = 4$. הצומת y מפוצל לשני צמתים, y ו־z, והמפתח החציון S מועבר למעלה אל האב של y.

B-TREE-SPLIT-CHILD פועלת בשיטה ישירה של "חיתוך והדבקה": y, הצומת המפוצל, הוא הבן ה־i של x. במצב ההתחלתי, לצומת y יש $2t$ בנים. לאחר פעולת הפיצול מספר בניו מצטמצם ל־t. הצומת z "מאמץ" את t הבנים הגדולים יותר של y והופך להיות בן חדש של x, הממוקם בדיוק

אחרי y בטבלת הבנים של x. המפתח החציון של y עובר ל־x והופך להיות המפתח המפריד בין y ו־z.

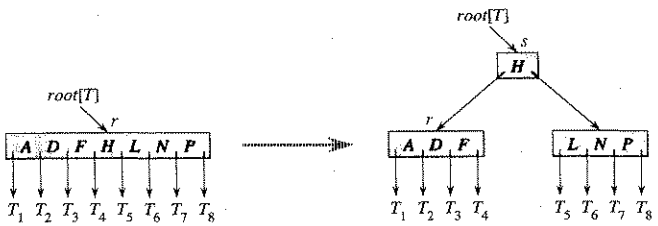
שורות 1-8 יוצרות צומת z ומעבירות אליו את $t-1$ המפתחות הגדולים יותר של y ואת t הבנים המתאימים. שורה 9 מעדכנת את מספר המפתחות ב־y, לבסוף, שורות 10-16 מכניסות את z לעץ כבן של x, מעבירות את המפתח החציון מ־y ל־x כדי שיפריד בין y ו־z, ומעדכנות את מספר המפתחות ב־x. שורות 17-19 כותבות על הדיסק את כל הדפים שחלו בהם שינויים. זמן היע"מ שצורכת B-TREE-SPLIT-CHILD הוא $\Theta(t)$, עקב הלולאות בשורות 4-5 ו־7-8. (הלולאות האחרות מבצעות לכל היותר t איטרציות.)

הכנסת מפתח לעץ־B

הכנסת מפתח k לעץ־B T שגובהו h מתבצעת במעבר יחיד במורד העץ ודורשת $O(h)$ גישות לדיסק. זמן היע"מ הנדרש הוא $O(th) = O(t \log_t n)$. השגרה B-TREE-INSERT משתמשת בשגרה B-TREE-SPLIT-CHILD כדי להבטיח שהרקורסיה לעולם לא תרד לצומת מלא.

```
B-TREE-INSERT(T, k)
 1  r ← root[T]
 2  if n[r] = 2t − 1
 3    then s ← ALLOCATE-NODE( )
 4         root[T] ← s
 5         leaf[s] ← FALSE
 6         n[s] ← 0
 7         c_1[s] ← r
 8         B-TREE-SPLIT-CHILD(s, 1, r)
 9         B-TREE-INSERT-NONFULL(s, k)
10    else B-TREE-INSERT-NONFULL(r, k)
```

שורות 3-9 מטפלות במקרה שבו צומת השורש r מלא: השורש מפוצל וצומת חדש s (שיש לו שני בנים) הופך להיות השורש. פיצול השורש הוא הדרך היחידה להגדלת גובהו של עץ־B. איור 19.6 מדגים מקרה זה. שלא כמו עץ חיפוש בינרי, עץ־B גדל בגובהו בראש העץ ולא בתחתיתו. השגרה מסתיימת בקריאה ל־B-TREE-INSERT-NONFULL לביצוע הכנסת המפתח k לעץ המושרש בשורש הלא־מלא. B-TREE-INSERT-NONFULL יורדת באופן רקורסיבי במורד העץ ככל שנדרש, ובכל שלב היא מבטיחה, באמצעות קריאות מתאימות ל־B-TREE-SPLIT-CHILD, כי הצומת שאליו היא מגיעה במהלך הרקורסיה אינו מלא.

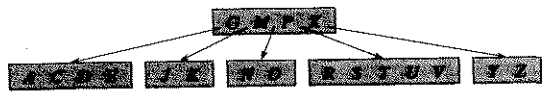


איור 19.6 פיצול השורש כאשר $t = 4$. צומת השורש r מפוצל לשניים ונוצר צומת שורש חדש s. השורש החדש מכיל את המפתח החציון של r, ושני חצאי r הם בניו. כאשר השורש של עץ־B מפוצל, גובהו של העץ גדל ב־1.

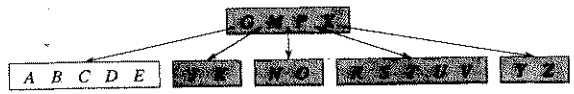
שגרת העזר הרקורסיבית B-TREE-INSERT-NONFULL מכניסה מפתח k לצומת x, שמניחים כי הוא אינו מלא כאשר מתבצעת הקריאה לשגרה. אופן פעולתה של B-TREE-INSERT ואופן פעולתה הרקורסיבי של B-TREE-INSERT-NONFULL מבטיחים את נכונותה של הנחה זו.

```
B-TREE-INSERT-NONFULL(x, k)
 1  i ← n[x]
 2  if leaf[x]
 3     then while i ≥ 1 and k < key_i[x]
 4               do key_{i+1}[x] ← key_i[x]
 5                  i ← i − 1
 6          key_{i+1}[x] ← k
 7          n[x] ← n[x] + 1
 8          DISK-WRITE(x)
 9     else while i ≥ 1 and k < key_i[x]
10               do i ← i − 1
11          i ← i + 1
12          DISK-READ(c_i[x])
13          if n[c_i[x]] = 2t − 1
14             then B-TREE-SPLIT-CHILD(x, i, c_i[x])
15                  if k > key_i[x]
16                     then i ← i + 1
17          B-TREE-INSERT-NONFULL(c_i[x], k)
```

השגרה B-TREE-INSERT-NONFULL פועלת כך: שורות 3-8 מטפלות במקרה שבו x הוא צומת עלה על־ידי הכנסת המפתח k ל־x. אם x אינו צומת עלה, יש להכניס את k לעלה המתאים בתת־עץ המושרש בצומת הפנימי x. במקרה זה, שורות 9-11 קובעות לאיזה מבניו של x תרד הרקורסיה. שורה 13 בודקת אם הרקורסיה תרד לבן מלא, ובמקרה כזה, שורה 14 משתמשת בשגרה B-TREE-SPLIT-CHILD כדי לפצל את הבן הזה לשני בנים לא־מלאים, ושורות 15-16 קובעות איזה משני הבנים האלה הוא הצומת שאליו צריך לרדת עתה. (נשים לב שאין צורך בפעולת DISK-READ$(c_i[x])$ לאחר ששורה 16 מוסיפה 1 ל־i, שכן במקרה זה הרקורסיה תרד לבן שנוצר זה עתה על־ידי B-TREE-SPLIT-CHILD.) תפקידן של שורות 13-16 הוא אפוא להבטיח שבמהלך הרקורסיה השגרה לעולם לא תגיע לצומת מלא. כעת מתבצעת הקריאה הרקורסיבית בשורה 17 כדי להכניס את k לתת־עץ המתאים. איור 19.7 מתאר את המקרים השונים של הכנסת מפתח לעץ־B.



(i) העץ ההתחלתי



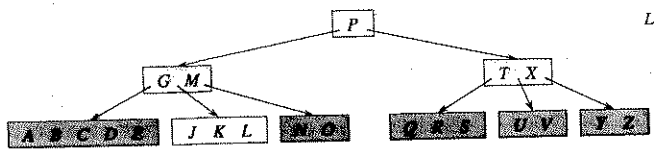
(ii) הכנסת B

איור 19.7 הכנסת מפתחות לעץ־B. דרגת המינימום t של עץ־B זה היא 3, כך שצומת יכול להכיל לכל היותר 5 מפתחות. הצמתים המשתנים במהלך ההכנסה הם בהירים. **(i)** העץ ההתחלתי לדוגמה זו. **(ii)** העץ המתקבל לאחר הכנסת B לעץ ההתחלתי; זוהי הכנסה פשוטה לצומת עלה. **(iii)** העץ המתקבל לאחר הכנסת Q לעץ הקודם. הצומת $RSTUV$ מפוצל לשני צמתים המכילים את RS ואת UV, המפתח T מועבר למעלה אל השורש, והמפתח Q מוכנס לשמאלי משני החצאים (הצומת RS). **(iv)** העץ המתקבל לאחר הכנסת L לעץ הקודם. השורש מלא ולכן הוא מפוצל מיד, וגובה העץ־B גדל ב־1. לאחר מכן, L מוכנס לעלה המכיל את JK. **(v)** העץ המתקבל לאחר הכנסת F לעץ הקודם. הצומת $ABCDE$ מפוצל לפני שמכניסים את F לימני משני החצאים (הצומת DE).

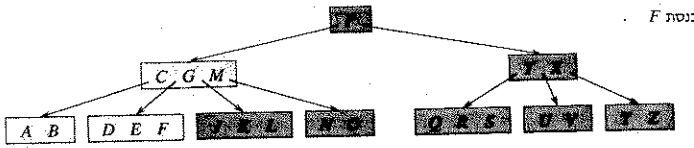
(iii) הכנסת Q



(iv) הכנסת L



(v) הכנסת F



איור 19.7 (המשך)

מספר הגישות לדיסק שמבצעת השגרה B-TREE-INSERT עבור עץ־B בגובה h הוא $O(h)$, שכן רק $O(1)$ פעולות DISK-READ ו־DISK-WRITE מתבצעות בין שתי קריאות לשגרה B-TREE-INSERT-NONFULL. זמן היע"מ הכולל הנדרש הוא $O(th) = O(t \log_t n)$. מכיוון ש־B-TREE-INSERT-NONFULL מבצעת רקורסיית־זנב (tail recursion), ניתן לממש אותה גם על־ידי לולאת **while** ולהיווכח כי מספר הדפים שיש לשמור בזיכרון הראשי בכל זמן נתון הוא $O(1)$.

תרגילים

19.2-1

הצג את תוצאות הכנסתם של המפתחות:

$$F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E$$

לפי הסדר (משמאל לימין) לעץ־B ריק. שרטט רק את תצורות העץ בדיוק לפני שיש לפצל את אחד הצמתים, וכן את התצורה הסופית.

19.2-2

הסבר באילו נסיבות, אם בכלל, מתבצעות פעולות DISK-READ או DISK-WRITE מיותרות במהלך ביצוע קריאה ל־B-TREE-INSERT. (פעולת DISK-READ מיותרת היא פעולת DISK-READ של דף שכבר נמצא בזיכרון הראשי. פעולת DISK-WRITE מיותרת כותבת לדיסק דף מידע זהה לדף שכבר מאוחסן שם.)

19.2-3

הסבר כיצד ניתן למצוא את המפתח המינימלי המאוחסן בעץ־B וכיצד ניתן למצוא את המפתח הקודם למפתח נתון המאוחסן בעץ־B.

19.2-4

נניח שמכניסים את המפתחות $\{1, 2, \ldots, n\}$ לעץ־B ריק שדרגת המינימום שלו 2. כמה צמתים יש בעץ־B הסופי?

19.2-5

מאחר שבצומתי עלים לא דרושים מצביעים לצומתי בנים, הרי שבעיקרון הם יכולים להשתמש בערך t שונה (גדול יותר) מאשר צמתים פנימיים עבור אותו גודל של דף דיסק. הראה כיצד ניתן לשנות את השגרות ליצירת עץ־B ולהכנסת מפתח לעץ־B כך שיטפלו בגרסה זו של העץ.

19.2-6

נניח שמממשים את B-TREE-SEARCH כך שהיא משתמשת בחיפוש בינרי ולא בחיפוש לינארי בתוך כל צומת. הראה שבמקרה כזה, זמן היעי"מ הנדרש הוא $O(\lg n)$, בלי תלות באופן שבו בוחרים את t כפונקציה של n.

19.2-7

נניח שחומרת הדיסק מאפשרת לנו לבחור את גודלו של דף דיסק באופן שרירותי, אולם הזמן הדרוש לקריאת דף דיסק הוא $a + bt$, כאשר a ו־b הם קבועים נתונים ו־t היא דרגת המינימום של עץ־B המשתמש בדפים בגודל שבחרנו. תאר כיצד יש לבחור t כך שימזער (בקירוב) את זמן החיפוש בעץ־B. הצע ערך אופטימלי עבור t במקרה שבו $a = 30$ מילי־שניות ו־$b = 40$ מיקרו־שניות.

19.3 מחיקת מפתח מעץ־B

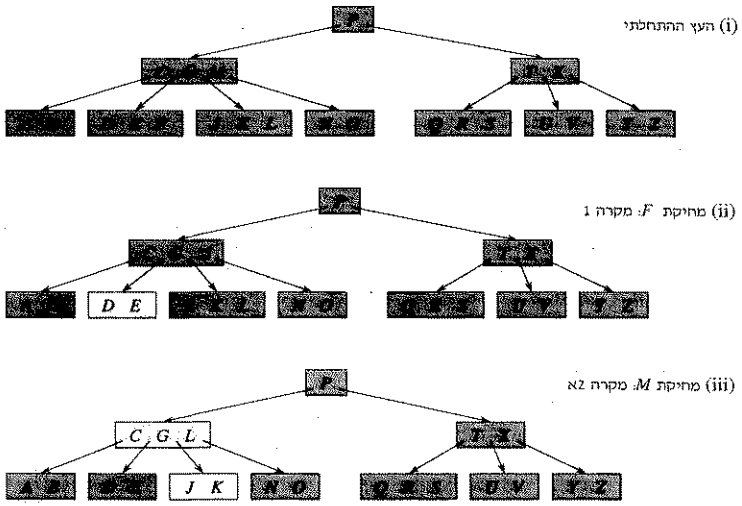
מחיקה מעץ־B מקבילה להכנסה אם כי היא מסובכת במעט. אנו נתאר בקווים כלליים כיצד היא פועלת ולא נביא פסאודו־קוד מלא.

נניח כי השגרה B-TREE-DELETE מתבקשת למחוק את המפתח k מהתת־עץ המושרש ב־x. המבנה של B-TREE-DELETE מבטיח שבכל פעם שהיא נקראת באופן רקורסיבי עם צומת x, מספר המפתחות ב־x הוא לפחות דרגת המינימום t. יש לשים לב כי תנאי זה דורש מפתח אחד יותר מהמינימום הנדרש בתנאים הרגילים של עץ־B, כך שלעתים יש צורך להעביר מפתח לצומת בן לפני שהרקורסיה יכולה לרדת לבן זה. תנאי חזק יותר זה מאפשר למחוק מפתח במעבר יחיד במורד העץ ללא צורך "לחזור אחורה" (עם יוצא מן הכלל אחד שיוסבר בהמשך). את המפרט הבא עבור מחיקה מעץ־B יש לפרש מתוך הבנה שאם קודה אי פעם שצומת השורש x הופך להיות צומת פנימי שאינו מכיל מפתחות, אזי x נמחק, ו־$c_1[x]$, בנו היחיד של x, הופך להיות השורש החדש של העץ, גובהו של העץ קטן ב־1, וכך משתמרת התכונה שהשורש של עץ־B מכיל לפחות מפתח אחד (אלא אם כן העץ ריק).

איור 19.8 מדגים את המקרים השונים של מחיקת מפתח מעץ־B.

1. אם המפתח k נמצא בצומת x ו־x הוא עלה, מחק את המפתח k מ־x.
2. אם המפתח k נמצא בצומת x ו־x הוא צומת פנימי, בצע את הפעולות הבאות:
   א. אם הבן y, הקודם ל־k בצומת x, מכיל לפחות t מפתחות, מצא את הקודם k' ל־k בתת־עץ המושרש ב־y. מחק באופן רקורסיבי את k', והחלף את k ב־k' בצומת x. (מציאת k' ומחיקתו יכולות להתבצע במעבר יחיד במורד העץ.)

   ב. באופן סימטרי, אם הבן z, העוקב ל־k בצומת x, מכיל לפחות t מפתחות, מצא את העוקב k' של k בתת־עץ המושרש ב־z. מחק באופן רקורסיבי את k', והחלף את k ב־k' בצומת x. (מציאת k' ומחיקתו יכולות להתבצע במעבר יחיד במורד העץ.)
   ג. אחרת, אם y וגם z מכילים רק $t-1$ מפתחות כל אחד, מזג את k ואת כל z לתוך y, כך ש־x יאבד הן את k והן את המצביע ל־z, ו־y יכיל עתה $2t-1$ מפתחות. לאחר מכן שחרר את z ומחק באופן רקורסיבי את k מ־y.
3. אם k אינו נמצא בצומת פנימי x, מצא את השורש $c_i[x]$ של התת־עץ שחייב להכיל את k אם k אכן נמצא בעץ. אם $c_i[x]$ מכיל רק $t-1$ מפתחות, בצע את צעד 3א או 3ב על פי הנחוץ כדי להבטיח שהרקורסיה תרד לצומת שמכיל לפחות t מפתחות. לאחר מכן סיים על־ידי קריאה רקורסיבית עם הבן המתאים של x.
   א. אם $c_i[x]$ מכיל רק $t-1$ מפתחות אבל יש לו אח שמכיל t מפתחות, תן ל־$c_i[x]$ מפתח נוסף על־ידי העברת מפתח מ־x למטה אל $c_i[x]$, העברת מפתח מאח סמוך, ימני או שמאלי, של $c_i[x]$ למעלה אל x, והעברת הבן המתאים מן האח אל $c_i[x]$.
   ב. אם $c_i[x]$ וכל אחד מאחיו מכילים $t-1$ מפתחות, מזג את c_i עם אחד מאחיו, פעולה הכרוכה בהעברת מפתח מ־x למטה אל הצומת הממוזג החדש כדי שיהיה המפתח החציון של צומת זה.



איור 19.8 מחיקת מפתחות מעץ־B. דרגת המינימום של עץ־B זה היא $t = 3$, כך שצומת (שאינו השורש) אינו יכול להכיל פחות מ־2 מפתחות. הצמתים שחלים בהם שינויים הם בהירים. **(i)** העץ־B מאיור 19.7(v). **(ii)** מחיקת F. זהו מקרה 1: מחיקה פשוטה מצומת עלה. **(iii)** מחיקת M. זהו מקרה 2א: L, הקודם ל־M, מועבר למעלה ותופס את מקומו של M. **(iv)** מחיקת G. זהו מקרה 2ג: G נדחף למטה ליצירת הצומת $DEGJK$, ואז G נמחק מצומת עלה זה (מקרה 1). **(v)** מחיקת D. זהו מקרה 3ב: הרקורסיה אינה יכולה לרדת לצומת CL מכיוון שהוא מכיל רק 2 מפתחות, ולכן P נדחף למטה וממוזג עם CL ו־TX ליצירת הצומת $CLPTX$; לאחר מכן D נמחק מן העלה (מקרה 1). **(v')** לאחר (iv), השורש נמחק וגובה העץ קטן ב־1. **(vi)** מחיקת B. זהו מקרה 3א: C מועבר ותופס את מקומו של B, ו־E מועבר ותופס את מקומו של C.

כתוב פסאודו-קוד עבור B-TREE-DELETE.

19.3-2 הצג את תוצאות מחיקתם של C ו-P לפי הסדר, מן העץ שבאיור 19.8(vi).

תרגילים

19.3-1 אף כי שערוך זה נראית מסובכת, ביצועה בדרך כלל עבור עץ-B בגובה h שבו בין שתי הפעולות רקורסיביות של השגרה מתבצעות רק $O(1)$ פעולות DISK-READ. זמן הריצה הנדרש הוא $O(t \log_t n) = O(th)$.

19.3-2 כתוב פסאודו-קוד עבור B-TREE-DELETE.

19.3-1 הצג את תוצאות מחיקתם של C ו-P לפי הסדר, מן העץ שבאיור 19.8(vi).

תרגילים

19.3-1 אף כי שערוך זה נראית מסובכת, ביצועה בדרך כלל עבור עץ-B בגובה h שבו בין שתי הפעולות רקורסיביות של השגרה מתבצעות רק $O(1)$ פעולות DISK-READ. זמן הריצה הנדרש הוא $O(t \log_t n) = O(th)$.

19.3-2 הצג את תוצאות מחיקתם של C ו-P לפי הסדר, מן העץ שבאיור 19.8(vi).

איור 19.8 (המשך)

מחיקת B מעץ B המופיע בתרגיל 19.3-1, תגרום לכך שהמפתח B יוסר מצומת עלה, ולכך שהמפתח C יוסר מצומת פנימי. מחיקת C מעץ B המופיע בתרגיל 19.3-2 תגרום לכך שהמפתח C יוסר מצומת עלה, ולכך שהמפתח P יוסר מצומת פנימי.

איור 19.8 (המשך)

מחיקת B מעץ B המופיע בתרגיל 19.3-1, תגרום לכך שהמפתח B יוסר מצומת עלה, ולכך שהמפתח C יוסר מצומת פנימי. מחיקת C מעץ B המופיע בתרגיל 19.3-2 תגרום לכך שהמפתח C יוסר מצומת עלה, ולכך שהמפתח P יוסר מצומת פנימי.

בעיות

19-1 מחסנית על התקני אחסון חיצוניים

לפעמים נדרש לתחזק מחסנית גדולה מכדי שתוכל להיות כולה בזיכרון הראשי של המחשב. אנו יכולים לממש פעולות PUSH ו-POP היעילות בזמן הריצה המשוערך בעזרת שיטה שבה אנו מחזיקים בזיכרון הראשי חלק מהמחסנית, המורכב מדף אחד בגודל m מילים. נניח שאנו יכולים לקרוא דף אחד או לכתוב דף אחד אל הזיכרון החיצוני בזמן $O(1)$, ושמספר המילים הנדרשות לאחסון המחסנית הוא n. נניח גם ש-$n \geq m$ ושהמספר m זוגי.

א. מהו מספר הגישות הנדרשות לדיסק הדרושות במקרה הגרוע עבור n פעולות על מחסנית? מהו זמן הריצה?

ב. נתאר כיצד ניתן לנהל את דפי המחסנית כך שמספר הגישות לדיסק עבור כל פעולת מחסנית יהיה $O(1/m)$, וזמן הריצה לפעולות על מחסנית יהיה $O(1)$.

2-19 איחוד ופיצול של עצי 2-3-4

כתוב פסאודו-קוד עבור הפעולה *join* (איחוד) המקבלת שתי קבוצות דינמיות S' ו-S'' ואבר x כך ש-$x' < x < x''$ לכל $x' \in S'$ ו-$x'' \in S''$. הפעולה מחזירה קבוצה $S = S' \cup \{x\} \cup S''$. כתוב גם פסאודו-קוד עבור הפעולה *split*, המקבלת קבוצה דינמית $S = S' \cup \{x\} \cup S''$ ואבר $x \in S$ ומחזירה את הקבוצה $S' - \{x\}$ ו-S''.

שהמפתחות שלהם קטנים מ־$key[x]$ וקבוצה S'' המורכבת מכל האיברים ב־$S - \{x\}$ שהמפתחות שלהם גדולים מ־$key[x]$. בבעיה זו נחקור כיצד מממשים פעולות אלה על עצי 2-3-4. מטעמי נוחות נניח כי האיברים מורכבים ממפתחות בלבד וכל ערכי המפתחות שונים זה מזה.

א. הראה כיצד ניתן לנהל עבור כל צומת x בעץ 2-3-4 שדה $height[x]$ המכיל את גובה תת־עץ המושרש ב־x. ודא שהמימוש שלך אינו משפיע על זמני הריצה האסימפטוטיים של חיפוש, הכנסה ומחיקה.

ב. הראה כיצד ניתן לממש את הפעולה ***join***: בהינתן שני עצי 2-3-4 T' ו־T'' ומפתח k, הפעולה ***join*** צריכה להתבצע בזמן $O(|h' - h''|)$, כאשר h' ו־h'' הם גובהי העצים T' ו־T'', בהתאמה.

ג. נתון המסלול p מהשורש של עץ 2-3-4 T למפתח נתון k, הקבוצה S' של מפתחות ב־T הקטנים מ־k, והקבוצה S'' של מפתחות ב־T הגדולים מ־k. הראה ש־p מפרק את S' לקבוצה של עצים $\{T'_0, T'_1, \ldots, T'_m\}$ וקבוצה של מפתחות $\{k'_1, k'_2, \ldots, k'_m\}$ כך שעבור $i = 1, 2, \ldots, m$, מתקיים $y < k'_i < z$ לכל המפתחות $y \in T'_{i-1}$ ו־$z \in T'_i$. מהו הקשר בין הגבהים של T'_{i-1} ושל T'_i? תאר כיצד p מפרק את S'' לקבוצות של עצים ושל מפתחות.

ד. הראה כיצד ניתן לממש את הפעולה ***split*** על T. השתמש בפעולה ***join*** כדי להרכיב מהמפתחות ב־S' עץ 2-3-4 יחיד T' ומהמפתחות ב־S'' עץ 2-3-4 יחיד T''. זמן הריצה של פעולת הפיצול צריך להיות $O(\lg n)$, כאשר n הוא מספר המפתחות ב־T. (**רמז:** עלויות הפיצולים צריכות ליצור טור טלסקופי.)

הערות ביבליוגרפיות לפרק 19

אצל Knuth [123], Aho, Hopcroft & Ullman [4] ו־Sedgewick [175] אפשר למצוא דיונים נוספים במבנים של עצים מאוזנים ובעצי־B. Comer [48] מביא סקירה מקיפה על עצי־B. Guibas & Sedgewick [93] דנים בקשר בין מבנים שונים של עצים מאוזנים, כולל עצים אדומים־שחורים ועצי 2-3-4.

ב־1970 המציא J. E. Hopcroft עצי 2-3, שהקדימו את עצי־B ועצי 2-3-4. בעצי 2-3, לכל צומת פנימי יש שני בנים או שלושה בנים. עצי־B הוצעו בידי Bayer ו־McCreight ב־1972 [18]; הם לא הסבירו מדוע בחרו לכנותם עצי־B.

20 ערימות בינומיות

בפרק זה ובפרק 21 נציג מבני נתונים המוכרים בשם **ערימות מתמזגות** (mergeable heaps), ואשר תומכים בחמש הפעולות הבאות:

MAKE-HEAP() — יוצרת ומחזירה ערימה חדשה שאינה מכילה איברים כלל.

INSERT(H, x) — מכניסה לערימה H את הצומת x אשר השדה key שלו כבר מכיל ערך.

MINIMUM(H) — מחזירה מצביע לצומת בעל המפתח המינימלי בערימה H.

EXTRACT-MIN(H) — מוחקת את הצומת בעל המפתח המינימלי מן הערימה H ומחזירה מצביע לצומת זה.

UNION(H_1, H_2) — יוצרת ומחזירה ערימה חדשה המכילה את כל צומתי הערימות H_1 ו־H_2. הערימות H_1 ו־H_2 "מושמדות" במהלך ביצוע הפעולה.

נוסף על אלה, מבני הנתונים הנדונים בפרקים אלה תומכים בשתי הפעולות הבאות:

DECREASE-KEY(H, x, k) — מציבה בשדה המפתח של צומת x השייך לערימה H מפתח חדש k, שמניחים כי ערכו אינו גדול מערך המפתח הנוכחי.

DELETE(H, x) — מוחקת את הצומת x מן הערימה H.

כפי שמראה הטבלה שבאיור 20.1, אם אין לנו צורך בפעולה UNION, ערימות בינריות רגילות, כמו אלה המשמשות במיון־ערימה (פרק 7), פועלות היטב: כל הפעולות, פרט ל־UNION, רצות על ערימות בינריות במקרה הגרוע בזמן $O(\lg n)$ (או בזמן טוב יותר). אולם, אם יש צורך לתמוך בפעולה UNION, הביצועים של ערימות בינריות אינם טובים. ביצוע UNION על־ידי שרשור שני המערכים המכילים את הערימות הבינריות שיש למזג ואז הרצת HEAPIFY, יארך במקרה הגרוע זמן $\Theta(n)$.

בפרק זה נבחן "ערימות בינומיות" אשר החסמים על זמני הריצה של הפעולות עליהן במקרה הגרוע מוצגים גם הם באיור 20.1. בפרט, הפעולה UNION למיזוג שתי ערימות בינומיות המכילות יחד n איברים רצה בזמן $O(\lg n)$ בלבד.

בפרק 21 נחקור ערימות פיבונצ'י, שבהן החסמים על זמני הריצה של חלק מהפעולות טובים אף יותר. עם זאת, יש לשים לב לכך שזמני הריצה עבור ערימות פיבונצ'י המובאים באיור 20.1 הם חסמי זמן לשיעורין ולא חסמים לכל פעולה במקרה הגרוע.

השגרה	ערימה בינרית (עלות במקרה הגרוע)	ערימה בינומית (עלות במקרה הגרוע)	ערימת פיבונצ'י (עלות לשיערוך)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$O(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$O(\lg n)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$O(\lg n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$

איור 20.1 זמני הריצה של פעולות על שלושה מימושים של ערימות מתמזגות. מספר האיברים בערימה (או בערימות) בעת ביצועה של פעולה מסומן ב־n.

בפרק זה נתעלם מהבעיות של הקצאת צמתים לפני הכנסה ושחרור צמתים לאחר מחיקה. אנו מניחים שהקוד הקורא לשגרות הפועלות על ערימות מטפל בפרטים אלה.

ערימות בינריות, ערימות בינומיות וערימות פיבונצ'י הן כולן בלתי יעילות בתמיכתן בפעולה SEARCH; מציאתו של צומת בעל מפתח נתון עלולה לארוך זמן רב למדי. מסיבה זו, פעולות כמו DECREASE-KEY ו־DELETE המתייחסות לצומת נתון דורשות מצביע לצומת זה כחלק מן הקלט שלהן. ביישומים רבים דרישה זו אינה יוצרת שום בעיה.

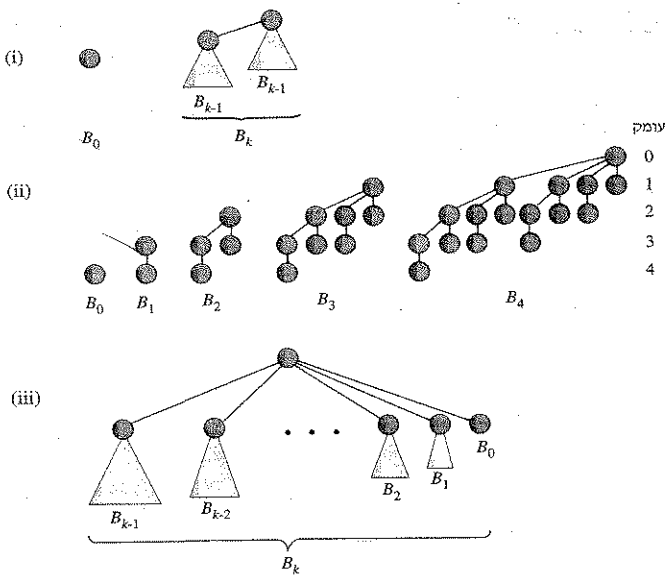
בסעיף 20.1 נגדיר ערימות בינומיות לאחר שנגדיר תחילה את העצים הבינומיים המרכיבים אותן. כמו כן נתאר ייצוג מסוים של ערימות בינומיות. סעיף 20.2 יראה כיצד ניתן לממש פעולות על ערימות בינומיות בחסמי הזמן המובאים באיור 20.1.

20.1 עצים בינומיים וערימות בינומיות

ערימה בינומית היא אוסף של עצים בינומיים; לכן נפתח סעיף זה בהגדרת עצים בינומיים ובהוכחת כמה תכונות מפתח. לאחר מכן נגדיר ערימות בינומיות ונראה כיצד ניתן לייצגן.

20.1.1 עצים בינומיים

עץ בינומי B_k (binomial tree) הוא עץ סדור (ראה סעיף 5.5.2) המוגדר באופן רקורסיבי. כפי שאפשר לראות באיור 20.2(i), העץ הבינומי B_0 מורכב מצומת יחיד. העץ הבינומי B_k מורכב משני עצים בינומיים B_{k-1} **המקושרים** (linked) זה לזה: השורש של עץ אחד הוא הבן השמאלי ביותר של השורש של העץ האחר. באיור 20.2(ii) מוצגים העצים הבינומיים B_0 עד B_4.



איור 20.2 **(i)** ההגדרה הרקורסיבית של עץ בינומי B_k. המשולשים מייצגים תת־עצים שורשיים. **(ii)** העצים הבינומיים B_0 עד B_4. ב־B_4 רשומים עומקי הצמתים. **(iii)** דרך אחרת להסתכלות על העץ הבינומי B_k.

תכונות אחדות של עצים בינומיים נתונות בלמה הבאה.

למה 20.1 (תכונותיהם של עצים בינומיים)
העץ הבינומי B_k מקיים:

1. הוא מכיל 2^k צמתים;
2. גובהו k;
3. ישנם בדיוק $\binom{k}{i}$ צמתים בעומק i, עבור $i = 0, 1, \ldots, k$;
4. דרגתו של השורש היא k, והיא גדולה מדרגתו של כל צומת אחר; יתר על כן, אם בניו של השורש ממוספרים משמאל לימין על־ידי $k-1, k-2, \ldots, 0$, אזי הבן i הוא השורש של תת־עץ B_i.

הוכחה ההוכחה היא באינדוקציה על k. לכל תכונה, הבסיס הוא העץ הבינומי B_0. ההוכחה ש־B_0 מקיים כל אחת מן התכונות היא טריוויאלית. עבור צעד האינדוקציה ננית שהלמה מתקיימת עבור B_{k-1}.

1. העץ הבינומי B_k מורכב משני עותקים של B_{k-1}, ולכן הוא מכיל $2^{k-1} + 2^{k-1} = 2^k$ צמתים.
2. בשל האופן שבו מקושרים שני העותקים של B_{k-1} ליצירת B_k, העומק המקסימלי של צומת ב־B_k גדול ב־1 מן העומק המקסימלי של צומת ב־B_{k-1}. מהנחת האינדוקציה נובע שהעומק המקסימלי ב־B_k הוא $(k-1) + 1 = k$.

3. יהי $D(k, i)$ מספר הצמתים בעומק i בעץ בינומי B_k. מאחר ש־B_k מורכב משני עותקים של B_{k-1} המקושרים יחדיו, צומת בעומק i ב־B_{k-1} מופיע ב־B_k פעם אחת בעומק i ופעם אחת בעומק $i+1$. במילים אחרות, מספר הצמתים בעומק i ב־B_k שווה למספר הצמתים בעומק i ב־B_{k-1} ועוד מספר הצמתים בעומק $i-1$ ב־B_{k-1}. לפיכך:

$$\begin{aligned} D(k, i) &= D(k-1, i) + D(k-1, i-1) \\ &= \binom{k-1}{i} + \binom{k-1}{i-1} \\ &= \binom{k}{i} \end{aligned}$$

השוויון השני נובע מהנחת האינדוקציה, והשלישי — מתרגיל 6.1-7.

4. הצומת היחיד שדרגתו ב־B_k גדולה מדרגתו ב־B_{k-1} הוא השורש, אשר במעבר מ־B_{k-1} ל־B_k נוסף לו בן. מכיוון שדרגתו של השורש ב־B_{k-1} היא $k-1$, דרגת השורש ב־B_k היא k. כעת, על פי הנחת האינדוקציה, וכפי שאפשר לראות באיור 20.2(iii), בניו של השורש של B_{k-1}, משמאל לימין, הם השורשים של $B_0, B_{k-2}, B_{k-3}, \ldots, B_0$. לכן, כאשר B_{k-1} מקושר ל־B_{k-1}, בניו של השורש של העץ B_k המתקבל הם שורשים של $B_{k-1}, B_{k-2}, \ldots, B_0$. ■

מסקנה 20.2

הדרגה המקסימלית של צומת כלשהו בעץ בינומי n־צומתי היא $\lg n$.

הוכחה הטענה נובעת ישירות מתכונות 1 ו־4 בלמה 20.1. ■

המונח "עץ בינומי" נובע מתכונה 3 בלמה 20.1, שהרי הביטויים $\binom{k}{i}$ הם המקדמים הבינומיים. בתרגיל 20.1-3 מובאת הצדקה נוספת למונח זה.

20.1.2 ערימות בינומית

ערימה בינומית (binomial heap) H היא קבוצה של עצים בינומיים המקיימת את **תכונות הערימה הבינומית** (binomial-heap properties) הבאות:

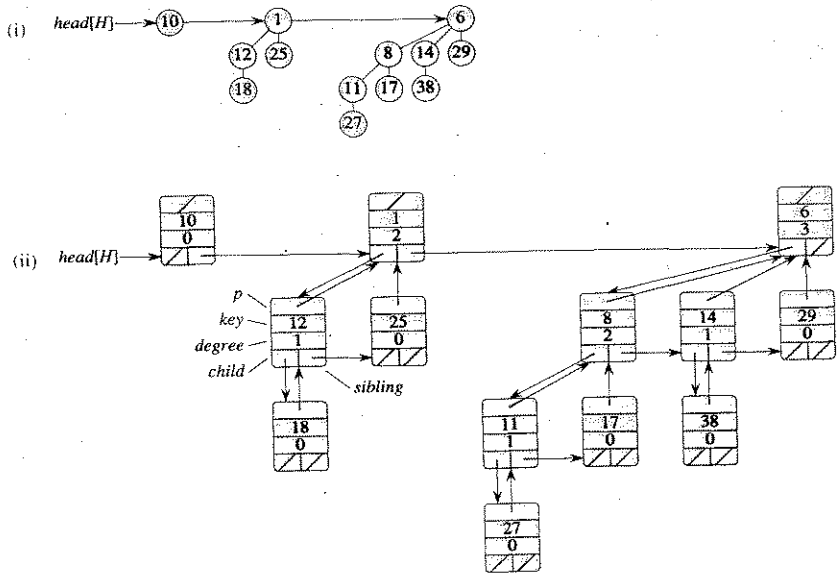
1. בכל עץ בינומי ב־H מתקיים **סדר הערימה** (heap order): המפתח של צומת גדול מהמפתח של אביו או שווה לו.

2. ב־H יש לכל היותר עץ בינומי אחד שהשורש שלו הוא בעל דרגה נתונה.

מן התכונה הראשונה אנו למדים שמפתחו של שורש של עץ המקיים את סדר הערימה הוא המפתח הקטן ביותר בעץ.

מן התכונה השנייה נובע שערימה בינומית n־צומתית H מורכבת מ־$\lfloor \lg n \rfloor + 1$ עצים בינומיים לכל היותר. כדי לראות מדוע, נשים לב לכך שהייצוג הבינרי של n הוא בן $\lfloor \lg n \rfloor + 1$ סיביות, נאמר, $\langle b_0, \ldots, b_{\lfloor \lg n \rfloor - 1}, b_{\lfloor \lg n \rfloor} \rangle$, כך ש־$n = \sum_{i=0}^{\lfloor \lg n \rfloor} b_i 2^i$. מתכונה 1 בלמה 20.1 נובע אפוא שהעץ הבינומי B_i מופיע ב־H אם ורק אם $b_i = 1$. לפיכך, ערימה בינומית H מכילה לכל היותר $\lfloor \lg n \rfloor + 1$ עצים בינומיים.

באיור 20.3(i) מוצגת ערימה בינומית H בעלת 13 צמתים. הייצוג הבינרי של 13 הוא $\langle 1101 \rangle$, ו־H מורכבת מן העצים הבינומיים המקיימים את סדר הערימה B_3, B_2, B_0, המכילים 8, 4 ו־1 צמתים, בהתאמה, ובסך הכל 13 צמתים.



איור 20.3 ערימה בינומית בעלת $n = 13$ צמתים. **(i)** הערימה מורכבת מן העצים הבינומיים B_0, B_2 ו־B_3 המכילים, בהתאמה, 1, 4 ו־8 צמתים, ובסך הכל 13 צמתים. מכיוון שכל עץ בינומי מקיים את סדר הערימה, המפתח של צומת כלשהו אינו קטן מהמפתח של אביו. כמו כן מוצגת רשימת השורשים, שהיא רשימה מקושרת של שורשי העצים הבינומיים בסדר עולה של דרגות. **(ii)** ייצוג מפורט יותר של ערימה בינומית H. כל עץ בינומי מאוחסן בייצוג של "בן שמאלי ואח ימני", ובכל צומת מאוחסנת גם דרגתו.

ייצוג ערימות בינומיות

כפי שאפשר לראות באיור 20.3(ii), כל עץ בינומי בתוך ערימה בינומית מאוחסן בייצוג של "בן שמאלי ואח ימני" (ראה סעיף 11.4). כל צומת מכיל שדה key וכן את המידע הנלווה הנדרש ביישום. בנוסף, כל צומת x מכיל את המצביעים הבאים: $p[x]$ — לאביו, $child[x]$ — לבן השמאלי ביותר שלו, ו־$sibling[x]$ — לאח של x הנמצא בדיוק מימינו. אם צומת x הוא שורש, אזי $p[x] = \text{NIL}$. אם לצומת x אין בנים, אזי $child[x] = \text{NIL}$. אם x הוא הבן הימני ביותר של אביו, אזי $sibling[x] = \text{NIL}$. כל צומת x מכיל גם את השדה $degree[x]$, המכיל את מספר הבנים של x.

איור 20.3 גם מראה שהשורשים של העצים הבינומיים המרכיבים ערימה בינומית מאורגנים ברשימה מקושרת, שנקרא לה **רשימת השורשים** (root list). השורשים מופיעים ברשימה בסדר עולה ממש של דרגות. מן התכונה השנייה של הערימה הבינומית נובע שבערימה בינומית n־צומתית, דרגות השורשים הן תת־קבוצה של $\{0, 1, \ldots, \lfloor \lg n \rfloor\}$. לשדה $sibling$ יש משמעות שונה בשורשים מאשר בצמתים שאינם שורשים. אם x הוא שורש, אזי $sibling[x]$ מצביע על השורש הבא ברשימת השורשים. (כרגיל, $sibling[x] = \text{NIL}$ אם x הוא השורש האחרון ברשימת השורשים.)

בהינתן ערימה בינומית H, הגישה אליה מתבצעת באמצעות השדה $head[H]$, שהוא פשוט מצביע לשורש הראשון ברשימת השורשים של H. אם ערימה בינומית H אינה מכילה איברים, אזי $head[H] = \text{NIL}$.

תרגילים

20.1-1

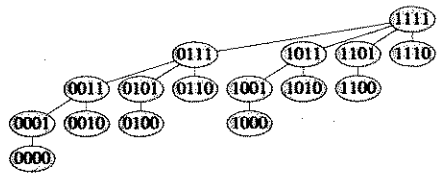
נניח כי x הוא צומת בעץ בינומי השייך לערימה בינומית, ונניח כי $sibling[x] \neq \text{NIL}$. אם x אינו שורש, כיצד היית משווה בין $degree[sibling[x]]$ לבין $degree[x]$? מה קורה כאשר x הוא שורש?

20.1-2

אם x הוא צומת שאינו שורש בעץ בינומי השייך לערימה בינומית, כיצד תשווה בין $degree[p[x]]$ לבין $degree[x]$?

20.1-3

נניח שאנו מסמנים את הצמתים בעץ בינומי B_k במספרים סידוריים בינריים בעת סריקה סופית של העץ, כמתואר באיור 20.4. נתבונן בצומת x בעומק i שמספרו l, ויהי $j = k - i$. הראה שבייצוג הבינרי של x יש j סיביות שערכן 1. כמה מחרוזות בינריות באורך k מכילות בדיוק j סיביות שערכן 1? הראה שדרגתו של x שווה למספר הסיביות שערכן 1 המופיעות מימין ל־0 הימני ביותר בייצוג הבינרי של l.



איור 20.4 העץ הבינומי B_4 כשצומתיו מסומנים במספרים סידוריים בינריים בסריקה סופית של העץ.

20.2 פעולות על ערימות בינומיות

בסעיף זה נראה כיצד ניתן לבצע פעולות על ערימות בינומיות בחסמי הזמן המופיעים באיור 20.1. אנו נוכיח את החסמים העליונים בלבד; את הוכחת החסמים התחתונים אנו משאירים כתרגיל לקורא (תרגיל 20.2-10).

יצירת ערימה בינומית חדשה

כדי ליצור ערימה בינומית ריקה, השגרה MAKE-BINOMIAL-HEAP פשוט מקצה ומחזירה עצם H, כך ש־$head[H] = \text{NIL}$. זמן הריצה הוא $\Theta(1)$.

מציאת המפתח המינימלי

השגרה BINOMIAL-HEAP-MINIMUM מחזירה מצביע לצומת בעל המפתח המינימלי בערימה בינומית n־צומתית H. מימוש זה מניח שלא קיימים מפתחות שערכם ∞. (ראה תרגיל 20.2-5.)

```
BINOMIAL-HEAP-MINIMUM(H)
1  y ← NIL
2  x ← head[H]
3  min ← ∞
4  while x ≠ NIL
5      do if key[x] < min
6          then min ← key[x]
7               y ← x
8       x ← sibling[x]
9  return y
```

מכיוון שערימה בינומית מקיימת את סדר הערימה, המפתח המינימלי חייב להימצא באחד השורשים. השגרה BINOMIAL-HEAP-MINIMUM בודקת את כל השורשים, שמספרם הוא לכל היותר $\lfloor \lg n \rfloor + 1$, ושומרת ב־$min$ את המינימום הנוכחי וב־y מצביע למינימום הנוכחי. כאשר היא נקראת עם הערימה הבינומית שבאיור 20.3, השגרה BINOMIAL-HEAP-MINIMUM מחזירה מצביע לצומת בעל המפתח 1.

מאחר שיש לבדוק לכל היותר $\lfloor \lg n \rfloor + 1$ שורשים, זמן הריצה של BINOMIAL-HEAP-MINIMUM הוא $O(\lg n)$.

איחוד שתי ערימות בינומיות

פעולת האיחוד של שתי ערימות בינומיות משמשת כתת־שגרה במרבית הפעולות הנותרות. השגרה BINOMIAL-HEAP-UNION מקשרת וחוזרת ומקשרת עצים בינומיים שהשורשים שלהם בעלי אותה דרגה. השגרה BINOMIAL-LINK שלהלן מקשרת את העץ B_{k-1} המושרש בצומת y לעץ B_{k-1} המושרש בצומת z; כלומר, עושה את z לאב של y. בצורה זו הצומת z הופך להיות השורש של עץ B_k.

```
BINOMIAL-LINK(y, z)
1  p[y] ← z
2  sibling[y] ← child[z]
3  child[z] ← y
4  degree[z] ← degree[z] + 1
```

השגרה BINOMIAL-LINK הופכת את הצומת y לראש החדש של הרשימה המקושרת של בניו של הצומת z בזמן $O(1)$. היא פועלת היטב מכיוון שהייצוג של בן־שמאלי־ואח־ימני של כל עץ בינומי מתאים לתכונת הסדר של העץ: הבן השמאלי ביותר של עץ B_k הוא השורש של עץ B_{k-1}.

השגרה BINOMIAL-HEAP-UNION שלהלן מאחדת את הערימות הבינומיות H_1 ו־H_2 ומחזירה את הערימה המתקבלת. במהלך האיחוד היא הורסת את הייצוגים של H_1 ו־H_2. מלבד BINOMIAL-LINK, שגרת האיחוד משתמשת בשגרת עזר BINOMIAL-HEAP-MERGE אשר ממזגת את רשימות השורשים של H_1 ו־H_2 לרשימה מקושרת אחת הממוינת בסדר מונוטוני עולה של דרגות. השגרה BINOMIAL-HEAP-MERGE, שאת כתיבת הפסאודו־קוד שלה השארנו כתרגיל 20.2-2, דומה לשגרה MERGE מסעיף 1.3.1.

```
BINOMIAL-HEAP-UNION(H1, H2)
 1  H ← MAKE-BINOMIAL-HEAP( )
 2  head[H] ← BINOMIAL-HEAP-MERGE(H1, H2)
 3  free the objects H1 and H2 but not the lists they point to
 4  if head[H] = NIL
 5     then return H
 6  prev-x ← NIL
 7  x ← head[H]
 8  next-x ← sibling[x]
 9  while next-x ≠ NIL
10        do if (degree[x] ≠ degree[next-x]) or
                    (sibling[next-x] ≠ NIL
                    and degree[sibling[next-x]] = degree[x])
11              then prev-x ← x                                  ▷ Cases 1 and 2
12                   x ← next-x                                  ▷ Cases 1 and 2
13              else if key[x] ≤ key[next-x]
14                      then sibling[x] ← sibling[next-x]        ▷ Case 3
15                           BINOMIAL-LINK(next-x, x)            ▷ Case 3
16                      else if prev-x = NIL                     ▷ Case 4
17                              then head[H] ← next-x            ▷ Case 4
18                              else sibling[prev-x] ← next-x    ▷ Case 4
19                           BINOMIAL-LINK(x, next-x)            ▷ Case 4
20                           x ← next-x                          ▷ Case 4
21           next-x ← sibling[x]
22  return H
```

באיור 20.5 מובאת דוגמה לפעולתה של BINOMIAL-HEAP-UNION שבה מתרחשים כל ארבעת המקרים המופיעים בפסאודו־קוד.

השגרה BINOMIAL-HEAP-UNION כוללת שני שלבים. השלב הראשון, המתבצע באמצעות קריאה ל־BINOMIAL-HEAP-MERGE, ממזג את רשימות השורשים של H_1 ו־H_2 לרשימה מקושרת אחת H הממוינת בסדר מונוטוני עולה של דרגות. אולם הרשימה הממוזגת עשויה להכיל שני שורשים (אך לא יותר) בעלי אותה דרגה, ולכן השלב השני מקשר שורשים בעלי אותה דרגה עד אשר נותר ברשימה לכל היותר שורש אחד מכל דרגה. מכיוון שהרשימה המקושרת H ממוינת על פי דרגות, ניתן לבצע במהירות את פעולות הקישור.

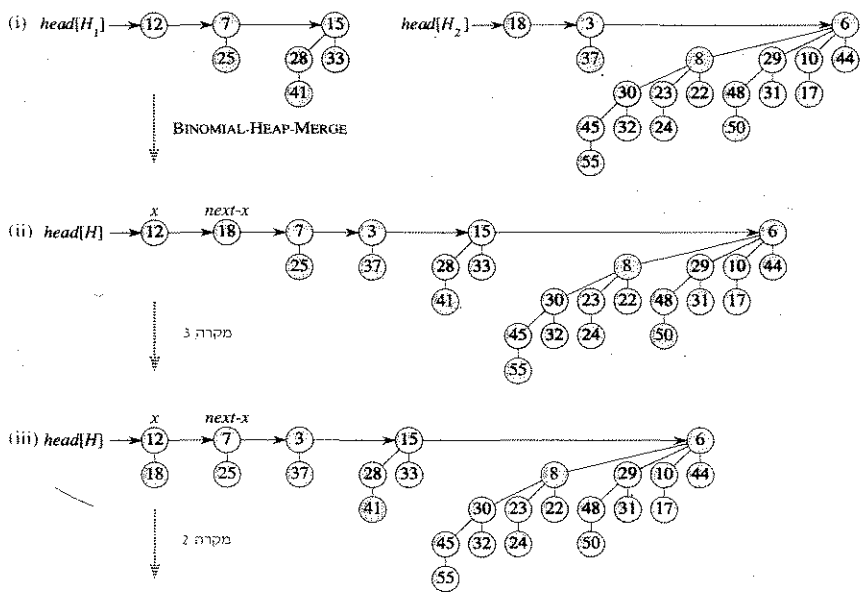
להלן תיאור מפורט של דרך פעולתה של השגרה: בתחילה, שורות 1-3 ממזגות את רשימות השורשים של הערימות הבינומיות H_1 ו־H_2 לרשימת שורשים יחידה H. רשימות השורשים של H_1 ו־H_2 ממוינות בסדר עולה ממש של דרגות, והשגרה BINOMIAL-HEAP-MERGE מחזירה רשימת שורשים H הממוינת בסדר מונוטוני עולה של דרגות. אם רשימות השורשים של H_1 ו־H_2 מכילות יחד m שורשים, BINOMIAL-HEAP-MERGE רצה בזמן $O(m)$ על־ידי כך שהיא חוזרת ובוחנת את השורשים שבראש שתי רשימות השורשים, מוסיפה את השורש בעל הדרגה הנמוכה יותר לרשימת הפלט, ומסירה אותו תוך כדי כך מרשימת הקלט שלו.

לאחר מכן, השגרה BINOMIAL-HEAP-UNION מאתחלת כמה מצביעים לרשימת השורשים של H. בראשית, השגרה פשוט חוזרת בשורות 4-5 אם שתי הערימות שהיא מאחדת ריקות. משורה 6 ואילך אנו יודעים אפוא ש־H מכילה לפחות שורש אחד. במהלך השגרה אנו מתחזקים שלושה מצביעים לרשימת השורשים:

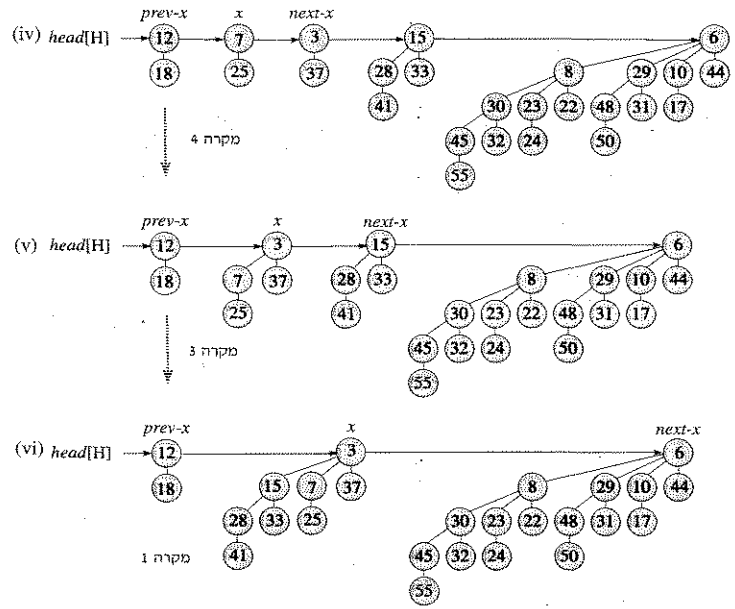
- x מצביע לשורש הנבחן עתה;
- $prev\text{-}x$ מצביע לשורש הקודם ל־x ברשימת השורשים: $sibling[prev\text{-}x] = x$;
- $next\text{-}x$ מצביע לשורש העוקב ל־x ברשימת השורשים: $sibling[x] = next\text{-}x$.

במצב ההתחלתי, רשימת השורשים של H מכילה לכל היותר שני שורשים בעלי דרגה נתונה. מאחר ש־H_1 ו־H_2 היו ערימות בינומיות, כל אחת מהן הכילה שורש אחד בלבד בעל דרגה נתונה. יתר על כן, BINOMIAL-HEAP-MERGE מבטיחה לנו שאם H מכילה שני שורשים בעלי אותה דרגה, הם סמוכים זה לזה ברשימת השורשים.

למעשה, במהלך ביצועה של BINOMIAL-HEAP-UNION ייתכן שברגע מסוים יופיעו ברשימת השורשים של H שלושה שורשים בעלי דרגה נתונה. מיד נראה כיצד יכול להיווצר מצב כזה. בכל איטרציה של לולאת ה־**while** שבשורות 9-21 אנו מחליטים אפוא אם לקשר את x ואת $next\text{-}x$, וזאת על סמך דרגתם ולעתים גם על סמך דרגתו של $sibling[next\text{-}x]$. אינווריאנטה של הלולאה היא, שבכל פעם שאנו מתחילים בביצוע גוף הלולאה, x וגם $next\text{-}x$ אינם NIL.



איור 20.5 פעולת השגרה BINOMIAL-HEAP-UNION. **(i)** הערימות הבינומיות H_1 ו־H_2. **(ii)** הערימה הבינומית H היא הפלט של BINOMIAL-HEAP-MERGE(H_1, H_2). במצב ההתחלתי, x הוא השורש הראשון ברשימת השורשים של H. מכיוון ש־$key[x] < key[next\text{-}x]$ וגם ל־$next\text{-}x$ דרגה 0 וכן x, המצב מתאים למקרה 3. **(iii)** לאחר ביצוע הקישור, x הוא הראשון מבין שלושה שורשים בעלי אותה דרגה, ולכן כאן מתאים מקרה 2. **(iv)** לאחר שכל המצביעים קודמו למיקום הבא ברשימת השורשים, המצב מתאים למקרה 4, שכן x הוא הראשון מבין שני שורשים בעלי אותה דרגה. **(v)** לאחר ביצוע הקישור, המצב מתאים שוב למקרה 3. **(vi)** לאחר קישור נוסף, המצב מתאים למקרה 1, שכן דרגתו של x היא 3 ודרגתו של $next\text{-}x$ היא 4. איטרציה זו של לולאת ה־**while** היא האחרונה, משום שלאחר שמקדמים את המצביעים למיקום הבא ברשימת השורשים, $next\text{-}x = \text{NIL}$.



איור 20.5 (המשך)

מקרה 1, המוצג באיור 20.6(i), מתרחש כאשר $degree[x] \neq degree[next\text{-}x]$, כלומר, כאשר x הוא השורש של עץ B_k ו־$next\text{-}x$ הוא השורש של עץ B_l, עבור $l > k$ כלשהם. שורות 11-12 מטפלות במקרה זה. אין צורך לקשר את x ואת $next\text{-}x$, כך שאנו פשוט מקדמים את המצביעים למיקום הבא ברשימה. עדכון $next\text{-}x$ כך שיצביע על הצומת העוקב לצומת x החדש מתבצע בשורה 21, המשותפת לכל המקרים.

מקרה 2, המוצג באיור 20.6(ii), מתרחש כאשר x הוא הראשון מבין שלושה שורשים בעלי אותה דרגה, כלומר, כאשר:

$$degree[x] = degree[next\text{-}x] = degree[sibling[next\text{-}x]]$$

אנו מטפלים במקרה זה כמו במקרה 1: אנו פשוט מקדמים את המצביעים למיקום הבא ברשימה. שורה 10 בודקת אם מתקיים אחד משני מקרים אלה ושורות 11-12 מטפלות בשניהם.

מקרים 3 ו־4 מתרחשים כאשר x הוא הראשון מבין שני שורשים בעלי אותה דרגה, כלומר, כאשר:

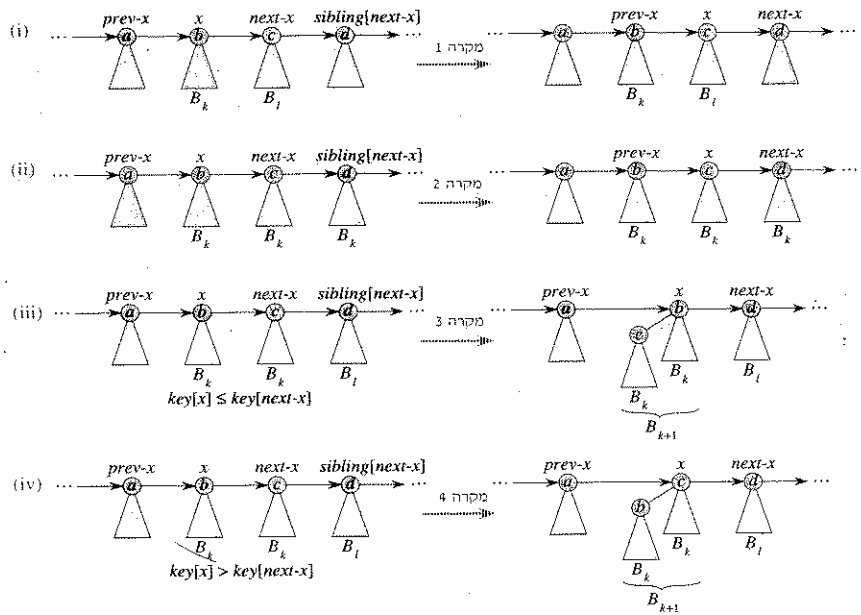
$$degree[x] = degree[next\text{-}x] \neq degree[sibling[next\text{-}x]]$$

מקרים אלה עשויים להתרחש באיטרציה הבאה בתור לאחר כל אחד מן המקרים, אולם אחד מהם מתרחש תמיד מיד לאחר מקרה 2. במקרים 3 ו־4 אנו מקשרים את x ואת $next\text{-}x$. ההבחנה בין המקרים תלויה בנוסדלם היחסי של המפתחות של x ושל $next\text{-}x$, דבר הקובע איזה צומת יהיה השורש לאחר הקישור של x ו־$next\text{-}x$.

במקרה 3, המוצג באיור 20.6(iii), $key[x] \leq key[next\text{-}x]$, ולכן $next\text{-}x$ מקושר ל־x. שורה 14 מסירה את $next\text{-}x$ מרשימת השורשים, ושורה 15 הופכת את $next\text{-}x$ לבן השמאלי ביותר של x.

במקרה 4, המוצג באיור 20.6(iv), $next\text{-}x$ הוא בעל המפתח הקטן יותר, ולכן x מקושר ל־$next\text{-}x$. שורות 16-18 מסירות את x מרשימת השורשים; כאן ייתכנו שני מקרים: האחד שבו x הוא השורש הראשון ברשימה (שורה 17) והאחר שבו x אינו הראשון (שורה 18). לאחר מכן, שורה 19 הופכת את x לבן השמאלי ביותר של $next\text{-}x$, ושורה 20 מעדכנת את x עבור האיטרציה הבאה.

בין אם התרחש מקרה 3 ובין אם התרחש מקרה 4, ההכנה לאיטרציה הבאה של לולאת ה־**while** זהה: זה עתה קישרנו שני עצי B_k ליצירת עץ B_{k+1}, שעליו x מצביע עתה. מספרם של עצי B_{k+1} ברשימת השורשים שהתקבלה כפלט מ־BINOMIAL-HEAP-MERGE היה אפס, אחד או שניים. לכן עתה x הוא עץ B_{k+1} היחיד ברשימת השורשים או שהוא הראשון מבין שניים או שלושה עצי B_{k+1}. אם x הוא היחיד, אזי באיטרציה הבאה ניכנס למקרה 4: $degree[x] \neq degree[next\text{-}x]$. אם x הוא הראשון מבין שניים, אזי באיטרציה הבאה ניכנס למקרה 3 או למקרה 4. כאשר x הוא הראשון מבין שלושה, זהו המצב שבו ניכנס באיטרציה הבאה למקרה 2.



איור 20.6 ארבעת המקרים המתרחשים ב־BINOMIAL-HEAP-UNION. התוויות a, b, c ו־d משמשות רק לזיהוי השורשים; הן אינן מציינות דרגות או מפתחות של שורשים אלה. בכל אחד מן המקרים, x הוא השורש של עץ B_k ו־$l > k$. **(i)** מקרה 1: $degree[x] \neq degree[next\text{-}x]$. המצביעים נעים למיקום הבא ברשימת השורשים. **(ii)** מקרה 2: $degree[x] = degree[next\text{-}x] = degree[sibling[next\text{-}x]]$. גם כאן המצביעים נעים למיקום הבא ברשימה, והאיטרציה הבאה מבצעת את מקרה 3 או את מקרה 4. **(iii)** מקרה 3: $degree[x] = degree[next\text{-}x] \neq degree[sibling[next\text{-}x]]$ וכן $key[x] \leq key[next\text{-}x]$. אנו מסירים את $next\text{-}x$ מרשימת השורשים ומקשרים אותו ל־x, ויוצרים בכך עץ B_{k+1}. **(iv)** מקרה 4: $degree[x] = degree[next\text{-}x] \neq degree[sibling[next\text{-}x]]$ וכן $key[x] \leq key[next\text{-}x]$. אנו מסירים את x מרשימת השורשים, מקשרים אותו ל־$next\text{-}x$, ויוצרים גם הפעם עץ B_{k+1}.

זמן הריצה של BINOMIAL-HEAP-UNION הוא $O(\lg n)$, כאשר n הוא המספר הכולל של צמתים בערימות הבינומיות H_1 ו־H_2. ניתן להיווכח בכך באופן הבא: נניח ש־H_1 מכילה n_1 צמתים ו־H_2 מכילה n_2 צמתים, כך ש־$n = n_1 + n_2$. אם כן, H_1 מכילה לכל היותר $\lfloor \lg n_1 \rfloor + 1$ שורשים ו־H_2 מכילה לכל היותר $\lfloor \lg n_2 \rfloor + 1$ שורשים, כך שמיד לאחר הקריאה ל־BINOMIAL-HEAP-MERGE, H מכילה לכל היותר $\lfloor \lg n_1 \rfloor + \lfloor \lg n_2 \rfloor + 2 \le 2\lfloor \lg n \rfloor + 2 = O(\lg n)$ שורשים. זמן הריצה של BINOMIAL-HEAP-MERGE הוא אפוא $O(\lg n)$. כל איטרציה של לולאת ה־**while** רצה בזמן $O(1)$, ומתבצעות לכל היותר $\lfloor \lg n_1 \rfloor + \lfloor \lg n_2 \rfloor + 2$ איטרציות, שכן כל איטרציה, או שהיא מקדמת את המצביעים למיקום הבא ברשימת השורשים של H, או שהיא מסירה שורש מרשימת השורשים. זמן הריצה הכולל הוא אפוא $O(\lg n)$.

הכנסת צומת

השגרה שלהלן מכניסה צומת x לערימה בינומית H. ההנחה היא, כמובן, שהצומת x כבר הוקצה ובשדה $key[x]$ כבר הוצב ערך.

```
BINOMIAL-HEAP-INSERT(H, x)
1  H' ← MAKE-BINOMIAL-HEAP( )
2  p[x] ← NIL
3  child[x] ← NIL
4  sibling[x] ← NIL
5  degree[x] ← 0
6  head[H'] ← x
7  H ← BINOMIAL-HEAP-UNION(H, H')
```

השגרה פשוט יוצרת ערימה בינומית H' בעלת צומת אחד בזמן $O(1)$, ומאחדת אותה עם הערימה הבינומית ה־n־צומתית H בזמן $O(\lg n)$. הקריאה לשגרה BINOMIAL-HEAP-UNION מטפלת בשחרור הערימה הבינומית הזמנית H'. (מימוש ישיר שאינו קורא לשגרה BINOMIAL-HEAP-UNION מובא בתרגיל 20.2-8.)

שליפת הצומת בעל המפתח המינימלי

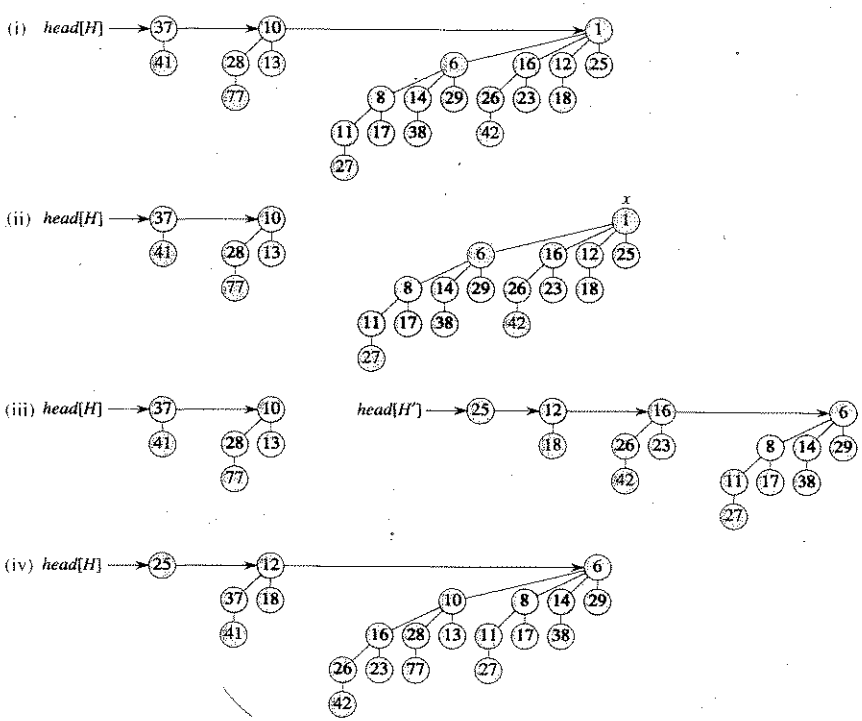
השגרה שלהלן שולפת את הצומת בעל המפתח המינימלי מערימה בינומית H ומחזירה מצביע לצומת שנשלף.

```
BINOMIAL-HEAP-EXTRACT-MIN(H)
1  find the root x with the minimum key in the root list of H,
       and remove x from the root list of H
2  H' ← MAKE-BINOMIAL-HEAP( )
3  reverse the order of the linked list of x's children,
       and set head[H'] to point to the head of the resulting list
4  H ← BINOMIAL-HEAP-UNION(H, H')
5  return x
```

שגרה זו פועלת כמתואר באיור 20.7. ערימת הקלט הבינומית H מוצגת באיור 20.7(i). איור 20.7(ii) מתאר את המצב לאחר ביצוע שורה 1 של השגרה: השורש x בעל המפתח המינימלי הוסר מרשימת השורשים של H. אם x הוא השורש של עץ B_k, אזי על פי תכונה 4 בלמה 20.1, בניו של x, משמאל לימין, הם השורשים של עצים $B_0, B_1, \ldots, B_{k-1}$. איור 20.7(iii) מראה

שעל־ידי היפוך רשימת הבנים של x בשורה 3, אנו מקבלים ערימה בינומית H'. המכילה את כל הצמתים בעץ של x, פרט ל־x עצמו. מכיוון שהעץ של x מוסר מ־H בשורה 1, הערימה הבינומית המתקבלת מאיחוד H ו־H' בשורה 4, והמתוארת באיור 20.7(iv), מכילה את כל הצמתים שהיו בערימה H המקורית, פרט ל־x. לבסוף, שורה 5 מחזירה את x.

מאחר שכל אחת מן השורות 1-4 מתבצעת בזמן $O(\lg n)$ אם H מכילה n צמתים, השגרה BINOMIAL-HEAP-EXTRACT-MIN רצה בזמן $O(\lg n)$.



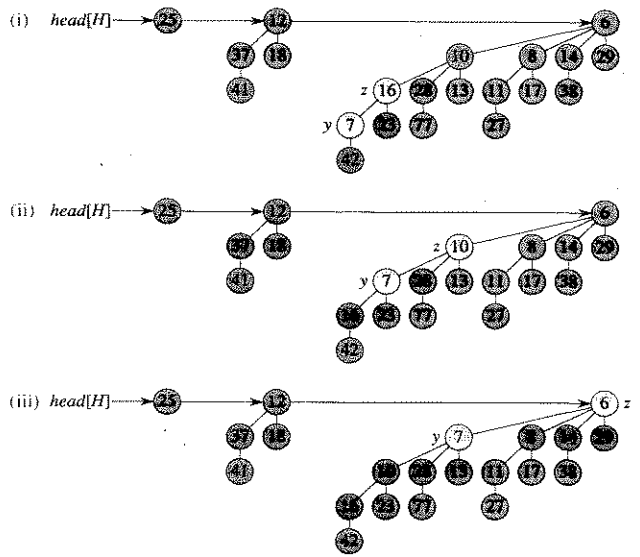
איור 20.7 פעולתה של BINOMIAL-HEAP-EXTRACT-MIN. **(i)** ערימה בינומית H. **(ii)** השורש x בעל המפתח המינימלי מוסר מרשימת השורשים של H. **(iii)** היפוך הרשימה המקושרת של בניו של x נותן ערימה בינומית נוספת H'. **(iv)** תוצאת האיחוד של H ו־H'.

הקטנת ערכו של מפתח

השגרה שלהלן מקטינה את ערכו של מפתח של צומת x בערימה בינומית H לערך חדש k. השגרה מודיעה על שגיאה אם k גדול מהמפתח הנוכחי של x.

BINOMIAL-HEAP-DECREASE-KEY(H, x, k)

```
1  if k > key[x]
2     then error "new key is greater than current key"
3  key[x] ← k
4  y ← x
5  z ← p[y]
6  while z ≠ NIL and key[y] < key[z]
7        do exchange key[y] ↔ key[z]
8           ▷ If y and z have satellite fields, exchange them, too.
9           y ← z
10          z ← p[y]
```



איור 20.8 פעולתה של BINOMIAL-HEAP-DECREASE-KEY. **(i)** המצב ממש לפני ביצוע שורה 5 באיטרציה הראשונה של לולאת ה־**while**. ערך המפתח של הצומת y הוקטן ל־7, והוא קטן מערכו של המפתח של z, אביו של y. **(ii)** מפתחותיהם של שני הצמתים מוחלפים זה בזה, ובאיור מתואר המצב ממש לפני ביצוע שורה 5 באיטרציה השנייה. המצביעים y ו־z התקדמו רמה אחת במעלה העץ, אולם סדר הערימה עדיין מופר. **(iii)** לאחר החלפה נוספת וקידום המצביעים y ו־z רמה אחת נוספת במעלה העץ, אנו מגלים סוף סוף שסדר הערימה מתקיים, ולכן לולאת ה־**while** עוצרת.

כפי שמראה איור 20.8, שגרה זו מקטינה ערך של מפתח בערימה בינומית באותו אופן שבו נעשה הדבר בערימה בינרית: המפתח החדש "מבעבע" במעלה הערימה. לאחר שהשגרה מוודאת כי המפתח החדש אכן אינו גדול מהמפתח הנוכחי, ולאחר שהיא מציבה את הערך החדש בשדה המפתח של x, השגרה מתקדמת במעלה העץ, כאשר y מצביע בתחילה על הצומת x. בכל איטרציה של לולאת ה־**while** שבשורות 6-10 משווים את $key[y]$ למפתח של z, אביו של y. אם y הוא השורש או $key[y] \geq key[z]$, העץ הבינומי מקיים עתה את סדר הערימה. אחרת,

הצומת y מפר את סדר הערימה, ולכן המפתח שלו והמפתח של אביו z מוחלפים זה בזה, יחד עם הנתונים הנלווים שלהם. לאחר מכן השגרה מציבה ב־y את z, ובכך היא מתקדמת רמה אחת במעלה העץ, וממשיכה באיטרציה הבאה.

השגרה BINOMIAL-HEAP-DECREASE-KEY רצה בזמן $O(\lg n)$. מתכונה 2 בלמה 20.1 נובע שהעומק המקסימלי של x הוא $\lfloor \lg n \rfloor$, כך שלולאת ה־**while** בשורות 6-10 מתבצעת לכל היותר $\lfloor \lg n \rfloor$ פעמים.

מחיקת מפתח

קל למחוק מפתח ונתונים נלווים של צומת x מערימה בינומית H בזמן $O(\lg n)$. המימוש שלהלן מניח שבערימה הבינומית הנוכחית לא קיים צומת בעל מפתח $-\infty$.

BINOMIAL-HEAP-DELETE(H, x)

```
1  BINOMIAL-HEAP-DECREASE-KEY(H, x, -∞)
2  BINOMIAL-HEAP-EXTRACT-MIN(H)
```

השגרה BINOMIAL-HEAP-DELETE הופכת את הצומת x לצומת בעל מפתח מינימלי יחיד בערימה הבינומית כולה על ידי כך שהיא נותנת למפתח שלו את הערך $-\infty$. (תרגיל 20.2-6 מטפל במקרה שבו $-\infty$ אינו יכול להופיע כמפתח, אפילו לא באופן זמני.) לאחר מכן היא גורמת לכך שמפתח זה והנתונים הנלווים שלו יבעבעו במעלה העץ עד לשורש, באמצעות קריאה ל־BINOMIAL-HEAP-DECREASE-KEY. הצעד הבא הוא הוצאת שורש זה מ־H באמצעות קריאה ל־BINOMIAL-HEAP-EXTRACT-MIN.

השגרה BINOMIAL-HEAP-DELETE רצה בזמן $O(\lg n)$.

תרגילים

20.2-1

הבא דוגמה לשתי ערימות בינריות שכל אחת מהן בת n איברים, כך שהפעלת BUILD-HEAP על שרשור שני המערכים שלהן מתבצעת בזמן $\Theta(n)$.

20.2-2

כתוב פסאודו־קוד עבור BINOMIAL-HEAP-MERGE.

20.2-3

הצג את הערימה הבינומית המתקבלת מהכנסת צומת בעל מפתח 24 לערימה הבינומית שבאיור 20.7(iv).

20.2-4

הצג את הערימה הבינומית המתקבלת ממחיקת הצומת בעל המפתח 28 מן הערימה הבינומית שבאיור 20.8(iii).

20.2-5

הסבר מדוע השגרה BINOMIAL-HEAP-MINIMUM עלולה שלא לפעול כהלכה כאשר המפתחות יכולים להיות בעלי הערך $-\infty$. כתוב מחדש את הפסאודו־קוד כך שיפעל היטב גם במקרים אלה.

20.2-6

נניח שאין דרך לייצג את המפתח $-\infty$. כתוב מחדש את השגרה BINOMIAL-HEAP-DELETE כך שתפעל כהלכה במצב זה. השגרה עדיין צריכה לרוץ בזמן $O(\lg n)$.

20.2-7

דון בקשר שבין הכנסת איבר לערימה בינומית לבין הוספת 1 למספר בינרי ובקשר שבין איחוד שתי ערימות בינומיות לבין חיבור שני מספרים בינריים.

20.2-8

לאור תרגיל 20.2-7, כתוב מחדש את BINOMIAL-HEAP-INSERT כך שהשגרה תכניס צומת ישירות לערימה בינומית, בלי לקרוא ל־BINOMIAL-HEAP-UNION.

20.2-9

הראה שאם שומרים רשימות שורשים כך שהן ממוינות בסדר יורד ממש (במקום בסדר עולה ממש) של דרגות, ניתן לממש כל אחת מן הפעולות על ערימות בינומיות בלי לשנות את זמן הריצה האסימפטוטי שלה.

20.2-10

מצא קלטים הגורמים לכך שהשגרות BINOMIAL-HEAP-EXTRACT-MIN, BINOMIAL-HEAP-DECREASE-KEY ו־BINOMIAL-HEAP-DELETE ירוצו בזמן $\Omega(\lg n)$. הסבר מדוע זמני הריצה במקרה הגרוע של BINOMIAL-HEAP-INSERT, BINOMIAL-HEAP-MINIMUM ו־BINOMIAL-HEAP-UNION הם $\tilde{\Omega}(\lg n)$ אך לא $\Omega(\lg n)$. (ראה בעיה 2-5.)

בעיות

20-1 ערימות 2-3-4

בפרק 19 הוצגו עצי 2-3-4, אשר בהם לכל צומת פנימי (פרט אולי לשורש) יש שניים, שלושה או ארבעה בנים, וכל העלים הם בעלי אותו עומק. בבעיה זו נממש **ערימות 2-3-4** (2-3-4 heaps) התומכות בפעולות על ערימות מתמזגות.

ערימות 2-3-4 שונות מעצי 2-3-4 בדברים הבאים: בערימות 2-3-4, מפתחות מאוחסנים אך ורק בעלים, ובכל עלה x מאוחסן מפתח אחד בדיוק בשדה $key[x]$. המפתחות בעלים אינם סדורים בסדר מסוים; כלומר, המפתחות, כשקוראים אותם משמאל לימין, עשויים להופיע בכל סדר שהוא. כל צומת פנימי x מכיל שדה $small[x]$ אשר ערכו שווה למפתח הקטן ביותר מבין המפתחות המאוחסנים בעלים של תת-עץ המושרש ב-x. השורש r מכיל שדה $height[r]$ שערכו הוא גובה העץ. לבסוף, ערימות 2-3-4 מיועדות להישמר בזיכרון הראשי, כך שאין צורך בפעולות קריאה או כתיבה על דיסק.

ממש את הפעולות שלהלן על ערימות 2-3-4. כל אחת מהפעולות שבסעיפים (א)-(ה) צריכה לרוץ בזמן $O(\lg n)$ על ערימת 2-3-4 בת n איברים. הפעולה UNION שבסעיף (ו) צריכה לרוץ בזמן $O(\lg n)$, כאשר n הוא מספר האיברים בשתי ערימות הקלט.

א. MINIMUM — מחזירה מצביע לעלה בעל המפתח הקטן ביותר.

ב. DECREASE-KEY — מקטינה את ערך המפתח של עלה נתון x לערך נתון $k \le key[x]$.

ג. INSERT — מכניסה עלה x בעל מפתח k.

ד. DELETE — מוחקת עלה נתון x.

ה. EXTRACT-MIN — שולפת מן הערימה את העלה בעל המפתח הקטן ביותר.

ו. UNION — מאחדת שתי ערימות 2-3-4, מחזירה ערימת 2-3-4 יחידה והורסת את ערימות הקלט.

20-2 אלגוריתם למציאת עץ פורש מינימלי בעזרת ערימות מתמזגות

בפרק 24 נציג שני אלגוריתמים לפתרון הבעיה של מציאת עץ פורש מינימלי עבור גרף בלתי מכוון. כאן נראה כיצד ניתן להשתמש בערימות מתמזגות לתכנון אלגוריתם שונה למציאת עץ פורש מינימלי.

נתון לנו גרף קשיר בלתי-מכוון $G = (V, E)$ עם פונקציית משקל $w : E \to \mathbf{R}$. $w(u, v)$ הוא משקל הקשת (u, v). ברצוננו למצוא עץ פורש מינימלי עבור G, דהיינו, תת-קבוצה ללא מעגלים $T \subseteq E$ הקושרת את כל הקדקודים ב-V ומשקלה הכולל:

$$w(T) = \sum_{(u,v)\in T} w(u, v)$$

הוא מינימלי.

הפסאודו-קוד שלהלן, שאת נכונותו ניתן להוכיח באמצעות שיטות המתוארות בסעיף 24.1, בונה עץ פורש מינימלי T. הוא מתחזק חלוקה $\{V_i\}$ של קדקודי V, ובצמוד לכל קבוצה V_i קבוצה E_i של קשתות הקשורות לקדקודים השייכים ל-V_i.

$E_i \subseteq \{(u, v) : v \in V_i$ או $u \in V_i\}$

```
MST-MERGEABLE-HEAP(G)
1   T ← φ
2   for each vertex v_i ∈ V[G]
3       do V_i ← {v_i}
4          E_i ← {(v_i, v) ∈ E[G]}
5   while there is more than one set V_i
6       do choose any set V_i
7          extract the minimum-weight edge (u, v) from E_i
8          assume without loss of generality that u ∈ V_i and v ∈ V_j
9          if i ≠ j
10            then T ← T ∪ {(u, v)}
11                 V_i ← V_i ∪ V_j, destroying V_j
12                 E_i ← E_i ∪ E_j
```

תאר כיצד ניתן לממש אלגוריתם זה תוך שימוש בפעולות על ערימות מתמזגות הנתונות באיור 20.1. מצא את זמן הריצה של המימוש שלך, בהנחה שהערימות המתמזגות ממומשות באמצעות ערימות בינומיות.

הערות ביבליוגרפיות לפרק 20

ערימות בינומיות הוצגו ב־1978 על־ידי Vuillemin [196]. Brown [36, 37] חקר בפירוט את תכונותיהן.

21 ערימות פיבונצ'י

בפרק 20 ראינו כיצד ערימות בינומיות תומכות בזמן $O(\lg n)$ במקרה הגרוע בפעולות INSERT, EXTRACT-MIN, MINIMUM, ו־UNION על ערימות מתמזגות, וכן בפעולות DECREASE-KEY ו־DELETE. בפרק זה נבחן ערימות פיבונצ'י, התומכות באותן פעולות אולם יש להן יתרון בכך שפעולות אשר אינן כרוכות במחיקת איבר רצות בזמן לשיעורין של $O(1)$.

מנקודת מבט תיאורטית, ערימות פיבונצ'י רצויות במיוחד כאשר מספר הפעולות EXTRACT-MIN ו־DELETE קטן יחסית למספר הפעולות האחרות המבוצעות. מצב כזה קיים ביישומים רבים. לדוגמה, אלגוריתמים מסוימים לפתרון בעיות גרפים עשויים לקרוא ל־DECREASE-KEY פעם אחת עבור כל קשת. עבור גרפים צפופים המכילים קשתות רבות, הזמן לשיעורין $O(1)$ של כל קריאה ל־DECREASE-KEY מצטבר לכדי שיפור ניכר בהשוואה לזמן $\Theta(\lg n)$ במקרה הגרוע עבור ערימות בינריות או בינומיות. ערימות פיבונצ'י מהוות מרכיב חיוני באלגוריתמים המהירים ביותר אסימפטוטית המוכרים לנו כיום לפתרון בעיות כגון בניית עצים פורשים מינימליים (פרק 24) ומציאת מסלולים קצרים ביותר ממקור יחיד (פרק 25).

אולם מנקודת מבט מעשית, הגורמים הקבועים ומורכבות התכנות של ערימות פיבונצ'י הופכים אותן לרצויות פחות מאשר ערימות בינריות (או k־אריות) רגילות עבור מרבית היישומים. העניין שיש לנו בערימות פיבונצ'י הוא אפוא תיאורטי בעיקרו. אם היו מצליחים לפתח מבנה נתונים פשוט יותר בעל אותם חסמי זמן לשיעורין כמו ערימות פיבונצ'י היתה בכך גם תועלת מעשית רבה.

כמו ערימה בינומית, ערימת פיבונצ'י היא אוסף של עצים. למעשה, ערימות פיבונצ'י מבוססות באופן רופף על ערימות בינומיות. אם לא מתבצעת אף פעם קריאה ל־DECREASE-KEY או ל־DELETE על ערימת פיבונצ'י, כל עץ בערימה הוא כמו עץ בינומי. אבל ערימות פיבונצ'י נבדלות מערימות בינומיות בכך שהמבנה שלהן נוקשה פחות, וכך הן מאפשרות חסמי זמן אסימפטוטיים משופרים. את עבודת תחזוקת המבנה ניתן לדחות עד למועד שבו נוח לבצעה.

כמו הטבלאות הדינמיות מסעיף 18.4, ערימות פיבונצ'י הן דוגמה טובה למבנה נתונים שתוכנן מתוך מחשבה שיעילותו תיקבע על־ידי ניתוח לשיעורין. ההבנה האינטואיטיבית והניתוח של הפעולות על ערימות פיבונצ'י שיוצגו בהמשך הפרק מסתמכים במידה רבה על שיטת הפוטנציאל שהוצגה בסעיף 18.3.

הסעיף הפותח פרק זה מניח שהקורא כבר מכיר את פרק 20 העוסק בערימות בינומיות. בפרק ההוא מופיעים אפיוני הפעולות, וכן מופיעה הטבלה שבאיור 20.1 המסכמת את חסמי הזמן עבור פעולות על ערימות בינריות, ערימות בינומיות וערימות פיבונצ'י. בהצגת ערימות פיבונצ'י אנו מסתמכים על המבנה של ערימות בינומיות. כמו כן נוכל לראות שחלק מן הפעולות המבוצעות על ערימות פיבונצ'י דומות לאלה המבוצעות על ערימות בינומיות.

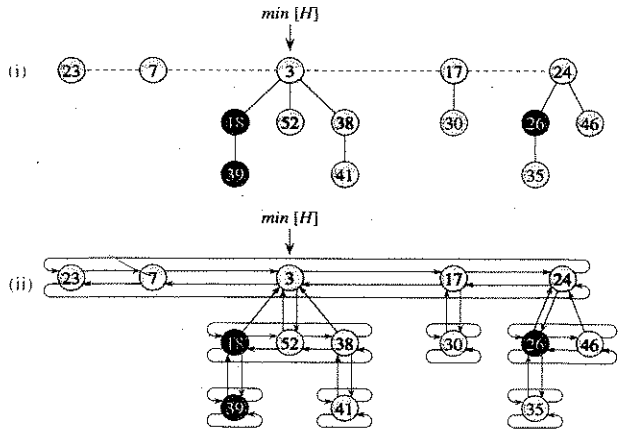
בדומה לערימות בינומיות, ערימות פיבונצ'י לא תוכננו לתמוך ביעילות בפעולה SEARCH. משום כך, פעולות המתייחסות לצומת נתון דורשות מצביע לצומת זה כחלק מן הקלט שלהן.

בסעיף 21.1 נגדיר ערימות פיבונצ'י, נדון בייצוג שלהן, ונציג את פונקציית הפוטנציאל המשמשת בניתוח לשיעורין שלהן. בסעיף 21.2 נראה כיצד ניתן לממש את הפעולות על ערימות מתמזגות כך שיושגו חסמי הזמן לשיעורין המובאים באיור 20.1. את שתי הפעולות הנותרות, DECREASE-KEY ו־DELETE, נציג בסעיף 21.3. לבסוף, בסעיף 21.4, נשלים חלק מרכזי של הניתוח.

21.1 המבנה של ערימות פיבונצ'י

כמו ערימה בינומית, **ערימת פיבונצ'י** (Fibonacci heap) היא אוסף של עצים אשר מקיימים את סדר הערימה. אולם העצים בערימת פיבונצ'י אינם חייבים להיות עצים בינומיים. באיור 21.1(i) מוצגת דוגמה של ערימת פיבונצ'י.

בניגוד לעצים המרכיבים ערימות בינומיות, שהם סדורים, העצים המרכיבים ערימות פיבונצ'י הם עצים מושרשים אך לא סדורים. כפי שמראה איור 21.1(ii), כל צומת x מכיל מצביע $p[x]$ לאביו ומצביע $child[x]$ לבן כלשהו מבניו. בניו של x מקושרים יחדיו ברשימה דו־מקושרת מעגלית שאותה אנו מכנים **רשימת הבנים** (child list) של x. כל בן y ברשימת הבנים מכיל מצביעים $left[y]$ ו־$right[y]$ לאח השמאלי ולאח הימני שלו, בהתאמה. אם צומת y הוא בן יחיד, אזי $left[y] = right[y] = y$. הסדר שבו מופיעים הצמתים ברשימת הבנים הוא שרירותי.



איור 21.1 **(i)** ערימת פיבונצ'י המורכבת מחמישה עצים המקיימים את סדר הערימה ומכילה 14 צמתים. הקו המקווקו מציין את רשימת השורשים. צומת המינימום של הערימה הוא הצומת המכיל את המפתח 3. שלושת הצמתים המסומנים מושחרים. הפוטנציאל של ערימת פיבונצ'י מסוימת זו הוא $11 = 3 + 2 \cdot 3 + 5$. **(ii)** ייצוג מלא יותר הכולל את המצביעים p (חיצים למעלה), המצביעים $child$ (חיצים למטה) והמצביעים $right$ ו־$left$ (חיצים ימינה ושמאלה, בהתאמה). פרטים אלה מושמטים משאר האיורים בפרק זה, שכן את כל המידע המופיע כאן ניתן להסיק מהמתואר בחלק (i) של האיור.

לרשימות דו־מקושרות מעגליות (ראה סעיף 11.2) יש שני יתרונות שאפשר לנצלם בערימות פיבונצ'י. ראשית, ניתן להסיר צומת מרשימה דו־מקושרת מעגלית בזמן $O(1)$. שנית, בהינתן שתי

דרגה מקסימלית

הוכיחו שבכל עץ בינומי H' יתכנו בדיוק x צמתים שדרגתם היא x עבור כל x שבין 0 ל-$\lg n$.

הכנסת צמתים

הוכיחו שבמקרה הגרוע ביותר, הכנסת צומת לערימת פיבונאצ'י H דורשת $\Omega(\lg n)$ פעולות. הניחו שבתחילה H ריקה, והשתמשו במספר הפעולות שנדרשות כדי לבצע n הכנסות עוקבות.

מיזוג ערימות פיבונאצ'י חדשה

נתאר כעת כיצד למזג שתי ערימות פיבונאצ'י, תוך עדכון מספר העצים $m(H)$ ומספר הצמתים הסמוכים ב-H: $t(H)$. הפונקציה הפוטנציאלית שנבחר תבטיח שמיזוג שתי ערימות יידרוש זמן מופחת בקירוב.

21.1 פונקציית פוטנציאל

כאמור, נשתמש בשיטת הפוטנציאל כדי לנתח את ביצועי הפעולות על ערימת פיבונאצ'י. בהינתן ערימת פיבונאצ'י H, נסמן ב-$t(H)$ את מספר העצים ברשימת השורשים של H וב-$m(H)$ את מספר הצמתים הסמוכים ב-H. הפוטנציאל של ערימת פיבונאצ'י H יוגדר אז על-ידי:

$$\Phi(H) = t(H) + 2m(H) \tag{21.1}$$

לדוגמה, הפוטנציאל של הערימת בתרגיל 21.1 הוא $11 = 2 \cdot 3 + 5$. הפוטנציאל של קבוצת ערימות פיבונאצ'י הוא סכום הפוטנציאלים של הערימות המרכיבות את הקבוצה. אנו נניח שיחידת הפוטנציאל שבה נמדוד את הפוטנציאל של ערימה כלשהי מתאימה ליחידת הזמן שבה נמדוד את זמן הריצה של הפעולות.

אנו מניחים שיישום של ערימת פיבונאצ'י מתחיל ללא ערימות כלל. לפיכך, הפוטנציאל ההתחלתי הוא 0, ולפי משוואה (21.1), הפוטנציאל הוא אי-שלילי בכל זמן מאוחר מכך. לפי משוואה (18.2), חסם עליון על העלות המשוערך הכוללת הוא אפוא חסם עליון על העלות הכוללת בפועל של כל סדרת הפעולות.

רשימת כאלה, ניתן לשייך אותן (או "לאחות" אותן) לרשימה דו-מקושרת מעגלית אחת בזמן $O(1)$ בזמן האיחוד. הפעולות על ערימת פיבונאצ'י נתוארו לפעולות אלה בצורה לא פורמלית ונותיר לקורא להשלים את פרטי מימושן.

כל צומת בערימת פיבונאצ'י מכיל שדות נוספים שעניינם בהם שימוש. נסמן את מספר הבנים ברשימת הבנים של צומת x באמצעות $x.degree$. השדה הבוליאני $x.mark$ מסמן אם צומת x איבד בן מאז הפעם האחרונה ש-x הפך לבנה של צומת אחר. אנו לא נעניין בצמתים שהם שורשים עד שנדון בפעולת המחיקה בסעיף 21.3. לצמתים שהם שורשים ולצמתים שאינם שורשים יש אותה משמעות בכל פעם שאנו מדברים על צמתים.

נניח שנתון ידית לכל צומת אחד. אין $min[H] = \text{NIL}$ פירושו שהערימה H ריקה, ואילו $min[H]$ הוא מצביע לצומת המינימום (minimum node) של הערימה. אם ערימת פיבונאצ'י H היא ריקה, אין צומת זו נקרא **צומת המינימום** של הערימה פיבונאצ'י H. הצומת $min[H]$ לשרש של רשימת השורשים. אם ערימת פיבונאצ'י H מכילה n צמתים שהם שורשים, אז $min[H]$ הוא השורש של הצומת שנמצאת במרחק $n-1$ צעדים מ-$min[H]$ לכיוון $right$ ברשימת השורשים. סדר הצמתים ברשימת השורשים שלהם לרשימה דו-מקושרת מעגלית הנקראת **רשימת השורשים** (root list) של הערימה. המצביע $left$ משמש בשיטת הפוטנציאל, אך לא נעקוב אחר כיוון המצביעים $left$ ו-$right$ בהקשר לשיטות אלה.

פונקציית פוטנציאל

נתחיל בכך שנגדיר את הפונקציה הפוטנציאלית שבה נשתמש לניתוח הפעולות על ערימות פיבונאצ'י. נתייחס לערימת פיבונאצ'י H שמספר הצמתים הסמוכים ב-H הוא n.

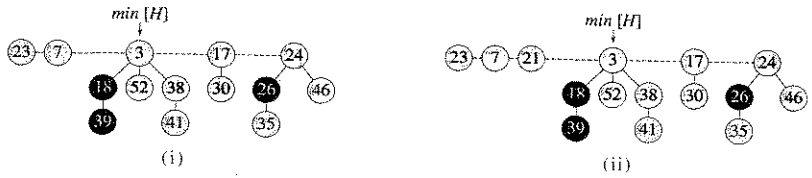
ב-$\Omega(\lg n)$ פעולות "עלה" הצומת x לראש רשימת השורשים, ולכן לאחר מכן ניתן יהיה לבצע את פעולת EXTRACT-MIN בזמן $O(\lg n)$. אנו נראה שכאשר יש לנו רק פעולות MAKE-FIB-HEAP, INSERT, MINIMUM, EXTRACT-MIN ו-UNION, אין צמתים שאינם שורשים, ולכן הערימה היא פשוט אוסף של העצים שנוצרו על-ידי הפעולות הקודמות. אם יש לנו גם פעולות DECREASE-KEY ו-DELETE, אז מספר העצים יכול לגדול הרבה יותר מ-$\lg n$. בסעיף 21.3 נראה שכאשר יש לנו רק פעולות אלה, $D(n) = O(\lg n)$.

21.2 פעולות על ערימות מתמזגות

בסעיף זה נתאר וננתח את הפעולות על ערימות מתמזגות שתומכות גם בפעולה DECREASE-KEY וגם בפעולה DELETE. נתאר גם כיצד שתי ערימות כאלה ניתנות למיזוג. הפעולות שנתאר מעלות את הסדר של $D(n)$ ל-$\Theta(\lg n)$.

```
FIB-HEAP-INSERT(H, x)
1  degree[x] ← 0
2  p[x] ← NIL
3  child[x] ← NIL
4  left[x] ← x
5  right[x] ← x
6  mark[x] ← FALSE
7  concatenate the root list containing x with root list H
8  if min[H] = NIL or key[x] < key[min[H]]
9       then min[H] ← x
10 n[H] ← n[H] + 1
```

שורות 1-6 מאתחלות את השדות המבניים של הצומת x, ובכך עושות אותו לרשימה דו־מקושרת מעגלית בפני עצמו. שורה 7 מוסיפה את x לרשימת השורשים של H, בזמן בפועל של $O(1)$. באופן זה הצומת x הופך לעץ חד־צומתי המקיים את סדר הערימה, ולכן לעץ בינומי בלתי־סדור בערימת פיבונצ'י זו. ל־x אין בנים והוא אינו מסומן. לאחר מכן, שורות 8-9 מעדכנות את המצביע לצומת המינימום של ערימת פיבונצ'י H אם יש צורך בכך. לבסוף, שורה 10 מוסיפה 1 ל־$n[H]$, שכן לערימה נוסף צומת חדש. באיור 21.2 מתוארת הכנסת צומת בעל מפתח 21 לערימת פיבונצ'י מאיור 21.1.



איור 21.2 הכנסת צומת לערימת פיבונצ'י. **(i)** ערימת פיבונצ'י H. **(ii)** הערימה H לאחר הכנסת צומת בעל מפתח 21. הצומת הופך לעץ בפני עצמו המקיים את סדר הערימה, ואז מוסיפים אותו לרשימת השורשים כאח שמאלי של השורש.

בניגוד לשגרה BINOMIAL-HEAP-INSERT, השגרה FIB-HEAP-INSERT אינה עושה שום ניסיון למזג את העצים בערימת פיבונצ'י. אם מתבצעות k פעולות FIB-HEAP-INSERT עוקבות, אזי לרשימת השורשים נוספים k עצים חד־צומתיים.

כדי לקבוע את העלות לשיעורין של FIB-HEAP-INSERT, נסמן ב־H את ערימת פיבונצ'י הנתונה כקלט וב־H' את הערימה המתקבלת לאחר ההכנסה. אזי, $t(H') = t(H) + 1$ ו־$m(H') = m(H)$, ותוספת הפוטנציאל היא:

$$((t(H) + 1) + 2m(H)) - (t(H) + 2m(H)) = 1$$

מאחר שהעלות בפועל היא $O(1)$, העלות לשיעורין היא $O(1) + 1 = O(1)$.

מציאת צומת המינימום

צומת המינימום של ערימת פיבונצ'י H נתון על־ידי המצביע $min[H]$, כך שהזמן בפועל למציאת צומת המינימום הוא $O(1)$. מכיוון שהפוטנציאל של H אינו משתנה, העלות לשיעורין של פעולה זו שווה לעלותה בפועל, שהיא $O(1)$.

איחוד שתי ערימות פיבונצ'י

השגרה הבאה מאחדת שתי ערימות פיבונצ'י H_1 ו־H_2, והורסת תוך כדי כך את H_1 ואת H_2.

```
FIB-HEAP-UNION(H1, H2)
1  H ← MAKE-FIB-HEAP()
2  min[H] ← min[H1]
3  concatenate the root list of H2 with the root list of H
4  if (min[H1] = NIL) or (min[H2] ≠ NIL and min[H2] < min[H1])
5       then min[H] ← min[H2]
6  n[H] ← n[H1] + n[H2]
7  free the objects H1 and H2
8  return H
```

שורות 1-3 משרשרות את רשימות השורשים של H_1 ושל H_2 לרשימת שורשים חדשה H. שורות 2, 4 ו־5 מציבות ב־$min[H]$ את צומת המינימום של H, ושורה 6 מציבה ב־$n[H]$ את מספר הצמתים הכולל. שורה 7 משחררת את העצמים H_1 ו־H_2, ושורה 8 מחזירה את ערימת פיבונצ'י H המתקבלת מן האיחוד. כמו FIB-HEAP-INSERT, גם שגרה זו אינה מבצעת מיזוג של עצים.

השינוי בפוטנציאל הוא:

$$\begin{aligned}
&\Phi(H) - (\Phi(H_1) + \Phi(H_2)) \\
&= (t(H) + 2m(H)) - ((t(H_1) + 2m(H_1)) + (t(H_2) + 2m(H_2))) \\
&= 0
\end{aligned}$$

שכן $t(H) = t(H_1) + t(H_2)$ ו־$m(H) = m(H_1) + m(H_2)$. העלות לשיעורין של FIB-HEAP-UNION שווה אפוא לעלותה בפועל, שהיא $O(1)$.

שליפת צומת המינימום

תהליך שליפתו של צומת המינימום הוא המסובך ביותר מבין הפעולות המוצגות בסעיף זה. כאן גם מתבצעת לבסוף העבודה הדחויה של מיזוג עצים ברשימת השורשים. הפסאודו־קוד שלהלן שולף את צומת המינימום. הקוד מניח מטעמי נוחות שכאשר צומת נשלף מרשימה מקושרת, המצביעים שנשארים ברשימה מעודכנים, אולם המצביעים בצומת שנשלף נותרים ללא שינוי. הקוד גם משתמש בשגרת העזר CONSOLIDATE שתוצג בהמשך.

```
FIB-HEAP-EXTRACT-MIN(H)
1  z ← min[H]
2  if z ≠ NIL
3       then for each child x of z
4                 do add x to the root list of H
5                    p[x] ← NIL
6            remove z from the root list of H
7            if z = right[z]
8              then min[H] ← NIL
9              else min[H] ← right[z]
10                   CONSOLIDATE(H)
11           n[H] ← n[H] - 1
12 return z
```

כמתואר באיור 21.3, FIB-HEAP-EXTRACT-MIN פועלת כך שבתחילה היא הופכת כל אחד מבניו של צומת המינימום לשורש, ומוציאה את צומת המינימום מרשימת השורשים. לאחר מכן היא ממזגת את רשימת השורשים על ידי קישור שורשים בעלי דרגה שווה עד שברשימת השורשים נותר לכל היותר שורש אחד מכל דרגה.

מתחילים בשורה 1 בשמירת מצביע z לצומת המינימום; מצביע זה מוחזר בסיום השגרה. אם $z = \text{NIL}$, אזי ערימת פיבונצ'י H כבר ריקה, וסיימנו. אחרת, כמו בשגרה BINOMIAL-HEAP-EXTRACT-MIN, מוחקים את הצומת z מ־H על־ידי הפיכת כל בניו לשורשים ב־H (צירופם לרשימת השורשים) בשורות 3-5, והוצאתו של z מרשימת השורשים בשורה 6. אם לאחר שורה 6 $z = right[z]$, הרי ש־z היה הצומת היחיד ברשימת השורשים ולא היו לו בנים, ולכן לא נותר אלא להפוך את הערימה לריקה בשורה 8 לפני שמחזירים את z. אחרת, משנים את ערכו של המצביע $min[H]$ לרשימת השורשים כך שיצביע על צומת שונה מ־z (במקרה זה, $right[z]$). איור 21.3(ii) מראה את ערימת פיבונצ'י המתקבלת מהערימה שבאיור 21.3(i) לאחר ביצוע שורה 9.

השלב הבא, שבו מצמצמים את מספר העצים בערימת פיבונצ'י, הוא **מיזוג** (consolidation) רשימת השורשים של H; מיזוג זה, המתבצע על ידי הקריאה CONSOLIDATE(H), מורכב מביצוע חוזר של הצעדים שלהלן, עד אשר לכל שורש ברשימת השורשים ערך $degree$ שונה.

1. מצא ברשימת השורשים שני שורשים x ו־y בעלי דרגה שווה, המקיימים $key[x] \le key[y]$.
2. **קשר** (link) את y ל־x: הוצא את y מרשימת השורשים והפוך אותו לבן של x. פעולה זו מתבצעת על־ידי השגרה FIB-HEAP-LINK. ערך השדה $degree[x]$ גדל ב־1, ואם יש סימון על y, מסירים אותו.

השגרה CONSOLIDATE משתמשת במערך עזר $A[0..D(n[H])]$; אם $A[i] = y$, אזי y הנוכחי הוא שורש בעל $degree[y] = i$.

```
CONSOLIDATE(H)
 1  for i ← 0 to D(n[H])
 2      do A[i] ← NIL
 3  for each node w in the root list of H
 4      do x ← w
 5         d ← degree[x]
 6         while A[d] ≠ NIL
 7           do y ← A[d]
 8              if key[x] > key[y]
 9                then exchange x ↔ y
10              FIB-HEAP-LINK(H, y, x)
11              A[d] ← NIL
12              d ← d + 1
13         A[d] ← x
14  min[H] ← NIL
15  for i ← 0 to D(n[H])
16      do if A[i] ≠ NIL
17           then add A[i] to the root list of H
18                if min[H] = NIL or key[A[i]] < key[min[H]]
19                  then min[H] ← A[i]
```

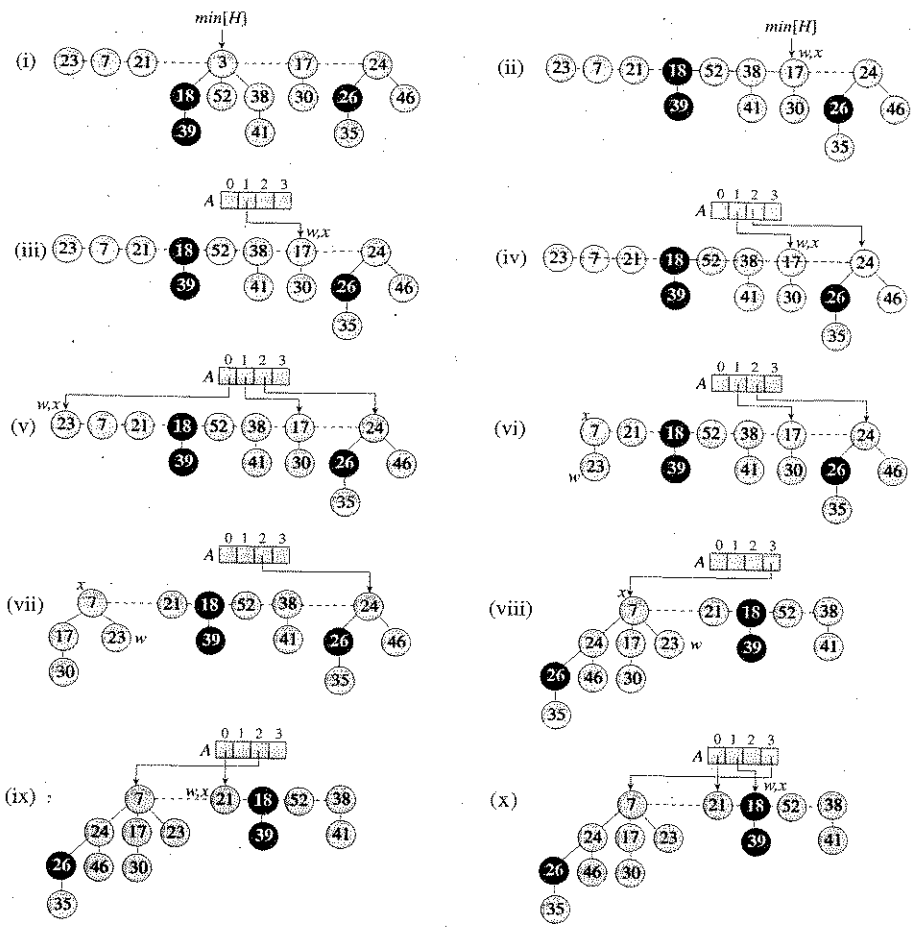


```
FIB-HEAP-LINK(H, y, x)
1  remove y from the root list of H
2  make y a child of x, incrementing degree[x]
3  mark[y] ← FALSE
```

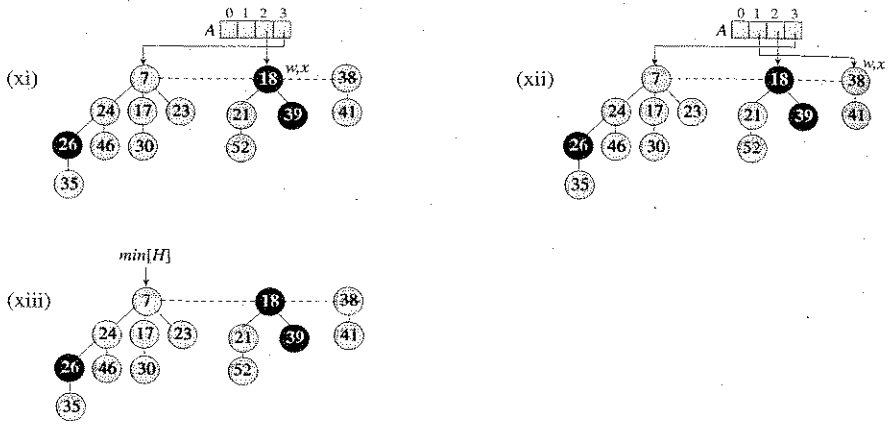
ביתר פירוט, השגרה CONSOLIDATE פועלת באופן הבא: בשורות 1-2 מאתחלים את A על־ידי הצבת NIL בכל תא. כל שורש w, עם סיום הטיפול בו, נמצא בעץ המושרש בצומת כלשהו x, שיכול להיות זהה ל־w או שונה ממנו. לאחר מכן, מעדכנים את התא $A[degree[x]]$ במערך A כך שיצביע על x. בלולאת ה־**for** שבשורות 3-13 בוחנים כל שורש w ברשימת השורשים. האינווריאנטה הנשמרת במהלך כל איטרציה של לולאת ה־**for** היא, שהצומת x הוא השורש של העץ המכיל את הצומת w. לולאת ה־**while** בשורות 6-12 שומרת על האינווריאנטה $d = degree[x]$ (חוץ מאשר בשורה 11, כפי שנראה בעוד רגע). בכל איטרציה של לולאת ה־**while**, $A[d]$ מצביע על שורש כלשהו y. מכיוון ש־$d = degree[x] = degree[y]$, אנו רוצים לקשר את x ו־y. הצומת בעל המפתח הקטן יותר מבין x ו־y הוא שהופך לאביו של הצומת האחר כתוצאה מפעולת הקישור, ולכן, אם יש צורך בכך, שורות 8-9 מחליפות זה בזה את המצביעים ל־x ול־y. לאחר מכן, אנו מקשרים את y ל־x על־ידי הקריאה FIB-HEAP-LINK(H, y, x) בשורה 10. שגרה זו מוסיפה 1 ל־$degree[x]$ אולם משאירה את $degree[y]$ ב־d. מכיוון שהצומת y שוב אינו שורש, המצביע אליו נמחק מן המערך A בשורה 11. מאחר שבמהלך ביצוע FIB-HEAP-LINK ערכו של $degree[x]$ גדל ב־1, שורה 12 מחזירה על כנה את האינווריאנטה $d = degree[x]$. כעת חוזרים על לולאת ה־**while** עד אשר $A[d] = \text{NIL}$, מצב שבו רשימת השורשים אינה מכילה אף שורש נוסף שדרגתו שווה לדרגה של x. בשורה 13 מציבים ב־$A[d]$ מצביע ל־x ומבצעים את האיטרציה הבאה של לולאת ה־**for**. באיורים 21.3(iii)-(v) מוצגים המערך A והעצים המתקבלים לאחר שלוש האיטרציות הראשונות של לולאת ה־**for** שבשורות 3-13. באיטרציה הבאה של לולאת ה־**for** מתבצעות שלוש פעולות קישור; תוצאותיהן מוצגות באיורים 21.3(vi)-(viii). תוצאות ארבע האיטרציות הבאות של לולאת ה־**for** מוצגות באיורים 21.3(ix)-(xii).

עתה, כל שנותר לנו לעשות הוא לנקות אחרינו. עם סיום ביצועה של לולאת ה־**for** שבשורות 3-13, שורה 14 מרוקנת את רשימת השורשים, ושורות 15-19 בונות אותה מחדש. ערימת פיבונצ'י המתקבלת מוצגת באיור 21.3(xiii). לאחר מיזוג רשימת השורשים, FIB-HEAP-EXTRACT-MIN מסיימת בהקטנת $n[H]$ ב־1 בשורה 11, ובהחזרת מצביע לצומת z שנמחק, בשורה 12.

נשים לב שאם כל העצים בערימת פיבונצ'י הם עצים בינומיים בלתי־סדורים לפני ביצועה של FIB-HEAP-EXTRACT-MIN, הרי שכולם נותרים כאלה גם לאחר ביצוע השגרה. קיימות שתי דרכים לשינוי עצים: ראשית, בשורות 3-5 של FIB-HEAP-EXTRACT-MIN, כל בן של השורש z הופך לשורש. לפי תרגיל 21.2-2, כל עץ חדש הוא כשלעצמו עץ בינומי בלתי־סדור. שנית, עצים מקושרים על־ידי FIB-HEAP-LINK רק אם הם בעלי אותה דרגה. מכיוון שלפני ביצוע הקישור כל העצים הם עצים בינומיים בלתי־סדורים, שני עצים אשר לשורשיהם יש k בנים חייבים להיות בעלי מבנה של U_k. לכן העץ המתקבל הוא בעל מבנה של U_{k+1}.



איור 21.3 פעולתה של FIB-HEAP-EXTRACT-MIN. **(i)** ערימת פיבונצ'י H. **(ii)** המצב לאחר הוצאתו של צומת המינימום z מרשימת השורשים והוספת בניו של z לרשימת השורשים. **(v)-(iii)** המערך A והעצים לאחר כל אחת משלוש האיטרציות הראשונות של לולאת ה־**for** שבשורות 3-13 של השגרה CONSOLIDATE. עיבוד רשימת השורשים מתחיל מצומת המינימום ועוקב אחר המצביעים *right*. כל חלק באיור מראה את הערכים של w ושל x בסופה של איטרציה. **(viii)-(vi)** האיטרציה הבאה של לולאת ה־**for**, עם הערכים של w ושל x בסופה של כל איטרציה של לולאת ה־**while** שבשורות 6-12. חלק (vi) מראה את המצב לאחר האיטרציה הראשונה של לולאת ה־**while**. הצומת בעל המפתח 23 כבר קושר לצומת בעל המפתח 7, שעליו x מצביע עתה. בחלק (vii), הצומת בעל המפתח 17 כבר קושר לצומת בעל המפתח 7 שעליו x מצביע עדיין. בחלק (viii), הצומת בעל המפתח 24 כבר קושר לצומת בעל המפתח 7. מכיוון ש־$A[3]$ לא הצביע קודם לכן על שום צומת, בסופה של האיטרציה של לולאת ה־**for** מוצב ב־$A[3]$ מצביע לשורש העץ שהתקבל. **(xii)-(ix)** המצב לאחר כל אחת מארבע האיטרציות הבאות של לולאת ה־**for**. **(xii)** ערימת פיבונצ'י H לאחר שרשימת השורשים נבנתה מחדש מתוך המערך A ונקבע המצביע $min[H]$ החדש.



איור 21.3 (המשך)

עתה אנו ערוכים להראות שהעלות לשיעורין של שליפת צומת המינימום מערימת פיבונצ'י n־צומתית היא $O(D(n))$. נסמן ב־H את ערימת פיבונצ'י בדיוק לפני ביצועה של פעולת FIB-HEAP-EXTRACT-MIN.

את העלות בפועל של שליפת צומת המינימום ניתן לחשב באופן הבא: תרומה של $O(D(n))$ נובעת מכך שלצומת המינימום יש לכל היותר $D(n)$ בנים אשר FIB-HEAP-EXTRACT-MIN מעבדת אותם וכן מהעבודה המתבצעת בשורות 2-1 ובשורות 19-14 של CONSOLIDATE; נותר לנתח את התרומה של לולאת ה־**for** שבשורות 13-3. בעת הקריאה ל־CONSOLIDATE, גודלה של רשימת השורשים הוא לכל היותר $D(n) + t(H) - 1$, מכיוון שהיא מורכבת מרשימת השורשים המקורית בת $t(H)$ הצמתים, פחות צומת השורש שנשלף, ועוד בניו של הצומת שנשלף שמספרם לכל היותר $D(n)$. בכל איטרציה של לולאת ה־**while** שבשורות 6-12 אחד השורשים מקושר לשורש אחר, ולכן כמות העבודה הכוללת המבוצעת בלולאת ה־**for** היא לכל היותר ביחס ישר ל־$D(n) + t(H)$. לפיכך, העלות הכוללת בפועל היא $O(D(n) + t(H))$.

הפוטנציאל לפני שליפת צומת המינימום הוא $t(H) + 2m(H)$, והפוטנציאל לאחר השליפה הוא לכל היותר $(D(n) + 1) + 2m(H)$, שכן נותרים לכל היותר $D(n) + 1$ שורשים ושום צומת אינו הופך למסומן במהלך הפעולה. מכאן שהעלות לשיעורין היא לכל היותר:

$$\begin{aligned} & O(D(n) + t(H)) + ((D(n) + 1) + 2m(H)) - (t(H) + 2m(H)) \\ &= O(D(n)) + O(t(H)) - t(H) \\ &= O(D(n)) \end{aligned}$$

שכן ניתן לבחור יחידות פוטנציאל גדולות דיין, כך שיהיו דומיננטיות ביחס לקבוע החבוי ב־$O(t(H))$. אינטואיטיבית, עלות הביצוע של כל קישור משולמת על־ידי ירידת הפוטנציאל כתוצאה מכך שהקישור מקטין ב־1 את מספר השורשים.

תרגילים

21.2-1
הראה את ערימת פיבונצ'י המתקבלת מקריאה ל־FIB-HEAP-EXTRACT-MIN על ערימת פיבונצ'י המוצגת באיור 21.3(xiii).

21.2-2
הוכח כי למה 20.1 מתקיימת עבור עצים בינומיים בלתי־סדורים, אך עם תכונה 4' במקום תכונה 4.

21.2-3
הראה שאם יש לתמוך רק בפעולות על ערימות מתמזגות, הדרגה המקסימלית $D(n)$ בערימת פיבונצ'י n־צומתית היא לכל היותר $\lfloor \lg n \rfloor$.

21.2-4
פרופסור מקגי פיתח מבנה נתונים חדש המבוסס על ערימות פיבונצ'י. ערימת מקגי היא בעלת מבנה זהה לזה של ערימת פיבונצ'י ותומכת בפעולות על ערימות מתמזגות. מימושי הפעולות זהים למימושיהן עבור ערימות פיבונצ'י, פרט לכך שבפעולות ההכנסה והאיחוד מתבצע מיזוג העצים כצעד אחרון. מהם זמני הריצה במקרה הגרוע של הפעולות על ערימות מקגי'ז עד כמה יש מן החידוש במבנה הנתונים שפיתח הפרופסור?

21.2-5
הראה שכאשר הפעולות היחידות על מפתחות הן השוואות של שני מפתחות (כמו במקרה של כל המימושים המוצגים בפרק זה), לא כל הפעולות על ערימות מתמזגות יכולות לרוץ בזמן לשיעורין של $O(1)$.

21.3 הקטנת ערכו של מפתח ומחיקת צומת

בסעיף זה נראה כיצד מקטינים את ערך המפתח של צומת בערימת פיבונצ'י בזמן לשיעורין $O(1)$ וכיצד מוחקים צומת כלשהו מערימת פיבונצ'י n־צומתית בזמן לשיעורין $O(D(n))$. פעולות אלה אינן משמרות את התכונה שכל העצים בערימת פיבונצ'י הם עצים בינומיים בלתי־סדורים, אולם הן קרובות לכך במידה מספקת כך שניתן לחסום את דרגת המקסימום $D(n)$ על־ידי $O(\lg n)$. מהוכחת חסם זה ינבע שהשגרות FIB-HEAP-EXTRACT-MIN ו־FIB-HEAP-DELETE רצות בזמן לשיעורין של $O(\lg n)$.

הקטנת ערכו של מפתח

בפסאודו־קוד שלהלן עבור הפעולה FIB-HEAP-DECREASE-KEY, אנו מניחים כמו קודם שהוצאתו של צומת מרשימה מקושרת אינה משנה אף אחד מהשדות המבניים בצומת שהוצא.


```
FIB-HEAP-DECREASE-KEY(H, x, k)
1  if k > key[x]
2     then error "new key is greater than current key"
3  key[x] ← k
4  y ← p[x]
5  if y ≠ NIL and key[x] < key[y]
6     then CUT(H, x, y)
7          CASCADING-CUT(H, y)
8  if key[x] < key[min[H]]
9     then min[H] ← x

CUT(H, x, y)
1  remove x from the child list of y, decrementing degree[y]
2  add x to the root list of H
3  p[x] ← NIL
4  mark[x] ← FALSE

CASCADING-CUT(H, y)
1  z ← p[y]
2  if z ≠ NIL
3     then if mark[y] = FALSE
4             then mark[y] ← TRUE
5             else CUT(H, y, z)
6                  CASCADING-CUT(H, z)
```

השגרה FIB-HEAP-DECREASE-KEY פועלת באופן הבא: שורות 1-3 מוודאות שהמפתח החדש אינו גדול מהמפתח הנוכחי של x ואז מציבות את המפתח החדש ב־x. אם x הוא שורש או אם $key[x] \geq key[y]$, כאשר y הוא אביו של x, אזי אין צורך לעשות שום שינוי מבני, שכן סדר הערימה לא הופר. שורות 4-5 בודקות אם מתקיים המצב הזה.

אם הופר סדר הערימה, עשויים להתרחש שינויים רבים. אנו מתחילים **בניתוק** (cutting) x בשורה 6. השגרה CUT "מנתקת" את הקשר בין x לאביו y והופכת את x לשורש.

אנו משתמשים בשדות *mark* לקבלת חסמי הזמן המבוקשים. הם עוזרים להשיג את התוצאה הבאה. נניח כי x הוא צומת שעבר במהלך חייו את הגלגולים הבאים:

1. בזמן מסוים, x היה שורש,
2. לאחר מכן, x קושר לצומת אחר,
3. לאחר מכן, שניים מבניו של x הוסרו באמצעות פעולות ניתוק.

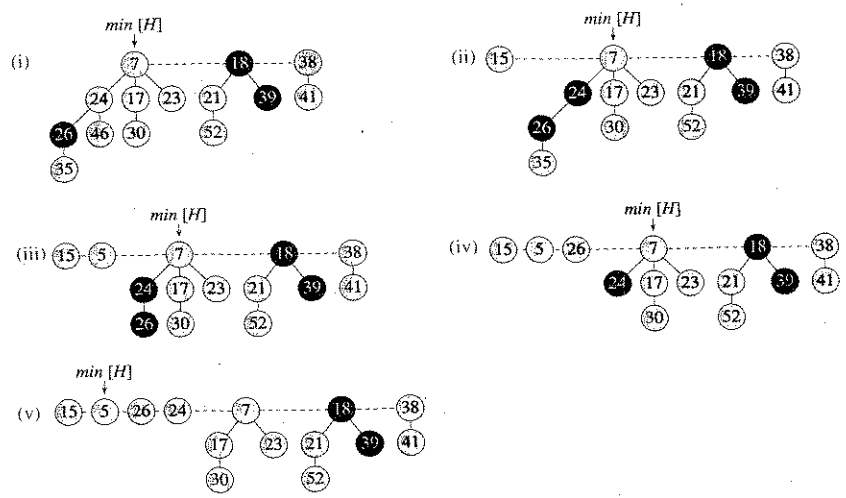
ברגע ש־x מאבד את בנו השני, x מנותק מאביו והופך להיות שורש חדש. ערכו של השדה $mark[x]$ הוא TRUE אם התרחשו שלבים 1 ו־2 ובן אחד של x נותק ממנו. השגרה CUT מסירה את הסימון $mark[x]$ בשורה 4, מכיוון שהיא מבצעת את שלב 1. (עתה נוכל לראות מדוע שורה 3 של FIB-HEAP-LINK מסירה את הסימון $mark[y]$: השגרה מקשרת את y לצומת אחר, דהיינו, מבצעת את שלב 2. בפעם הבאה שבן של y ינותק ממנו, יוצב ב־$mark[y]$ הערך TRUE.)

עדיין לא סיימנו, שכן x עשוי להיות הבן השני שנותק מאביו y מאז ש־y קושר לצומת אחר. לפיכך, שורה 7 של FIB-HEAP-DECREASE-KEY מבצעת פעולת **ניתוק־שרשרת** (cascading-cut) על y. אם y הוא שורש, אזי הבדיקה בשורה 2 של CASCADING-CUT גורמת לכך שהשגרה פשוט תחזור.

אם y אינו מסומן, השגרה מסמנת אותו בשורה 4, שכן זה עתה נותק ממנו הבן הראשון, וחוזרת. אולם אם y מסומן, פירוש הדבר שזה עתה נותק ממנו הבן השני; לפיכך, y מנותק מאביו בשורה 5, והשגרה CASCADING-CUT קוראת לעצמה באופן רקורסיבי בשורה 6 על z, אביו של y. השגרה CASCADING-CUT מתקדמת באופן רקורסיבי במעלה העץ עד שהיא מגיעה לצומת בלתי מסומן או לשורש.

לאחר שבוצעו כל הפעולות של ניתוק־שרשרת, שורות 8-9 של השגרה FIB-HEAP-DECREASE-KEY מסיימות את התהליך על־ידי עדכון $min[H]$, אם יש צורך בכך.

איור 21.4 מתאר ביצוע של שתי קריאות ל־FIB-HEAP-DECREASE-KEY כשמתחילים עם ערימת פיבונצ'י המוצגת באיור 21.4(i). הקריאה הראשונה, המתוארת באיור 21.4(ii), אינה כרוכה בניתוקי־שרשרת. הקריאה השנייה, המתוארת באיורים 21.4(iii)-(v), מפעילה שני ניתוקי־שרשרת.



איור 21.4 שתי קריאות ל־FIB-HEAP-DECREASE-KEY. **(i)** ערימת פיבונצ'י ההתחלתית. **(ii)** בצומת בעל המפתח 46 מקטינים את ערך המפתח ל־15. הצומת הופך לשורש, ואביו (בעל המפתח 24), שקודם לכן היה בלתי־מסומן, הופך להיות מסומן. **(iii)-(v)** בצומת בעל המפתח 35 מקטינים את ערך המפתח ל־5. בחלק (iii), צומת זה, שעתה הוא בעל המפתח 5, הופך לשורש. אביו, בעל המפתח 26, מסומן, ולכן מתרחש ניתוק־שרשרת. הצומת בעל המפתח 26 מנותק מאביו והופך לשורש בלתי־מסומן בחלק (iv). עתה מתרחש ניתוק־שרשרת נוסף, שכן הצומת בעל המפתח 24 מסומן אף הוא. צומת זה מנותק מאביו והופך לשורש בלתי־מסומן בחלק (v). בנקודה זו ניתוקי השרשרת נעצרים, שכן הצומת בעל המפתח 7 הוא שורש. (גם אילו צומת זה לא היה שורש, ניתוקי־השרשרת היו נפסקים מכיוון שהוא אינו מסומן.) התוצאה של פעולת FIB-HEAP-DECREASE-KEY מוצגת בחלק (v), כאשר $min[H]$ מצביע על צומת המינימום החדש.

נראה עתה שהעלות לשיעורין של FIB-HEAP-DECREASE-KEY היא $O(1)$ בלבד. נתחיל בקביעת העלות בפועל. הזמן הדרוש לביצוע השגרה FIB-HEAP-DECREASE-KEY הוא $O(1)$ ועוד הזמן הדרוש לביצוע ניתוקי־השרשרת. נניח כי CASCADING-CUT נקראת באופן רקורסיבי c פעמים במהלך הפעלה מסוימת של FIB-HEAP-DECREASE-KEY. כל קריאה ל־CASCADING-CUT צורכת

זמן של $O(1)$, לא כולל קריאות רקורסיביות. העלות בפועל של FIB-HEAP-DECREASE-KEY, כולל כל הקריאות הרקורסיביות, היא אפוא $O(c)$.

עתה נחשב את השינוי בפוטנציאל. נסמן ב־H את הערימה בדיוק לפני הפעלת השגרה FIB-HEAP-DECREASE-KEY. כל קריאה רקורסיבית ל־CASCADING-CUT, פרט לאחרונה, מנתקת צומת מסומן ומאפסת את סיבית הסימון שלו. לאחר מכן, הערימה מכילה $t(H)+c$ עצים ($t(H)$ העצים המקוריים, $c-1$ העצים שנוצרו על ידי ניתוקי־השרשרת, והעץ המושרש ב־x) ולכל היותר $m(H)-c+2$ צמתים מסומנים ($c-1$ צמתים הפכו לבלתי־מסומנים על־ידי ניתוקי־השרשרת והקריאה האחרונה ל־CASCADING-CUT עשויה היתה לסמן צומת). אם כן, השינוי בפוטנציאל הוא לכל היותר:

$$((t(H)+c)+2(m(H)-c+2))-(t(H)+2m(H)) = 4-c$$

לכן העלות לשיעורין של FIB-HEAP-DECREASE-KEY היא לכל היותר:

$$O(c)+4-c = O(1)$$

שכן ניתן לבחור יחידות פוטנציאל גדולות דיין כך שיהיו דומיננטיות ביחס לקבוע החבוי ב־$O(c)$.

עתה אפשר להבין מדוע כללנו בהגדרת פונקציית הפוטנציאל איבר שהוא פעמיים מספר הצמתים המסומנים. כאשר צומת מסומן y מנותק על־ידי ניתוק־שרשרת, סיבית הסימון שלו מאופסת ולכן הפוטנציאל שלו קטן ב־2. יחידת פוטנציאל אחת משלמת עבור הניתוק ואיפוס סיבית הסימון, והיחידה השנייה מפצה על עליית הפוטנציאל ביחידה אחת כתוצאה מכך שהצומת y הפך לשורש.

מחיקת צומת

קל למחוק צומת מערימת פיבונצ'י n־צומתית בזמן לשיעורין $O(D(n))$, כפי שעושה הפסאודו־קוד שלהלן. אנו מניחים שערימת פיבונצ'י הנתונה אינה מכילה מפתח שערכו $-\infty$.

```
FIB-HEAP-DELETE(H, x)
1  FIB-HEAP-DECREASE-KEY(H, x, -∞)
2  FIB-HEAP-EXTRACT-MIN(H)
```

FIB-HEAP-DELETE מקבילה ל־BINOMIAL-HEAP-DELETE. היא הופכת את x לצומת המינימום בערימת פיבונצ'י הנתונה על־ידי הצבת הערך הייחודי הקטן $-\infty$ בשדה המפתח שלו. לאחר מכן היא מוציאה את x מן הערימה באמצעות קריאה לשגרה FIB-HEAP-EXTRACT-MIN. הזמן לשיעורין של השגרה FIB-HEAP-DELETE הוא הסכום של הזמן לשיעורין $O(1)$ של השגרה FIB-HEAP-DECREASE-KEY והזמן לשיעורין $O(D(n))$ של השגרה FIB-HEAP-EXTRACT-MIN.

תרגילים

21.3-1

נניח ששורש x בערימת פיבונצ'י מסומן. הסבר כיצד הפך x להיות שורש מסומן. הראה שעבור הניתוח אין חשיבות לכך ש־x מסומן, למרות שאינו שורש שקושר תחילה לצומת אחר ואז איבד בן אחד.

21.3-2

הצדק את הזמן לשיעורין $O(1)$ של FIB-HEAP-DECREASE-KEY תוך שימוש בשיטת הצבירה מסעיף 18.1.

21.4 חסימת הדרגה המקסימלית

כדי להוכיח שהזמן לשיערוך של FIB-HEAP-EXTRACT-MIN ושל FIB-HEAP-DELETE הוא $O(\lg n)$, עלינו להראות שהחסם העליון $D(n)$ על דרגתו של צומת כלשהו בערימת פיבונצ'י n-צומתית הוא $O(\lg n)$. לפי תרגיל 21.2-3, כאשר כל העצים בערימת פיבונצ'י הם עצים בינומיים בלתי־סדורים, אזי $D(n) = \lfloor \lg n \rfloor$. אולם הניתוקים המתרחשים במהלך FIB-HEAP-DECREASE-KEY עלולים לגרום לכך שעצים בתוך ערימת פיבונצ'י יפרו את תכונות העצים הבינומיים הבלתי־סדורים. בסעיף זה נראה שמכיוון שאנו מנתקים צומת מאביו ברגע שהצומת מאבד שני בנים, $D(n)$ הוא $O(\lg n)$. בפרט, נראה כי $D(n) \le \lfloor \log_\phi n \rfloor$, כאשר $\phi = (1 + \sqrt{5})/2$.

המפתח לניתוח הוא זה: עבור כל צומת x בערימת פיבונצ'י נגדיר את $size[x]$ כמספר הצמתים, כולל x עצמו, בתת־עץ המושרש ב־x. ניש לשים לב לכך ש־x אינו חייב להשתייך לרשימת השורשים; הוא יכול להיות כל צומת שהוא.) אנו נראה כי $size[x]$ אקספוננציאלי ב־$degree[x]$. נזכור כי $degree[x]$ מתוחזק כך שהוא מכיל תמיד את הדרגה המדויקת של x.

למה 21.1

יהי x צומת כלשהו בערימת פיבונצ'י, ונניח כי $degree[x] = k$. נסמן על־ידי $y_1, y_2, \ldots, y_k$ את בניו של x בסדר שבו הם קושרו ל־x, מן המוקדם אל המאוחר. אזי $degree[y_1] \ge 0$ ו־$degree[y_i] \ge i - 2$ עבור $i = 2, 3, \ldots, k$.

הוכחה ברור מאליו כי $degree[y_1] \ge 0$.

עבור $i \ge 2$, נשים לב כי כאשר y_i קושר ל־x, הצמתים $y_1, y_2, \ldots, y_{i-1}$ היו כולם בנים של x, כך שבאותו זמן התקיים בהכרח $degree[x] \ge i - 1$. הצומת y_i קושר ל־x רק אם $degree[x] = degree[y_i]$, כך שבאותו זמן התקיים בהכרח גם $degree[y_i] \ge i - 1$. מאז, הצומת y_i איבד לכל היותר בן אחד, שכן אילו היה מאבד שני בנים, הוא היה מנותק מ־x. מכך נוכל להסיק כי $degree[y_i] \ge i - 2$. ■

לבסוף אנו מגיעים לשלב הניתוח המסביר את השם "ערימות פיבונצ'י". נזכור כי בסעיף 2.2 הוגדר מספר פיבונצ'י ה־k, עבור $k = 0, 1, 2, \ldots$, על־ידי נוסחת הנסיגה:

$$F_k = \begin{cases} 0 & \text{אם } k = 0 \\ 1 & \text{אם } k = 1 \\ F_{k-1} + F_{k-2} & \text{אם } k \ge 2 \end{cases}$$

בלמה הבאה מוצגת דרך נוספת לבטא את F_k.

למה 21.2

עבור כל השלמים $k \ge 0$, מתקיים:

$$F_{k+2} = 1 + \sum_{i=0}^{k} F_i$$

הוכחה ההוכחה היא באינדוקציה על k. כאשר $k = 0$, נקבל:

$$\begin{aligned} 1 + \sum_{i=0}^{0} F_i &= 1 + F_0 \\ &= 1 + 0 \\ &= 1 \\ &= F_2 \end{aligned}$$

עתה נניח את הנחת האינדוקציה, $F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$, ונקבל:

$$\begin{aligned} F_{k+2} &= F_k + F_{k+1} \\ &= F_k + \left(1 + \sum_{i=0}^{k-1} F_i\right) \\ &= 1 + \sum_{i=0}^{k} F_i \end{aligned}$$

■

הלמה הבאה והמסקנה הנובעת ממנה משלימות את הניתוח. הלמה משתמשת באי־שוויון (שהוכח בתרגיל 2.2-8):

$$F_{k+2} \ge \phi^k$$

כאשר ϕ הוא יחס הזהב המוגדר במשוואה (2.14) על־ידי $\phi = (1 + \sqrt{5})/2 = 1.61803\ldots$.

למה 21.3

יהי x צומת כלשהו בערימת פיבונצ'י, ותהי דרגתו $k = degree[x]$. אזי $size(x) \ge F_{k+2} \ge \phi^k$, כאשר $\phi = (1 + \sqrt{5})/2$.

הוכחה נסמן ב־s_k את הערך המינימלי האפשרי של $size(z)$ על כל הצמתים z המקיימים $degree[z] = k$. באופן טריוויאלי, $s_0 = 1$, $s_1 = 2$ ו־$s_2 = 3$. המספר s_k הוא לכל היותר $size(x)$. כמו בלמה 21.1, נסמן ב־$y_1, y_2, \ldots, y_k$ את בניו של x בסדר שבו קושרו אל x. כדי לחשב חסם תחתון על $size(x)$, נמנה 1 עבור x עצמו ו־1 עבור הבן הראשון y_1 (המקיים $size(y_1) \ge 1$), ואז ניישם את למה 21.1 עבור שאר הבנים. נקבל:

$$\begin{aligned} size(x) &\ge s_k \\ &\ge 2 + \sum_{i=2}^{k} s_{i-2} \end{aligned}$$

עתה נוכיח באינדוקציה על k כי $s_k \ge F_{k+2}$ עבור כל השלמים האי־שליליים k. מקרי הבסיס, $k = 0$ ו־$k = 1$, טריוויאליים. בצעד האינדוקציה, נניח כי $k \ge 2$ וכי $s_i \ge F_{i+2}$ עבור $i = 0, 1, \ldots, k - 1$. כעת מתקיים:

$$
\begin{aligned}
s_k &\geq 2 + \sum_{i=2}^{k} s_{i-2} \\
&\geq 2 + \sum_{i=2}^{k} F_i \\
&= 1 + \sum_{i=0}^{k} F_i \\
&= F_{k+2}
\end{aligned}
$$

השוויון האחרון נובע מלמה 21.2.

הראינו אפוא כי $size(x) \geq s_k \geq F_{k+2} \geq \phi^k$. ■

מסקנה 21.4

הדרגת המקסימלית $D(n)$ של צומת כלשהו בערימת פיבונצ'י n-צומתית היא $O(\lg n)$.

הוכחה יהי x צומת כלשהו בערימת פיבונצ'י n-צומתית ויהי $k = degree[x]$. לפי למה 21.3 מתקיים $n \geq size(x) \geq \phi^k$. ניקח לוגריתם לפי בסיס ϕ ונקבל $k \leq \log_\phi n$. (למעשה, מאחר ש-k שלם, $k \leq \lfloor \log_\phi n \rfloor$.) הדרגה המקסימלית $D(n)$ של צומת כלשהו היא אפוא $O(\lg n)$. ■

תרגילים

21.4-1

פרופסור פינוקיו טוען שגובהה של ערימת פיבונצ'י n-צומתית הוא $O(\lg n)$. הראה שהפרופסור טועה על ידי כך שתציג, עבור כל n שלם חיובי, סדרת פעולות על ערימת פיבונצ'י שיוצרת ערימת פיבונצ'י המורכבת מעץ אחד בלבד שהוא שרשרת לינארית של n צמתים.

21.4-2

נניח שאנו מכלילים את כלל ניתוק-השרשרת כך שצומת x ינותק מאביו ברגע שהוא מאבד את הבן ה-k שלו, עבור קבוע שלם k כלשהו. (הכלל בסעיף 21.3 משתמש ב-$k = 2$.) עבור אילו ערכים של k מתקיים $D(n) = O(\lg n)$?

בעיות

21-1 מימוש חלופי של מחיקה

פרופסור פיאנו הציע את הגרסה שלהלן לשגרה FIB-HEAP-DELETE, בטענה שהיא מהירה יותר כאשר הצומת שנמחק אינו זה שעליו מצביע $min[H]$.


```
PISANO-DELETE(H, x)
 1  if x = min[H]
 2     then FIB-HEAP-EXTRACT-MIN(H)
 3     else y ← p[x]
 4          if y ≠ NIL
 5             then CUT(H, x, y)
 6                  CASCADING-CUT(H, y)
 7          add x's child list to the root list of H
 8          remove x from the root list of H
```

א. טענתו של הפרופסור ששגרה זו מהירה יותר מבוססת בחלקה על ההנחה שניתן לבצע את שורה 7 בזמן בפועל $O(1)$. מה שגוי בהנחה זו?

ב. מצא חסם עליון טוב על הזמן בפועל של PISANO-DELETE כאשר $x \neq min[H]$. עליך לבטא את החסם במונחי $degree[x]$ והמספר c של קריאות לשגרה CASCADING-CUT.

ג. תהי H' ערימת פיבונצ'י המתקבלת לאחר ביצוע של PISANO-DELETE(H, x). בהנחה שהצומת x אינו שורש, מצא חסם על הפוטנציאל של H' במונחי $degree[x]$, c, $t(H)$ ו-$m(H)$.

ד. הסק שהזמן לשיעורין של PISANO-DELETE אינו טוב יותר אסימפטוטית מזה של FIB-HEAP-DELETE, גם כאשר $x \neq min[H]$.

22-1 עוד פעולות על ערימת פיבונצ'י

ברצוננו להרחיב את ערימת פיבונצ'י כך שתתמוך בשתי פעולות חדשות, בלי לשנות את הזמן לשיעורין של כל אחת מהפעולות האחרות על ערימות פיבונצ'י.

א. הצג מימוש יעיל של הפעולה FIB-HEAP-CHANGE-KEY(H, x, k), אשר משנה את המפתח של הצומת x לערך k. נתח את הזמן לשיעורין של המימוש שלך עבור המקרים שבהם k גדול מ-$key[x]$, קטן ממנו או שווה לו.

ב. הצג מימוש יעיל של הפעולה FIB-HEAP-PRUNE(H, r), אשר מוחקת $min(r, n[H])$ צמתים מ-H. הבחירה אילו צמתים למחוק צריכה להיות שרירותית. נתח את הזמן לשיעורין של המימוש שלך. (**רמז:** ייתכן שיהא עליך לשנות את מבנה הנתונים ואת פונקציית הפוטנציאל.)

הערות ביבליוגרפיות לפרק 21

ערימות פיבונצ'י הוצגו על-ידי Fredman & Tarjan [75]. מאמרם מתאר גם את יישומן של ערימות פיבונצ'י בפתרון הבעיות הבאות: מסלולים קצרים ביותר ממקור יחיד, מסלולים קצרים ביותר בין כל הזוגות, והתאמה דו-אזורית משוקללת ובעיית העץ הפורש המינימלי.

בעקבותיהם פיתחו Driscoll, Gabow, Shrairman & Tarjan [58] "ערימות נינוחות" (relaxed heaps) כחלופה לערימות פיבונצ'י. קיימות שתי גרסאות של ערימות נינוחות. האחת נותנת אותם חסמי זמן לשיעורין כמו ערימות פיבונצ'י. השנייה מאפשרת ביצוע DECREASE-KEY בזמן $O(1)$ במקרה הגרוע (ולא לשיעורין), וביצוע EXTRACT-MIN ו-DELETE בזמן $O(\lg n)$ במקרה הגרוע. לערימות נינוחות יש גם יתרונות מסוימים על ערימות פיבונצ'י באלגוריתמים מקביליים.

22 מבני נתונים לקבוצות זרות

קיימים יישומים הכרוכים בקיבוץ n איברים שונים זה מזה לאוסף של קבוצות זרות. שתי פעולות חשובות על אוסף כזה הן מציאת הקבוצה שאליה שייך איבר נתון ואיחוד שתי קבוצות. פרק זה חוקר שיטות לניהול מבנה נתונים התומך בפעולות אלה.

בסעיף 22.1 נתאר את הפעולות שבהן תומך מבנה נתונים לקבוצות זרות ונציג יישום פשוט. בסעיף 22.2 נתבונן במימוש פשוט של אוסף קבוצות זרות באמצעות רשימות מקושרות. ייצוג יעיל יותר, באמצעות עצים מושרשים, מוצג בסעיף 22.3. בייצוג באמצעות עצים, זמן הריצה הוא לינארי לכל מטרה מעשית, אולם תיאורטית הוא על־לינארי. בסעיף 22.4 נגדיר וננתח את פונקציית אקרמן ואת הפונקציה ההופכית שלה, אשר גדלה לאט מאוד, ואשר מופיעה בזמן הריצה של פעולות במימוש המבוסס על עצים. לאחר מכן נשתמש בניתוח לשיערון כדי להוכיח חסם עליון חלש מעט יותר על זמן הריצה.

22.1 פעולות על אוסף קבוצות זרות

מבנה נתונים לקבוצות זרות (disjoint-set data structure) מנהל אוסף $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ של קבוצות דינמיות זרות. כל קבוצה מזוהה על־ידי **נציג** (representative) שהוא איבר מסוים בקבוצה. ביישומים מסוימים אין זה משנה איזה איבר בקבוצה משמש כנציג; הדרישה היחידה היא שאם אנו ניגשים פעמיים לנציג של קבוצה דינמית בלי ששינינו את הקבוצה בין שתי הגישות, נקבל בשתיהן אותו איבר. ביישומים אחרים עשוי להיות כלל קבוע מראש לבחירת הנציג, כמו למשל, בחירת האיבר הקטן ביותר בקבוצה (בהנחה, כמובן, שניתן לסדר את איברי הקבוצה על פי גודלם).

כמו במימושים האחרים של קבוצות דינמיות שעסקנו בהם, כל איבר בקבוצה מיוצג על ידי עצם. אם נסמן עצם על־ידי x, אנו מעוניינים לתמרך בפעולות הבאות:

MAKE-SET(x) — יוצרת קבוצה חדשה שהאיבר היחיד בה (ולכן גם הנציג שלה) הוא x. מכיוון שהקבוצות זרות, הדרישה היא ש־x אינו שייך כבר לאחת הקבוצות.

UNION(x, y) — מאחדת את הקבוצות הדינמיות המכילות את x ואת y, נאמר S_x ו־S_y, לקבוצה חדשה שהיא איחוד שתי הקבוצות האלה. ההנחה היא שלפני האיחוד שתי הקבוצות זרות. נציג הקבוצה המתקבלת הוא איבר כלשהו של $S_x \cup S_y$, אם כי מימושים רבים של UNION בוחרים

בנציג של S_x או בנציג של S_y כנציג החדש של האיחוד. מאחר שאנו דורשים שהקבוצות באוסף יהיו זרות, אנו "משמידים" את הקבוצות S_x ו־S_y ומוציאים אותן מן האוסף $\mathcal{S}$.

FIND-SET(x) — מחזירה מצביע לנציג הקבוצה (היחידה) המכילה את x.

במהלך פרק זה ננתח את זמני הריצה של הפעולות על מבני נתונים לקבוצות זרות במונחי שני פרמטרים: n — מספר הפעולות MAKE-SET, ו־m — המספר הכולל של פעולות UNION, MAKE-SET ו־FIND-SET. מאחר שהקבוצות זרות, כל פעולת UNION מקטינה ב־1 את מספר הקבוצות, ולכן לאחר $n - 1$ פעולות UNION נותרת קבוצה אחת בלבד. מספר הפעולות UNION הוא אפוא $n - 1$ לכל היותר. עוד נשים לב כי מאחר שפעולות MAKE-SET נכללות במספר הפעולות הכולל m, הרי שמתקיים $m \geq n$.

יישום של מבני נתונים לקבוצות זרות

אחד מן היישומים הרבים של מבני הנתונים לקבוצות זרות הוא מציאת הרכיבים הקשירים של גרף בלתי־מכוון (ראה סעיף 5.4). באיור 22.1(i) לדוגמה, מתואר גרף בעל ארבעה רכיבים קשירים.

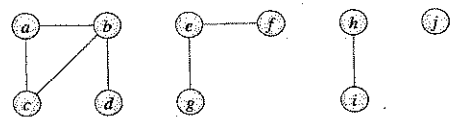
השגרה CONNECTED-COMPONENTS שלהלן משתמשת בפעולות על קבוצות זרות לחישוב הרכיבים הקשירים של גרף. לאחר הרצת CONNECTED-COMPONENTS כשלב של עיבוד מקדים, השגרה SAME-COMPONENT עונה על שאילתות המבקשות למצוא אם שני קדקודים שייכים לאותו רכיב קשיר בגרף.[1] (קבוצת הקדקודים של גרף G מסומנת על־ידי $V(G)$ וקבוצת הקשתות מסומנת על־ידי $E(G)$.)

```
CONNECTED-COMPONENTS(G)
1  for each vertex v ∈ V[G]
2      do MAKE-SET(v)
3  for each edge (u, v) ∈ E[G]
4      do if FIND-SET(u) ≠ FIND-SET(v)
5             then UNION(u, v)

SAME-COMPONENT(u, v)
1  if FIND-SET(u) = FIND-SET(v)
2      then return TRUE
3      else return FALSE
```

השגרה CONNECTED-COMPONENTS יוצרת תחילה לכל קדקוד v קבוצה משלו. לאחר מכן, עבור כל קשת (u, v), היא מאחדת את הקבוצות המכילות את u ואת v. לפי תרגיל 22.1-2, לאחר שעובדו כל הקשתות, שני קדקודים שייכים לאותו רכיב קשיר אם ורק אם העצמים המתאימים שייכים לאותה קבוצה. CONNECTED-COMPONENTS יוצרת אפוא קבוצות בדרך המאפשרת לשגרה SAME-COMPONENT לקבוע אם שני קדקודים שייכים לאותו רכיב קשיר. איור 22.1(ii) מדגים כיצד CONNECTED-COMPONENTS מחשבת את הקבוצות הזרות.

1. כאשר הקשתות בגרף הן "סטטיות" — אינן משתנות בזמן — ניתן לחשב את הרכיבים הקשירים מהר יותר באמצעות חיפוש לעומק־תחילה (depth-first search) (תרגיל 23.3-9). אולם לעתים מוסיפים קשתות "באופן דינמי", ואז צריך לעדכן את הרכיבים הקשירים בכל פעם שנוספת קשת. במקרה זה, המימוש המובא כאן עשוי להיות יעיל יותר מאשר הרצה מחדש של חיפוש לעומק־תחילה עבור כל קשת חדשה.



(i)

הקשת המעובדת	אוסף הקבוצות הזרות									
הקבוצות ההתחלתיות	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(b,d)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(e,g)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(a,c)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(h,i)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(a,b)	$\{a,b,c,d\}$				$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(e,f)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$
(b,c)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$

(ii)

איור 22.1 **(i)** גרף בעל ארבעה רכיבים קשירים: $\{a,b,c,d\}$, $\{e,f,g\}$, $\{h,i\}$ ו־$\{j\}$. **(ii)** אוסף הקבוצות הזרות לאחר עיבוד כל אחת מהקשתות.

תרגילים

22.1-1

נניח שמריצים את CONNECTED-COMPONENTS על הגרף הבלתי־מכוון $G = (V, E)$, כאשר $V = \{a, b, c, d, e, f, g, h, i, j, k\}$, והקשתות ב־$E$ מעובדות בסדר הבא (מימין לשמאל): (d, i), (f, k), (g, i), (b, g), (a, h), (i, j), (d, k), (b, j), (d, f), (g, j), (a, e). רשום את הקדקודים שבכל רכיב קשיר לאחר כל איטרציה של הלולאה שבשורות 3-5 ו־(i, d).

22.1-2

הראה שלאחר שכל הקשתות עובדו על־ידי CONNECTED-COMPONENTS, שני קדקודים נמצאים באותו רכיב קשיר אם ורק אם הם שייכים לאותה קבוצה.

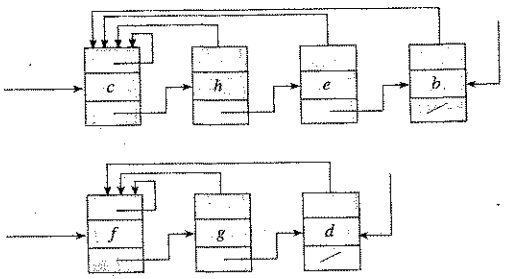
22.1-3

במהלך הרצתה של CONNECTED-COMPONENTS על גרף בלתי־מכוון $G = (V, E)$ בעל k רכיבים קשירים, כמה פעמים נקראת השגרה FIND-SET? כמה פעמים נקראת UNION? בטא את תשובותיך במונחי $|V|$, $|E|$ ו־k.

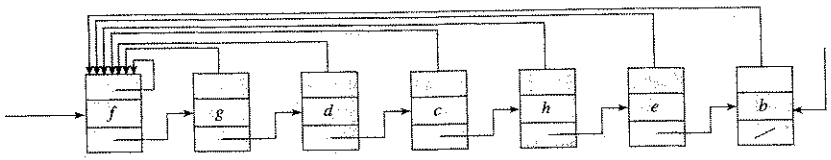
22.2 ייצוג קבוצות זרות על־ידי רשימות מקושרות

דרך פשוטה למימוש מבנה נתונים לקבוצות זרות היא לייצג כל קבוצה על־ידי רשימה מקושרת. העצם הראשון בכל רשימה מקושרת משמש כנציג הקבוצה שהרשימה מייצגת. כל עצם ברשימה המקושרת מכיל איבר של קבוצה, מצביע לעצם המכיל את האיבר הבא בקבוצה, ומצביע לנציג

הקבוצה. איור 22.2(i) מתאר ייצוג שתי קבוצות על־ידי רשימות מקושרות. בתוך כל רשימה מקושרת, העצמים מופיעים בסדר כלשהו (בכפוף להנחתנו שהעצם הראשון בכל רשימה הוא הנציג). בנוסף למצביע לראש הרשימה (המכיל את נציג הקבוצה), הייצוג מכיל גם מצביע לאיבר האחרון ברשימה.



(i)



(ii)

איור 22.2 **(i)** ייצוג שתי קבוצות על־ידי רשימות מקושרות. הקבוצה האחת מכילה את העצמים b, c, e ו־h, והנציג שלה הוא c; השנייה מכילה את העצמים d, f ו־g, והנציג שלה הוא f. כל עצם ברשימה מכיל איבר של קבוצה, מצביע לעצם הבא ברשימה ומצביע לעצם הראשון ברשימה, שהוא נציג הקבוצה. מבנה הנתונים מכיל גם מצביע לראש הרשימה ומצביע לאיבר האחרון בה. **(ii)** הרשימה המתקבלת מ־UNION(e, g). נציג הקבוצה המתקבלת הוא f.

בייצוג כזה על־ידי רשימות מקושרות קל לממש את הפעולות MAKE-SET ו־FIND-SET, וביצוען דורש זמן $O(1)$. לביצוע MAKE-SET(x) אנו יוצרים רשימה מקושרת חדשה שהעצם היחיד בה הוא x. FIND-SET(x) פשוט מחזירה את המצביע מ־x לנציג.

מימוש פשוט של UNION

כאשר משתמשים בייצוג הקבוצות על־ידי רשימות מקושרות, זמן הריצה של המימוש הפשוט ביותר של UNION ארוך במידה ניכרת מזה של MAKE-SET או FIND-SET. כפי שמראה איור 22.2(ii), אנו מבצעים UNION(x, y) על־ידי הוספת הרשימה של x בסוף הרשימה של y. (הוספה זו נעשית בזמן $O(1)$, הודות למצביע לסוף הרשימה של y.) נציג הקבוצה החדשה הוא האיבר שהיה במקור נציג הקבוצה שהכילה את y. לרוע המזל עלינו לעדכן את המצביע לנציג בכל עצם שהיה שייך במקורו לרשימה של x. לעדכון זה דרוש זמן התלוי לינארית באורך הרשימה של x.

למעשה, אין זה קשה למצוא סדרה של m פעולות אשר לביצועה יידרש זמן $\Theta(m^2)$. יהיו $n = \lceil m/2 \rceil + 1$ ו־$q = m - n + 1 = \lfloor m/2 \rfloor$, ונניח שנתונים לנו העצמים $x_1, x_2, \ldots, x_n$.

נבצע את סדרת $m = n + q - 1$ הפעולות המוצגת באיור 22.3. ביצוע n פעולות MAKE-SET צורך זמן $\Theta(n)$. מכיוון שהפעולה UNION ה־i מעדכנת i עצמים, המספר הכולל של עצמים המעודכנים על־ידי כל הפעולות UNION הוא:

$$\sum_{i=1}^{q-1} i = \Theta(q^2)$$

הזמן הכולל שצורכות פעולות אלה הוא אפוא $\Theta(n + q^2)$, שהוא $\Theta(m^2)$, שכן $n = \Theta(m)$ ו־$q = \Theta(m)$. לכן, בממוצע, לכל פעולה דרוש זמן $\Theta(m)$. כלומר, הזמן לשיעורין של פעולה הוא $\Theta(m)$.

הפעולה	מספר העצמים המעודכנים
MAKE-SET(x_1)	1
MAKE-SET(x_2)	1
⋮	⋮
MAKE-SET(x_n)	1
UNION(x_1, x_2)	1
UNION(x_2, x_3)	2
UNION(x_3, x_4)	3
⋮	⋮
UNION(x_{q-1}, x_q)	$q - 1$

איור 22.3 סדרה של m פעולות שצורכת זמן של $\Theta(m^2)$ כאשר משתמשים בייצוג קבוצות על־ידי רשימות מקושרות ובמימוש פשוט של UNION. בדוגמה זו, $n = \lceil m/2 \rceil + 1$ ו־$q = m - n + 1$.

היוריסטיקה של איחוד משוקלל

המימוש שהוצג לעיל של UNION דורש בממוצע זמן של $\Theta(m)$ לכל קריאה לשגרה, שכן ייתכן שאנו מוסיפים את הרשימה הארוכה יותר בסוף הרשימה הקצרה יותר, ועלינו לעדכן את המצביע לנציג עבור כל איבר ברשימה הארוכה יותר. במקום זאת נניח כי כל נציג מכיל גם את אורך הרשימה (שדה שקל לתחזק אותו) וכי אנו מוסיפים תמיד את הרשימה הקצרה יותר לרשימה הארוכה יותר, עם החלטה שרירותית במקרה של שוויון. בהיוריסטיקה פשוטה זו של **איחוד משוקלל** (weighted-union heuristic), פעולת UNION יחידה עדיין עשויה לצרוך זמן של $\Omega(m)$ אם כל אחת משתי הקבוצות מכילה $\Omega(m)$ איברים. אולם, כפי שמראה המשפט שלהלן, סדרה של m פעולות MAKE-SET, UNION ו־FIND-SET, אשר n מהן הן פעולות MAKE-SET, מתבצעת בזמן $O(m + n \lg n)$.

משפט 22.1

כאשר משתמשים בייצוג קבוצות זרות על־ידי רשימות מקושרות ובהיוריסטיקה של איחוד משוקלל, סדרה של m פעולות MAKE-SET, UNION ו־FIND-SET, אשר n מהן הן פעולות MAKE-SET, מתבצעת בזמן $O(m + n \lg n)$.

הוכחה תחילה נחשב, עבור כל עצם בקבוצה שגודלה n, חסם עליון על מספר הפעמים שעודכן המצביע מן העצם אל הנציג. נתבונן בעצם קבוע x. אנו יודעים שבכל פעם שעודכן המצביע מ־x לנציג, הרי שבהכרח x השתייך קודם לכן לקבוצה הקטנה יותר. לכן, בפעם הראשונה שעודכן

המצביע מ־x לנציג, הקבוצה שהתקבלה הכילה בהכרח 2 איברים לפחות. באותו אופן, בפעם הבאה שעודכן המצביע מ־x לנציג, הקבוצה שהתקבלה הכילה בהכרח 4 איברים לפחות. אם נמשיך באופן זה נראה שעבור כל $k \le n$, לאחר שעודכן המצביע מ־x לנציג $\lceil \lg k \rceil$ פעמים, הקבוצה שהתקבלה הכילה בהכרח k איברים לפחות. מכיוון שהקבוצה הגדולה ביותר מכילה n איברים לכל היותר, הרי שבכל עצם, המצביע לנציג עודכן $\lceil \lg n \rceil$ פעמים לכל היותר במהלך כל הפעולות UNION. הזמן הכולל שצורך עדכונם של n העצמים הוא אפוא $O(n \lg n)$.

הזמן לביצוע סדרת m הפעולות כולה נובע מכאן בנקל. כל פעולת MAKE-SET ו־FIND-SET מתבצעת בזמן $O(1)$, וישנן $O(m)$ פעולות כאלה. הזמן הכולל לביצוע סדרת הפעולות כולה הוא אפוא $O(m + n \lg n)$. ■

תרגילים

22.2-1

כתוב פסאודו־קוד עבור MAKE-SET, FIND-SET ו־UNION, תוך שימוש בייצוג על־ידי רשימות מקושרות ובהיוריסטיקה של איחוד משוקלל. הנח שכל עצם x מכיל את המאפיינים הבאים: $rep[x]$ — מצביע לנציג הקבוצה המכילה את x, $last[x]$ — מצביע לעצם האחרון ברשימה המקושרת המכילה את x, ו־$size[x]$ — גודל הקבוצה המכילה את x. בפסאודו־קוד שתכתוב אתה יכול להניח כי $last[x]$ ו־$size[x]$ נכונים רק אם x הוא נציג.

22.2-2

הצג את מבנה הנתונים המתקבל ואת התשובות שמחזירות הקריאות ל־FIND-SET בתכנית שלהלן. השתמש בייצוג על־ידי רשימות מקושרות ובהיוריסטיקה של איחוד משוקלל.

```
1  for i ← 1 to 16
2      do MAKE-SET(x_i)
3  for i ← 1 to 15 by 2
4      do UNION(x_i, x_{i+1})
5  for i ← 1 to 13 by 4
6      do UNION(x_i, x_{i+2})
7  UNION(x_1, x_5)
8  UNION(x_11, x_13)
9  UNION(x_1, x_10)
10 FIND-SET(x_2)
11 FIND-SET(x_9)
```

22.2-3

שנה את ההוכחה של משפט 22.1 כך שיתקבל חסם של $O(1)$ על הזמן לשיעורין עבור MAKE-SET ו־FIND-SET וחסם של $O(\lg n)$ עבור UNION, תוך שימוש בייצוג על־ידי רשימות מקושרות ובהיוריסטיקה של איחוד משוקלל.

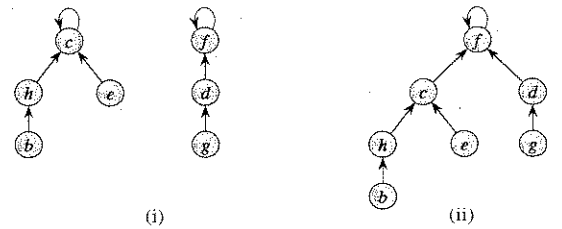
22.2-4

מצא חסם אסימפטוטי הדוק על זמן הריצה של סדרת הפעולות מאיור 22.3, בהנחה של שימוש בייצוג על־ידי רשימות מקושרות ובהיוריסטיקה של איחוד משוקלל.

22.3 יערות של קבוצות זרות

במימוש מהיר יותר של קבוצות זרות מייצגים קבוצות על־ידי עצים מושרשים, כאשר כל צומת מכיל איבר אחד של קבוצה וכל עץ מייצג קבוצה אחת. **ביער קבוצות זרות** (disjoint-set forest), המתואר באיור 22.4(i), כל איבר מצביע אך ורק על אביו. השורש של כל עץ מכיל את הנציג והוא האב של עצמו. כפי שנראה, האלגוריתמים הישירים המשתמשים בייצוג זה אינם מהירים מאלה המשתמשים בייצוג על־ידי רשימות מקושרות, אולם על־ידי שימוש בשתי היוריסטיקות — "איחוד על פי דרגה" ו"כיווץ מסלולים" — אנו יכולים לקבל את מבנה הנתונים לקבוצות זרות המהיר ביותר אסימפטוטית המוכר היום.

את שלוש הפעולות על קבוצות זרות מבצעים באופן הבא. הפעולה MAKE-SET פשוט יוצרת עץ ובו צומת אחד. את הפעולה FIND-SET מבצעים על־ידי מעקב אחר מצביעים לאבות עד שמוצאים את שורש העץ. הצמתים שאנו מבקרים בהם במסלול זה אל השורש מרכיבים את **מסלול הגילוי** (find path). הפעולה UNION, המתוארת באיור 22.4(ii), גורמת לכך שהשורש של עץ אחד יצביע על השורש של העץ האחר.



איור 22.4 יער קבוצות זרות. **(i)** שני עצים המייצגים את שתי הקבוצות מאיור 22.2. העץ השמאלי מייצג את הקבוצה $\{b, c, e, h\}$, שהנציג שלה הוא c, והעץ הימני מייצג את הקבוצה $\{d, f, g\}$, שהנציג שלה הוא f. **(ii)** העץ המתקבל מ־$\text{UNION}(e, g)$.

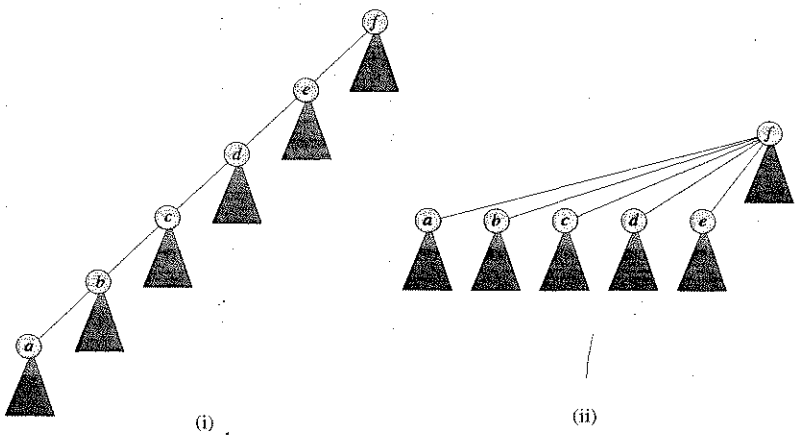
היוריסטיקות לשיפור זמן הריצה

עד כה לא שיפרנו את הביצועים בהשוואה למימוש באמצעות רשימות מקושרות. סדרה של $n-1$ פעולות UNION עלולה ליצור עץ שאינו אלא שרשרת לינארית של n צמתים. אולם על־ידי שימוש בשתי היוריסטיקות נוכל לקבל זמן ריצה שהוא כמעט לינארי במספר הפעולות הכולל m.

ההיוריסטיקה הראשונה, **איחוד על פי דרגה** (union by rank)[2], דומה להיוריסטיקת האיחוד המשוקלל שהפעלנו בייצוג על־ידי רשימות מקושרות. הרעיון הוא לדאוג לכך ששורש העץ המכיל פחות צמתים יצביע על שורש העץ המכיל יותר צמתים. במקום לנהל מעקב מפורש אחר גודל התת־עץ המושרש בכל צומת, ננקוט בגישה המקלה את הניתוח. עבור כל צומת נתחזק שדה **דרגה** (rank) שערכו הוא בקירוב הלוגריתם של גודל התת־עץ וזהו מהווה גם חסם עליון על גובהו של הצומת. באיחוד על פי דרגה מבצעים את הפעולה UNION כך שהשורש בעל הדרגה הקטנה יותר יצביע על השורש בעל הדרגה הגדולה יותר.

2. יש לשים לב לכך שבעברית אנו משתמשים במילה דרגה הן לתרגום המונח degree והן לתרגום המונח rank. בדרך כלל אין קושי להבין מן ההקשר לאיזה מונח מתכוונים.

ההיוריסטיקה השנייה, **כיווץ מסלולים** (path compression), גם היא פשוטה למדי ויעילה מאוד. כפי שאפשר לראות באיור 22.5, משתמשים בהיוריסטיקה זו בעת ביצוע פעולות FIND-SET כדי לגרום לכך שכל צומת על מסלול הגילוי יצביע ישירות על השורש. כיווץ מסלולים אינו משנה שום דרגה.



איור 22.5 כיווץ מסלולים במהלך הפעולה FIND-SET. חיצים ולולאות עצמיות בשורשים הושמטו. **(i)** עץ המייצג קבוצה לפני ביצוע $\text{FIND-SET}(a)$. המשולשים מייצגים תת־עצים המושרשים בצמתים המוצגים. כל צומת מכיל מצביע לאביו. **(ii)** אותה קבוצה לאחר ביצוע $\text{FIND-SET}(a)$. כל צומת על מסלול הגילוי מצביע עתה ישירות לשורש.

פסאודו־קוד עבור יערות של קבוצות זרות

כדי לממש יער קבוצות זרות כך שניתן יהיה להשתמש בהיוריסטיקת האיחוד על פי דרגה, יש לנהל מעקב אחר דרגותיהם של צמתים. עבור כל צומת x, אנו מתחזקים שדה בעל ערך שלם בשם $rank[x]$, שהוא חסם עליון על גובהו של x (מספר הקשתות במסלול הארוך ביותר מ־x לצאצא עלה). כאשר הפעולה MAKE-SET יוצרת קבוצה בת איבר אחד, הדרגה ההתחלתית של הצומת היחיד בעץ המייצג את הקבוצה היא 0. כל פעולת FIND-SET מותירה את כל הדרגות ללא שינוי. כאשר מבצעים UNION של שני עצים, הופכים את השורש בעל הדרגה הגבוהה יותר לאביו של השורש בעל הדרגה הנמוכה יותר. כאשר דרגות השורשים שוות, בוחרים באופן שרירותי את אחד השורשים כאב ומגדילים ב־1 את דרגתו.

הבה נתרגם שיטה זו לפסאודו־קוד. נסמן ב־$p[x]$ את אביו של צומת x. השגרה LINK, הנקראת על ידי UNION, מקבלת כקלט מצביעים לשני שורשים.

```
MAKE-SET(x)
1  p[x] ← x
2  rank[x] ← 0
```

```
UNION(x, y)
1  LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
1  if rank[x] > rank[y]
2     then p[y] ← x
3     else p[x] ← y
4          if rank[x] = rank[y]
5             then rank[y] ← rank[y] + 1
```

השגרה FIND-SET עם כיווץ מסלולים פשוטה למדי.

```
FIND-SET(x)
1  if x ≠ p[x]
2     then p[x] ← FIND-SET(p[x])
3  return p[x]
```

השגרה FIND-SET פועלת ב**שיטה של שני מעברים**: היא מבצעת מעבר אחד במעלה מסלול הגילוי כדי למצוא את השורש, ומעבר שני בחזרה במורד מסלול הגילוי כדי לעדכן כל צומת במסלול כך שיצביע ישירות לשורש. כל קריאה ל-$\text{FIND-SET}(x)$ מחזירה את $p[x]$ בשורה 3. אם x הוא השורש, שורה 2 אינה מבוצעת והשגרה מחזירה את $x = p[x]$. זהו גם המקרה שבו נעצרת שרשרת הקריאות הרקורסיביות והשגרה מתחילה לחזור כלפי מעלה. אחרת, שורה 2 מבוצעת והקריאה הרקורסיבית עם הפרמטר $p[x]$ מחזירה מצביע לשורש. שורה 2 מעדכנת את הצומת x כך שיצביע ישירות לשורש, ושורה 3 מחזירה מצביע זה.

השפעת ההיוריסטיקות על זמן הריצה

שתי ההיוריסטיקות, איחוד על פי דרגה וכיווץ מסלולים, משפרות כל אחת בנפרד את זמן הריצה של הפעולות על יערות של קבוצות זרות, והשיפור גדול אף יותר כאשר מפעילים את שתיהן ביחד. איחוד על פי דרגה לבדו מניב אותו זמן ריצה כמו זה שהושג בשימוש בהיוריסטיקת האיחוד המשוקלל עבור ייצוג על-ידי רשימות: זמן הריצה של המימוש המתקבל הוא $O(m \lg n)$ (ראה תרגיל 22.4-3). זהו חסם הדוק (ראה תרגיל 22.3-3). אמנם לא נוכיח זאת כאן, אולם אם מבצעים n פעולות MAKE-SET (ולכן לכל היותר $n-1$ פעולות UNION) ו-f פעולות FIND-SET, ומשתמשים בהיוריסטיקת כיווץ המסלולים לבדה, זמן הריצה במקרה הגרוע הוא $\Theta(f \log_{(1+f/n)} n)$ אם $f \geq n$, ו-$\Theta(n + f \lg n)$ אם $f < n$.

כאשר מפעילים איחוד על פי דרגה וגם כיווץ מסלולים, זמן הריצה במקרה הגרוע הוא $O(m\,\alpha(m, n))$, כאשר $\alpha(m, n)$ היא הפונקציה ההופכית לפונקציית אקרמן, אשר גדלה לאט *מאוד*. פונקציה זו תוגדר בסעיף 22.4. בכל יישום של מבנה נתונים לקבוצות זרות שניתן להעלות על הדעת, $\alpha(m, n) \leq 4$; לכן נוכל לראות את זמן הריצה כלינארי ב-m בכל מצב מעשי. בסעיף 22.4 נוכיח את החסם החלש במעט $O(m \lg^* n)$.

תרגילים

22.3-1

פתור את תרגיל 22.2-2 תוך שימוש ביער של קבוצות זרות עם איחוד על פי דרגה וכיווץ מסלולים.

22.3-2

כתוב גרסה לא רקורסיבית של FIND-SET עם כיווץ מסלולים.

22.3-3

הצג סדרה של m פעולות MAKE-SET, UNION ו-FIND-SET, אשר n מהן הן פעולות MAKE-SET, המתבצעת בזמן $\Omega(m \lg n)$ כאשר משתמשים באיחוד על פי דרגה בלבד.

22.3-4 *

הראה שכל סדרה שהיא של m פעולות MAKE-SET, FIND-SET ו-LINK, אשר בה כל הפעולות LINK מופיעות לפני כל הפעולות FIND-SET, מתבצעת בזמן $O(m)$ בלבד אם משתמשים גם בכיווץ מסלולים וגם באיחוד על פי דרגה. מה קורה במצב כזה אם משתמשים בכיווץ מסלולים בלבד?

22.4 * ניתוח של איחוד על פי דרגה יחד עם כיווץ מסלולים

כפי שהזכרנו בסעיף 22.3, זמן הריצה של ההיוריסטיקה המשולבת איחוד על פי דרגה וכיווץ מסלולים הוא $O(m\,\alpha(m, n))$ עבור m פעולות על קבוצות זרות המכילות ביחד n איברים. בסעיף זה נבחן את הפונקציה α כדי להיווכח בדיוק עד כמה לאט היא גדלה. לאחר מכן, במקום להביא את ההוכחה המורכבת מאוד של זמן הריצה ($O(m\,\alpha(m, n))$), נציע הוכחה פשוטה יותר עבור חסם עליון חלש במעט על זמן הריצה: $O(m \lg^* n)$.

פונקציית אקרמן והפונקציה ההופכית לה

כדי להבין את פונקציית אקרמן ואת הפונקציה ההופכית לה α, ניעזר בסימון עבור העלאה חוזרת בחזקה. עבור שלם $i \geq 0$, הביטוי:

$$2^{2^{\cdot^{\cdot^{2}}}\}i}$$

מייצג את הפונקציה $g(i)$, המוגדרת באופן רקורסיבי על-ידי:

$$g(i) = \begin{cases} 2^1 & \text{אם } i = 0 \\ 2^2 & \text{אם } i = 1 \\ 2^{g(i-1)} & \text{אם } i > 1 \end{cases}$$

אינטואיטיבית, הפרמטר i מציין את "גובה המגדל של מופעי ה-2" המרכיב את מעריך החזקה. לדוגמה:

$$2^{2^{\cdot^{\cdot^{2}}}\}4} = 2^{2^{2^2}} = 2^{65536}$$

ניזכר בהגדרת הפונקציה $\lg^*$ (סעיף 2.2) במונחי הפונקציות $\lg^{(i)}$, המוגדרות עבור $i \geq 0$ שלם כך:

$$\lg^{(i)} n = \begin{cases} n & \text{אם } i = 0 \\ \lg(\lg^{(i-1)} n) & \text{אם } i > 0 \text{ וגם } \lg^{(i-1)} n > 0 \\ \text{לא-מוגדר} & \text{אם } i > 0 \text{ וגם } \lg^{(i-1)} n \leq 0 \text{ או } \lg^{(i-1)} n \text{ אינו מוגדר} \end{cases}$$

$$\lg^* n = \min\{i \geq 0 : \lg^{(i)} n \leq 1\}$$

הפונקציה $\lg^* n$ היא בעצם הפונקציה ההופכית לפונקציה של העלאה חוזרת בחזקה:

$$\lg^* 2^{2^{\cdot^{\cdot^{2}}}\}n} = n+1$$

עתה אנו מוכנים להצגתה של פונקציית אקרמן, המוגדרת עבור שלמים $i, j \geq 1$ על-ידי:

$$A(1, j) = 2^j \qquad \text{עבור } j \geq 1$$

$$A(i, 1) = A(i-1, 2) \qquad \text{עבור } i \geq 2$$

$$A(i, j) = A(i-1, A(i, j-1)) \qquad \text{עבור } i, j \geq 2$$

באיור 22.6 מוצגים ערכי הפונקציה עבור ערכים קטנים של i ו־j.

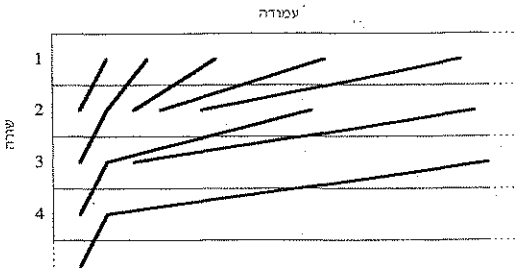
	$j=1$	$j=2$	$j=3$	$j=4$
$i=1$	2^1	2^2	2^3	2^4
$i=2$	2^2	2^{2^2}	$2^{2^{2^2}}$	$2^{2^{2^{2^2}}}$
$i=3$	2^{2^2}	$2^{2^{\cdot^{\cdot^{2}}}\}16}$	$2^{2^{\cdot^{\cdot^{2}}}\}2^{2^{\cdot^{\cdot^{2}}}\}16}}$	$2^{2^{\cdot^{\cdot^{2}}}\}2^{2^{\cdot^{\cdot^{2}}}\}2^{2^{\cdot^{\cdot^{2}}}\}16}}}$

איור 22.6 ערכי $A(i, j)$ עבור ערכים קטנים של i ו־j.

איור 22.7 מראה באופן סכמטי מדוע פונקציית אקרמן גדלה באופן דרמטי כל כך. הערכים בשורה הראשונה, שהם אקספוננציאליים במספר העמודה j, כבר גדלים במהירות. השורה השנייה מורכבת מהתת־קבוצה המרווחת מאוד של העמודות $2, 2^2, 2^{2^2}, 2^{2^{2^2}}, \ldots$ מן השורה הראשונה. הקווים המופיעים באיור בין שורות סמוכות מציינים עמודות בשורה שמספרה גמוד יותר השייכות לתת־קבוצה שממנה מורכבת השורה שמספרה גבוה יותר. השורה השלישית מורכבת מהתת־קבוצה המרווחת אף יותר של העמודות:

$$2, 2^{2^2}, 2^{2^{\cdot^{\cdot^{2}}}\}16}, 2^{2^{\cdot^{\cdot^{2}}}\}2^{2^{\cdot^{\cdot^{2}}}\}16}}, \ldots$$

מן השורה השנייה, שהיא תת־קבוצה דלילה עוד יותר של עמודות מן השורה הראשונה. באופן כללי, המרווחים בין עמודות השורה ה־$i-1$ המופיעות בשורה i גדלים באופן דרמטי הן עם מספר העמודה והן עם מספר השורה. נשים לב כי $A(2, j) = 2^{2^{\cdot^{\cdot^{2}}}\}j}$ עבור כל השלמים $j \geq 1$. לכן, עבור $i > 2$, הפונקציה $A(i, j)$ גדלה מהר יותר מאשר $2^{2^{\cdot^{\cdot^{2}}}\}j}$.



איור 22.7 הגידול הדרמטי של פונקציית אקרמן. הקווים בין השורות $i-1$ ו־i מציינים ערכים משורה $i-1$ המופיעים בשורה i. עקב הגידול הדרמטי של הפונקציה, המרווחים האופקיים אינם באותו קנה מידה. המרווחים האופקיים בין ערכים משורה $i-1$ המופיעים בשורה i גדלים מאוד עם מספר העמודה ועם מספר השורה. אם עוקבים אחר הערכים משורה i עד להופעותיהם המקוריות בשורה הראשונה, ניתן לראות באופן ברור עוד יותר את הגידול הדרמטי.

את הפונקציה ההופכית לפונקציית אקרמן אנו מגדירים על־ידי[3]:

$$\alpha(m, n) = \min\{i \geq 1 : A(i, \lfloor m/n \rfloor) > \lg n\}$$

אם נתבונן בערך קבוע של n, אזי הפונקציה $\alpha(m, n)$ יורדת מונוטונית כאשר m גדל. כדי לראות תכונה זו, נזכור כי $\lfloor m/n \rfloor$ עולה מונוטונית כאשר m גדל; לכן, מאחר ש־n קבוע, הערך הקטן ביותר של i, הדרוש כדי ש־$A(i, \lfloor m/n \rfloor)$ יהיה גדול מ־$\lg n$, יורד מונוטונית. תכונה זו מקבילה לאינטואיציה שלנו לגבי יערות של קבוצות זרות עם כיווץ מסלולים: עבור מספר נתון n של איברים שונים, ככל שגדל מספר הפעולות m, היינו מצפים שיקטן האורך הממוצע של מסלול הגילוי בשל כיווץ המסלולים. אם אנו מבצעים m פעולות בזמן $O(m\,\alpha(m, n))$, אזי הזמן הממוצע לפעולה הוא $O(\alpha(m, n))$, וזמן זה יורד מונוטונית כאשר m גדל.

כדי לתמוך בטענתנו הקודמת כי $\alpha(m, n) \leq 4$ לכל מטרה מעשית, נשים לב תחילה, כי ערכו של הגודל $\lfloor m/n \rfloor$ הוא לפחות 1, שכן $m \geq n$. מאחר שפונקציית אקרמן עולה ממש עם כל אחד מהארגומנטים שלה, הרי ש־$A(i, \lfloor m/n \rfloor) \geq A(i, 1)$ גורר $\lfloor m/n \rfloor \geq 1$ עבור כל $i \geq 1$. בפרט, $A(4, \lfloor m/n \rfloor) \geq A(4, 1)$. אולם $A(4, 1)$ מקיים גם:

$$A(4, 1) = A(3, 2) = 2^{2^{\cdot^{\cdot^{2}}}\}16}$$

וערך זה גדול בהרבה מהמספר המשוער של אטומים ביקום הנצפה (שהוא 10^{80} בערך). רק עבור ערכים של n שהם בלתי מעשיים בגודלם מתקיים $A(4, 1) \leq \lg n$, ולכן $\alpha(m, n) \leq 4$ לכל מטרה מעשית. נשים לב שהחסם $O(m \lg^* n)$ חלש אך במעט מהחסם $O(m\,\alpha(m, n))$; $\lg^* 65536 = 4$ ו־$\lg^* 2^{65536} = 5$, כך שלכל מטרה מעשית, $\lg^* n \leq 5$.

3. אף כי פונקציה זו אינה הפונקציה ההופכית לפונקציית אקרמן במשמעות המתמטית הטהורה, היא מבטאת את מהות הפונקציה ההופכית באופן גידולה, שהוא איטי כשם שגידולה של פונקציית אקרמן מהיר. הסיבה לשימוש בסף המסתורי $\lg n$ מתחוורת בהוכחה של החסם $O(m\,\alpha(m, n))$ על זמן הריצה, הוכחה החורגת מתחומי ספר זה.

תכונות של דרגות

ביתרונו של סעיף זה נוכיח חסם של $O(m \lg^* n)$ על זמן הריצה של הפעולות על קבוצות זרות עם איחוד על פי דרגה וכיווץ מסלולים. כדי להוכיח חסם זה, נוכיח תחילה כמה תכונות פשוטות של דרגות.

למה 22.2

לכל צומת x מתקיים $rank[x] \le rank[p[x]]$, כאשר אי־שוויון ממש מתקיים אם $x \ne p[x]$. הערך של $rank[x]$ הוא בתחילה 0 והוא גדל עם הזמן עד אשר $x \ne p[x]$; מכאן ואילך $rank[x]$ אינו משתנה. הערך של $rank[p[x]]$ הוא פונקציה מונוטונית עולה של הזמן.

הוכחה ההוכחה נעשית באינדוקציה פשוטה על מספר הפעולות, תוך שימוש במימושים של MAKE-SET, UNION ו־FIND-SET שהובאו בסעיף 22.3. אנו מותירים את ההוכחה לקורא בתרגיל 22.4-1.

■

נגדיר את $size(x)$ כמספר הצמתים שמכיל העץ המושרש ב־x, כולל x עצמו.

למה 22.3

לכל שורש x של עץ מתקיים $size(x) \ge 2^{rank[x]}$.

הוכחה ההוכחה נעשית באינדוקציה על מספר הפעולות LINK. נזכור שפעולות FIND-SET אינן משנות את דרגתו של שורש ולא את גודלו של העץ שלו.

בסיס: הלמה מתקיימת לפני פעולת LINK הראשונה מכיוון שערכן ההתחלתי של הדרגות הוא 0, וכל עץ מכיל לפחות צומת אחד.

צעד האינדוקציה: נניח שהלמה מתקיימת לפני ביצוע הפעולה $LINK(x, y)$. נסמן ב־$rank$ את הדרגה ממש לפני ביצוע הפעולה LINK וב־$rank'$ את הדרגה מיד לאחר ביצועה. נגדיר את $size$ ו־$size'$ באופן דומה.

אם $rank[x] \ne rank[y]$, נניח בלי הגבלת הכלליות כי $rank[x] < rank[y]$. הצומת y הוא שורש העץ שנוצר על־ידי הפעולה LINK, וכמו כן:

$$\begin{aligned} size'(y) &= size(x) + size(y) \\ &\ge 2^{rank[x]} + 2^{rank[y]} \\ &\ge 2^{rank[y]} \\ &= 2^{rank'[y]} \end{aligned}$$

שום דרגות וגדלים של צמתים אחרים לבד מ־y אינם משתנים.

אם $rank[x] = rank[y]$, y הוא שוב השורש של העץ החדש, וכמו כן:

$$\begin{aligned} size'(y) &= size(x) + size(y) \\ &\ge 2^{rank[x]} + 2^{rank[y]} \\ &= 2^{rank[y]+1} \\ &= 2^{rank'[y]} \end{aligned}$$

■

למה 22.4

לכל שלם $r \ge 0$, קיימים לכל היותר $n/2^r$ צמתים בעלי דרגה r.

הוכחה נתבונן בערך קבוע מסוים של r. נניח שכאשר אנו נותנים לצומת x דרגה r (בשורה 2 של MAKE-SET או בשורה 5 של LINK), אנו מצמידים תווית x לכל צומת בעץ המושרש ב־x. לפי למה 22.3, בכל פעם מוצמדות תוויות ל־2^r צמתים לפחות. נניח ששורש העץ המכיל את x משתנה. למה 22.2 מבטיחה לנו שדרגתו של השורש החדש (ולמעשה של כל אב קדמון ממש של x) היא לפחות $r+1$. מאחר שאנו מצמידים תוויות אך ורק כאשר שורש מקבל דרגה r, הרי שמעתה ואילך לעולם לא תוצמד תווית לאף צומת בעץ החדש. לכל צומת מוצמדת אפוא תווית פעם אחת לכל היותר, כאשר השורש שלו מקבל בפעם הראשונה דרגה r. מכיוון שישנם n צמתים, ישנם לכל היותר n צמתים בעלי תוויות, כשלפחות 2^r תוויות מוצמדות עבור כל צומת שדרגתו r. אילו היו יותר מאשר $n/2^r$ צמתים בעלי דרגה r, אזי היו יותר מאשר $2^r \cdot (n/2^r) = n$ צמתים שהיו מוצמדות אליהם תוויות על־ידי צומת בעל דרגה r, וזוהי סתירה. לכן יש לכל היותר $n/2^r$ צמתים שמקבלים אי פעם דרגה r.

■

מסקנה 22.5

הדרגה של כל צומת היא לכל היותר $\lfloor \lg n \rfloor$.

הוכחה אם $r > \lg n$, אזי מספר הצמתים שדרגתם r הוא לכל היותר $n/2^r < 1$, בסתירה לכך שדרגות הן מספרים טבעיים. מכאן שדרגתו של צומת אינה עולה על $\lfloor \lg n \rfloor$. ■

הוכחת החסם על זמן הריצה

את החסם $O(m \lg^* n)$ על זמן הריצה נוכיח בשיטת הצבירה של ניתוח לשיעורין (ראה סעיף 18.1). בעת ביצוע הניתוח לשיעורין נוח להניח כי אנו קוראים לפעולה LINK ולא לפעולה UNION. כלומר, מאחר שהפרמטרים של השגרה LINK הם מצביעים לשני שורשים, אנו מניחים כי מתבצעות הפעולות FIND-SET המתאימות אם יש צורך בכך. הלמה שלהלן מראה שגם אם מביאים בחשבון את הפעולות FIND-SET הנוספות, זמן הריצה האסימפטוטי אינו משתנה.

למה 22.6

נניח שאנו ממירים סדרה S' של m' פעולות MAKE-SET, UNION ו־FIND-SET בסדרה S של m פעולות MAKE-SET, LINK ו־FIND-SET על־ידי החלפת כל פעולת UNION בשתי פעולות FIND-SET שאחריהן פעולת LINK. אזי, אם זמן הריצה של הסדרה S הוא $O(m \lg^* n)$, זמן הריצה של הסדרה S' הוא $O(m' \lg^* n)$.

הוכחה מאחר שכל פעולת UNION בסדרה S' מוחלפת בשלוש פעולות בסדרה S, הרי שמתקיים $m' \le m \le 3m'$. מאחר ש־$m = O(m')$, חסם של $O(m \lg^* n)$ על זמן הריצה של הסדרה החדשה S גורר חסם של $O(m' \lg^* n)$ על זמן הריצה של הסדרה המקורית S'. ■

בשאריתו של הסעיף הזה נניח כי הסדרה ההתחלתית של m' פעולות MAKE-SET, UNION ו־FIND-SET הומרה בסדרה של m פעולות MAKE-SET, LINK ו־FIND-SET. עתה נוכיח חסם של $O(m \lg^* n)$ על זמן הריצה של הסדרה שהתקבלה כתוצאה מהההמרה, ונסתמך על למה 22.6 כדי להוכיח חסם של $O(m' \lg^* n)$ על זמן הריצה של הסדרה המקורית של m' הפעולות.

משפט 22.7

סדרה של m פעולות MAKE-SET, LINK ו־FIND-SET, אשר n מהן הן פעולות MAKE-SET, ניתנת לביצוע על יער קבוצות זרות עם איחוד על פי דרגה וכיווץ מסלולים, בזמן $O(m \lg^* n)$ במקרה הגרוע.

הוכחה אנו קובעים **חיובים** המתאימים לעלויות בפועל של כל אחת מן הפעולות על קבוצות, ומסכמים את המספר הכולל של יחידות החיוב שנקבעו לאחר שבוצעה סדרת הפעולות כולה. סכום זה נותן לנו את העלות בפועל של כל הפעולות על קבוצות.

החיובים לפעולות LINK ו־MAKE-SET הם פשוטים: יחידת חיוב אחת לכל פעולה. מאחר שהזמן בפועל לביצוע כל אחת מפעולות אלה הוא $O(1)$, החיובים שנקבעו שווים לעלויות בפועל של הפעולות.

לפני שנדון בחיוב הפעולות FIND-SET, נחלק דרגות של צמתים **לגושים** על־ידי כך שנשייך כל דרגה r לגוש שמספרו $\lg^* r$, עבור $r = 0, 1, \ldots, \lfloor \lg n \rfloor$. (נזכור כי $\lfloor \lg n \rfloor$ היא הדרגה המקסימלית.) הגוש בעל המספר הגבוה ביותר הוא אפוא גוש $\lg^*(\lg n) = \lg^* n - 1$. לצורך נוחות הסימון, נגדיר עבור שלמים $j \geq -1$:

$$B(j) = \begin{cases} -1 & \text{אם } j = -1 \\ 1 & \text{אם } j = 0 \\ 2 & \text{אם } j = 1 \\ 2^{B(j-1)} & \text{אם } j \geq 2 \end{cases}$$

$$= 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \quad (j-1 \text{ פעמים})$$

אזי, עבור $j = 0, 1, \ldots, \lg^* n - 1$, הגוש ה־$j$ מורכב מקבוצת הדרגות:

$$\{B(j-1)+1, B(j-1)+2, \ldots, B(j)\}$$

עבור הפעולה FIND-SET נשתמש בשני סוגים של חיובים: **חיובי־גוש וחיובי־מסלול**. נניח כי FIND-SET מתחילה בצומת x_0 וכי מסלול הגילוי מורכב מן הצמתים $x_l, \ldots, x_1, x_0$, כאשר עבור $i = 1, 2, \ldots, l$, הצומת x_i הוא $p[x_{i-1}]$ והצומת x_l (שהוא שורש) הוא $p[x_l]$. עבור $j = 0, 1, \ldots, \lg^* n - 1$, אנו קובעים יחידה אחת של חיובי־גוש לצומת *האחרון* במסלול שדרגתו שייכת לגוש j. (נשים לב כי מלמה 22.2 נובע שבכל מסלול גילוי, הצמתים אשר דרגותיהם שייכות לאותו גוש הם צמתים עוקבים.) אנו גם קובעים יחידה אחת של חיובי־גוש לבנו של השורש, דהיינו, ל־x_{l-1}. מאחר שהדרגות עולות ממש לאורך כל מסלול גילוי, הרי שבאופן שקול ניתן לומר שקובעים יחידה אחת של חיובי־גוש לכל צומת x_i המקיים $p[x_i] = x_l$ (x_i הוא השורש או בנו) או $\lg^* rank[x_i] < \lg^* rank[x_{i+1}]$ (הדרגה של x_i והדרגה של אביו שייכות לגושים שונים). לכל צומת על מסלול הגילוי שלא קבענו לו חיובי־גוש, אנו קובעים יחידה אחת של חיובי־מסלול.

ברגע שקבענו לצומת שאינו השורש או בנו חיובי־גוש, שוב לא נקבע לו לעולם חיובי־מסלול. כדי להיווכח בכך, נשים לב שבכל פעם שמתרחש כיווץ מסלול, דרגתו של צומת x_i שעבורו $p[x_i] \neq x_l$ אינה משתנה, אולם דרגתו של אביו החדש של x_i גדולה ממש מדרגתו של אביו הקודם. ההפרש בין דרגתו של x_i לדרגת אביו הוא פונקציה מונוטונית עולה בזמן. לכן, ההפרש בין $\lg^* rank[p[x_i]]$ ל־$\lg^* rank[x_i]$ גם הוא פונקציה מונוטונית עולה בזמן. מרגע שדרגותיהם של x_i ואביו שייכות לגושים שונים, דרגותיהם לעולם ישתייכו לגושים שונים, ולכן לעולם שוב לא נקבע ל־x_i חיובי־מסלול.

מכיוון שקבענו יחידת חיוב אחת לכל צומת שביקרנו בו בכל אחת מהפעולות FIND-SET, המספר הכולל של יחידות החיוב שנקבעו שווה למספר הכולל של הצמתים שביקרנו בהם במהלך כל הפעולות FIND-SET שבוצעו; מספר כולל זה מייצג את העלות בפועל של כל הפעולות FIND-SET. ברצוננו להראות שמספר כולל זה הוא $O(m \lg^* n)$.

את חיובי הגוש קל לחסום. עבור כל מספר גוש במסלול גילוי נתון נקבעת לכל היותר יחידה אחת של חיובי־גוש, ועוד יחידה אחת של חיובי־גוש עבור בנו של השורש. מאחר שמספרי הגושים נעים

מ־0 עד $\lg^* n - 1$, הרי שבמהלך ביצועה של פעולת FIND-SET אחת נקבעות לכל היותר $\lg^* n + 1$ יחידות של חיובי־גוש. לכן, במהלך ביצוען של כל הפעולות FIND-SET נקבעות לכל היותר $m(\lg^* n + 1)$ יחידות של חיובי־גוש.

חסימת חיובי־המסלול אינה פשוטה כל כך. תחילה, נשים לב שאם לצומת x_i נקבעה יחידה של חיובי־גוש, פירוש הדבר שלפני כיווץ המסלול התקיים $p[x_i] \neq x_l$, ולכן במהלך כיווץ המסלול, x_i יקבל אב חדש. יתר על כן, כפי שכבר ראינו, אביו החדש של x_i הוא בעל דרגה גבוהה יותר מזו של אביו הקודם. נניח שדרגתו של הצומת x_i שייכת לגוש j. כמה פעמים יכול x_i לקבל אב חדש, ולכן גם חיובי־מסלול, לפני שיקבל אב שדרגתו שייכת לגוש שונה (ומכאן ואילך שוב לא ייקבעו לו לעולם חיובי־מסלול)? מספר פעמים זה הוא מקסימלי כאשר x_i הוא בעל הדרגה הנמוכה ביותר בגוש שלו, דהיינו, $B(j-1)+1$, ואבותיו מקבלים בזו אחר זו את סדרת הדרגות: $B(j-1)+2, B(j-1)+3, \ldots, B(j)$. סדרה כזאת מונה $B(j) - B(j-1) - 1$ דרגות, ומכאן שכל עוד דרגתו של צומת שייכת לגוש j, מספר היחידות של חיובי־המסלול שיכולות להיקבע לו הוא לכל היותר $B(j) - B(j-1) - 1$.

הצעד הבא שלנו בחסימת חיובי־המסלול הוא חסימת מספר הצמתים אשר דרגותיהם שייכות לגוש j, עבור $j \geq 0$ שלם. (נזכור שעל פי למה 22.2, דרגתו של צומת נשארת קבועה מרגע שהוא הופך לבן של צומת אחר.) נסמן ב־$N(j)$ את מספר הצמתים אשר דרגותיהם שייכות לגוש j. אזי, לפי למה 22.4, נקבל:

$$N(j) \leq \sum_{r=B(j-1)+1}^{B(j)} \frac{n}{2^r}$$

עבור $j = 0$, ערכו של סכום זה הוא:

$$\begin{aligned} N(0) &= n/2^0 + n/2^1 \\ &= 3n/2 \\ &= 3n/2B(0) \end{aligned}$$

עבור $j \geq 1$, נקבל:

$$\begin{aligned} N(j) &\leq \frac{n}{2^{B(j-1)+1}} \sum_{r=0}^{B(j)-(B(j-1)+1)} \frac{1}{2^r} \\ &< \frac{n}{2^{B(j-1)+1}} \sum_{r=0}^{\infty} \frac{1}{2^r} \\ &= \frac{n}{2^{B(j-1)}} \\ &= \frac{n}{B(j)} \end{aligned}$$

ולכן, עבור כל השלמים $j \geq 0$ מתקיים $N(j) \leq 3n/2B(j)$.

נסיים את חסימת חיובי־המסלול על־ידי כך שנסכם, על כל הגושים, את מכפלת המספר המקסימלי של צמתים שדרגותיהם שייכות לגוש במספר המקסימלי של יחידות חיובי־המסלול הנקבעות לצומת השייך לגוש זה. נסמן ב־$P(n)$ את המספר הכולל של יחידות חיובי־המסלול, ונקבל:

$$P(n) \le \sum_{j=0}^{\lg^* n - 1} \frac{n}{B(j)} \left(B(j) - B(j-1) - 1\right)$$

$$\le \sum_{j=0}^{\lg^* n - 1} \frac{n}{B(j)} \cdot B(j)$$

$$= n \lg^* n$$

אם כן, המספר הכולל של יחידות החיוב הנקבעות במהלך ביצוע הפעולות FIND-SET הוא $O(m(\lg^* n + 1) + n \lg^* n)$, שהוא $O(m \lg^* n)$, שכן $m \ge n$. מכיוון שמספר הפעולות MAKE-SET ו־LINK הוא $O(n)$, ולכל אחת מהן נקבעת יחידת חיוב אחת, הזמן הכולל לביצוע סדרת m הפעולות הוא $O(m \lg^* n)$. ■

מסקנה 22.8

סדרה של m פעולות MAKE-SET, UNION ו־FIND-SET, אשר n מהן הן פעולות MAKE-SET, ניתנת לביצוע על יער קבוצות זרות עם איחוד על פי דרגה וכיווץ מסלולים, בזמן $O(m \lg^* n)$ במקרה הגרוע.

הוכחה ההוכחה נובעת ישירות ממשפט 22.7 ומלמה 22.6. ■

תרגילים

22.4-1

הוכח את למה 22.2.

22.4-2

עבור כל צומת x, כמה סיביות דרושות לאחסון $size(x)$? וכמה לאחסון $rank[x]$?

22.4-3

בעזרת למה 22.2 ומסקנה 22.5, מצא הוכחה פשוטה לכך שפעולות על יער קבוצות זרות, עם איחוד על פי דרגה אך ללא כיווץ מסלולים, רצות בזמן $O(m \lg n)$.

22.4-4 *

נניח שאנו משנים את הכלל לקביעת החיובים, כך שאנו קובעים יחידה אחת של חיובי־גוש לצומת האחרון במסלול הגילוי שדרגתו שייכת לגוש j, עבור $j = 0, 1, \ldots, \lg^* n - 1$. אחרת, אנו קובעים לצומת יחידה אחת של חיובי־מסלול. כך, אם צומת הוא בנו של השורש ואינו הצומת האחרון בגוש, תיקבע לו יחידת חיובי־מסלול ולא יחידת חיובי־גוש. הראה כי לצומת נתון עשויות להיקבע $\Omega(m)$ יחידות של חיובי־מסלול בעת שדרגתו שייכת לגוש נתון j.

בעיות

22-1 מינימום בלתי־מקוון

בבעיית המינימום הבלתי־מקוונת (off-line minimum problem), אנו מתבקשים לתחזק קבוצה דינמית T שאיבריה לקוחים מן התחום $\{1, 2, \ldots, n\}$ והפעולות עליה הן INSERT ו־EXTRACT-MIN. נתונה לנו סדרה S של n קריאות ל־INSERT ו־m קריאות ל־EXTRACT-MIN, כאשר כל אחד מהמפתחות $\{1, 2, \ldots, n\}$ מוכנס לקבוצה בדיוק פעם אחת. ברצוננו לקבוע איזה מפתח מחזירה כל אחת מן הקריאות ל־EXTRACT-MIN. באופן מפורט, ברצוננו למלא מערך $extracted[1..m]$ שבו, עבור $i = 1, 2, \ldots, m$, $extracted[i]$ הוא המפתח שמחזירה הקריאה ה־i ל־EXTRACT-MIN. הבעיה היא "בלתי־מקוונת" במובן זה שמותר לנו להפעיל את סדרת S הפעולות כולה לפני שאנו קובעים אילו מפתחות החזירו הקריאות ל־EXTRACT-MIN.

א. במופע הבא של בעיית המינימום הבלתי־מקוונת, כל פעולת INSERT מיוצגת על־ידי מספר וכל פעולת EXTRACT-MIN מיוצגת על־ידי האות E:

4, 8, E, 3, E, 9, 2, 6, E, E, E, 1, 7, E, 5

מלא את המערך *extracted* בערכים המתאימים.

כדי לפתח אלגוריתם לפתרון בעיה זו, נפרק את הסדרה S לתת־סדרות הומוגניות, כלומר, נייצג את S על־ידי:

$$I_1, E, I_2, E, I_3, \ldots, I_m, E, I_{m+1}$$

כאשר כל E מייצג קריאה יחידה ל־EXTRACT-MIN וכל I_j מייצג סדרה (אולי ריקה) של קריאות ל־INSERT. לכל תת־סדרה I_j, אנו מקבצים תחילה את המפתחות שהוכנסו על־ידי הפעולות שב־I_j לקבוצה K_j, שהיא ריקה אם I_j ריקה, ואז אנו מבצעים את השגרה הבאה:

```
OFF-LINE-MINIMUM(m, n)
1  for i ← 1 to n
2      do determine j such that i ∈ K_j
3         if j ≠ m + 1
4           then extracted[j] ← i
5                let l be the smallest value greater than j
                     for which set K_l exists
6                K_l ← K_j ∪ K_l, destroying K_j
7  return extracted
```

ב. הראה שהמערך *extracted* שמחזירה השגרה OFF-LINE-MINIMUM הוא המערך הנכון.

ג. תאר כיצד ניתן להשתמש במבנה נתונים לקבוצות זרות כדי לממש ביעילות את OFF-LINE-MINIMUM. מצא חסם הדוק על זמן הריצה של המימוש שלך במקרה הגרוע.

22-2 קביעת עומק

בבעיה של קביעת עומק (depth-determination problem), אנו מתחזקים יער $\mathcal{F} = \{T_i\}$ של עצים מושרשים עם שלוש פעולות:

MAKE-TREE(v) — יוצרת עץ המכיל צומת יחיד v.

FIND-DEPTH(v) — מחזירה את עומקו של צומת v בעץ שבו הוא נמצא.

GRAFT(r, v) — הופכת צומת r, שההנחה היא כי הוא שורש של עץ, לבנו של צומת v, שההנחה היא כי הוא שייך לעץ שונה מזה של r; הצומת v עצמו עשוי להיות שורש או צומת שאינו שורש.

א. נניח שאנו משתמשים בייצוג עצים הדומה לייצוגם ביער קבוצות זרות: $p[v]$ הוא אביו של צומת v, פרט לכך ש־$p[v] = v$ אם v הוא שורש. הראה שאם נממש את GRAFT(r, v) על־ידי ההצבה $p[r] \leftarrow v$ ואת FIND-DEPTH(v) על־ידי מעקב אחר מסלול הגילוי במעלה העץ עד לשורש ומניית כל הצמתים שאנו נתקלים בהם פרט ל־v, אזי זמן הריצה של סדרה של m פעולות MAKE-TREE, FIND-DEPTH ו־GRAFT במקרה הגרוע הוא $O(m^2)$.

על־ידי שימוש בהיוריסטיקות איחוד על פי דרגה וכיווץ מסלולים, אפשר לקצר את זמן הריצה במקרה הגרוע. נשתמש ביער קבוצות זרות $\mathcal{S} = \{S_i\}$ שבו כל קבוצה S_i (שהיא עצמה עץ) מקבילה לעץ T_i מן היער $\mathcal{F}$. אולם מבנה העץ בתוך קבוצה S_i לא בהכרח מקביל לזה של T_i. למעשה, המימוש של S_i אינו שומר את היחסים המדויקים בין אבות ובנים בעץ, אולם למרות זאת מאפשר לנו לקבוע את עומקו של כל צומת ב־T_i.

הרעיון הבסיסי הוא לתחזק בכל צומת v "פסאודו־מרחק" $d[v]$, המוגדר כך שסכום הפסאודו־מרחקים של הצמתים לאורך המסלול מ־v לשורש הקבוצה שלו S_i שווה לעומקו של v ב־T_i. כלומר, אם המסלול מ־v לשורשו ב־S_i הוא $v_0, v_1, \ldots, v_k$, כאשר $v_0 = v$ ו־v_k הוא השורש של S_i, אזי עומקו של v ב־T_i הוא $\sum_{j=0}^{k} d[v_j]$.

ב. כתוב מימוש של MAKE-TREE.

ג. הראה כיצד יש לשנות את FIND-SET כדי לממש את FIND-DEPTH. המימוש שלך צריך לבצע כיווץ מסלולים וזמן הריצה שלו צריך להיות לינארי באורך מסלול הגילוי. עליך להבטיח שהמימוש שלך מעדכן נכונה פסאודו־מרחקים.

ד. הראה כיצד יש לשנות את השגרות UNION ו־LINK כדי לממש את GRAFT(r, v), אשר יוצרת מן הקבוצות המכילות את r ואת v קבוצה אחת. עליך להבטיח שהמימוש שלך מעדכן נכונה פסאודו־מרחקים. זכור כי שורש הקבוצה S_i אינו בהכרח שורש העץ המקביל T_i.

ה. מצא חסם הדוק על זמן הריצה במקרה הגרוע של סדרה של m פעולות MAKE-TREE ו־FIND-DEPTH ו־GRAFT, אשר n מהן הן פעולות MAKE-TREE.

22-3 האלגוריתם הבלתי־מקוון של Tarjan למציאת אבות קדמונים משותפים עמוקים ביותר

אב קדמון משותף עמוק ביותר (least common ancestor) של שני צמתים u ו־v בעץ מושרש T, הוא הצומת w שהוא אב קדמון של u וגם של v, ואשר מבין האבות הקדמונים של שני הצמתים הוא בעל העומק המקסימלי. **בגרסה הבלתי־מקוונת של בעיית מציאת אבות קדמונים משותפים עמוקים ביותר** (off-line least-common-ancestors problem), נתונים לנו עץ מושרש T וסדרה כלשהי $P = \{\{u, v\}\}$ של זוגות בלתי־סדורים של צמתים ב־T. ברצוננו למצוא את האב הקדמון המשותף העמוק ביותר של כל אחד מן הזוגות ב־P.

כדי לפתור את הבעיה הנתונה, השגרה שלהלן מבצעת סריקה של T, כאשר הקריאה ההתחלתית היא LCA$(root[T])$. אנו מניחים שלפני הסריקה כל צומת צבוע בלבן (WHITE).


```
LCA(u)
1  MAKE-SET(u)
2  ancestor[FIND-SET(u)] ← u
3  for each child v of u in T
4      do LCA(v)
5         UNION(u, v)
6         ancestor[FIND-SET(u)] ← u
7  color[u] ← BLACK
8  for each node v such that {u, v} ∈ P
9      do if color[v] = BLACK
10        then print "The least common ancestor of"
                u "and" v "is" ancestor[FIND-SET(v)]
```

א. הראה ששורה 10 מבוצעת בדיוק פעם אחת עבור כל זוג $\{u,v\} \in P$.

ב. הראה שבעת הקריאה LCA(u), מספר הקבוצות במבנה הנתונים לקבוצות זרות שווה לעומקו של u ב־T.

ג. הוכח שהשגרה LCA מדפיסה נכונה את האב הקדמון המשותף העמוק ביותר של כל זוג $\{u,v\} \in P$.

ד. נתח את זמן הריצה של LCA בהנחה שמשתמשים במימוש מבנה הנתונים לקבוצות זרות שהוצג בסעיף 22.3.

הערות ביבליוגרפיות לפרק 22

רבות מן התוצאות החשובות עבור מבני הנתונים לקבוצות זרות נזקפות, לפחות בחלקן, לזכותו של R. E. Tarjan. החסם העליון $O(m\,\alpha(m, n))$ הוצג לראשונה על־ידי Tarjan [186, 188]. החסם העליון $O(m \lg^* n)$ הוכח קודם לכן על־ידי Hopcroft & Ullman [4, 103]. Tarjan & van Leeuwen [190] דנים בגרסאות שונות של היוריסטיקת כיווץ המסלולים, כולל "שיטות מעבר־יחיד" שהביצועים שלהן מציעים לעתים גורמים קבועים טובים יותר מאלה של שיטות שני המעברים. Gabow & Tarjan [76] מראים שביישומים מסוימים ניתן לבצע את הפעולות על קבוצות זרות בזמן $O(m)$.

Tarjan [187] הראה שדרוש חסם תחתון של $\Omega(m\,\alpha(m, n))$ על זמן הריצה של פעולות על כל מבנה נתונים לקבוצות זרות המקיים תנאים טכניים מסוימים. חסם תחתון זה הוכלל מאוחר יותר על־ידי Fredman & Saks [74], שהראו כי במקרה הגרוע, נדרשת בהכרח גישה ל־$\Omega(m\,\alpha(m, n))$ מילות זיכרון בנות $\lg n$ סיביות כל אחת.

VI אלגוריתמים על גרפים

בפסאודו־קוד שבהמשך.

ובו־$E[G]$ את קבוצת הקשתות שעליו נמסרה הודעה G נסמן ב־$V[G]$ את קבוצת הקודקודים ו

משפט. אם $O(|V|+|E|)$ הוא חסם־העליון הטוב ביותר שאפשר להבטיח עבור $E(V)$ "מספר ההשוואות" במיון־השוואות, אז $O(|V|+|E|)$ הוא חסם־העליון הטוב ביותר שאפשר להבטיח עבור $E(V)$ "מספר ההשוואות" במיון־השוואות שנכון לכל גרף G בעל $|V|$ קודקודים ו־$|E|$ קשתות.

הוכחה. נוכיח את המשפט עבור גרפים מכוונים. ההוכחה עבור גרפים לא־מכוונים דומה.

נתבונן בגרף G בעל $|V|$ קודקודים ו־$|E|$ קשתות. נניח שאלגוריתם כלשהו A מיין את G בעזרת $E(V,G)$ השוואות. נראה שקיים גרף G' בעל $|V|$ קודקודים ו־$|E|$ קשתות, שעבורו A מבצע $E(V,G)$ השוואות. מכאן נובע שמספר ההשוואות שמבצע A על G' הוא $\Omega(|V|+|E|)$, ומכאן נובע שמספר ההשוואות שמבצע A על G הוא $\Omega(|V|+|E|)$.

נתבונן בגרף G שמיין על־ידי A. נסמן ב־π את סדר־הטופולוגי שמתקבל מהרצת A על G. נסמן ב־$\pi(i)$ את הקודקוד שמופיע במקום ה־i ב־π. נסמן ב־$\pi^{-1}(v)$ את המקום שבו מופיע הקודקוד v ב־π. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π.

נתבונן בקשת (u,v) ב־G. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π.

נתבונן בקשת (u,v) ב־G. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π.

נתבונן בקשת (u,v) ב־G. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π.

נתבונן בקשת (u,v) ב־G. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π.

נתבונן בקשת (u,v) ב־G. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π.

נתבונן בקשת (u,v) ב־G. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π.

נתבונן בקשת (u,v) ב־G. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π. נסמן ב־$\pi^{-1}(u,v)$ את המקום שבו מופיעה הקשת (u,v) ב־π.

מבוא

23 אלגוריתמים בסיסיים על גרפים

בפרק זה נציג שיטות לייצוג גרף ולסריקתו. סריקת גרף פירושה מעבר שיטתי על קשתות הגרף לצורך ביקור בקדקודיו. אלגוריתם הסורק גרף יכול לגלות דברים רבים על מבנהו. אלגוריתמים רבים מתחילים בסריקת גרף הקלט שלהם לצורך איסוף מידע כזה על מבנה הגרף. אלגוריתמים אחרים על גרפים מאורגנים כהרחבות פשוטות של אלגוריתמים בסיסיים לסריקת גרף. שיטות לסריקת גרפים הן לב ליבו של תחום האלגוריתמים על גרפים.

בסעיף 23.1 נדון בשני הייצוגים החישוביים השכיחים ביותר של גרפים: רשימות סמיכות (adjacency lists) ומטריצת סמיכויות (adjacency matrix). בסעיף 23.2 נציג אלגוריתם פשוט לסריקת גרפים הנקרא חיפוש לרוחב־תחילה ונראה כיצד לבנות עץ רוחב־תחילה. בסעיף 23.3 נציג חיפוש לעומק־תחילה ונוכיח כמה תוצאות סטנדרטיות על הסדר שבו חיפוש לעומק־תחילה מבקר בקדקודים. בסעיף 23.4 נציג את היישום הממשי הראשון שלנו לחיפוש לעומק־תחילה: מיון טופולוגי של גרף מכוון ללא מעגלים. יישום שני של חיפוש לעומק־תחילה, מציאת הרכיבים הקשירים היטב של גרף מכוון, יובא בסעיף 23.5.

23.1 ייצוגים של גרפים

קיימות שתי דרכים מקובלות לייצוג גרף $G = (V, E)$: כאוסף רשימות סמיכות או כמטריצת סמיכויות. בדרך כלל מועדף הייצוג באמצעות רשימות סמיכות, משום שהוא מספק דרך קומפקטית לייצוג גרפים **דלילים** (sparse) — אלה שעבורם $|E|$ קטן בהרבה מ־$|V|^2$. רוב האלגוריתמים על גרפים המוצגים בספר זה מניחים שגרף הקלט מיוצג על־ידי רשימות סמיכות. עם זאת, הייצוג על־ידי מטריצת סמיכויות עשוי להיות עדיף כאשר הגרף **צפוף** (dense) — $|E|$ קרוב ל־$|V|^2$ — או כאשר נדרשת יכולת לגלות במהירות אם קיימת קשת המחברת שני קדקודים נתונים. לדוגמה, שניים מן האלגוריתמים המוצגים בפרק 26 למציאת מסלולים קצרים ביותר בין זוגות קדקודים מניחים שגרפי הקלט שלהם מיוצגים על־ידי מטריצות סמיכויות.

הייצוג על־ידי רשימות סמיכות (adjacency-list representation) של גרף $G = (V, E)$ מורכב ממערך Adj של $|V|$ רשימות, אחת עבור כל קדקוד ב־V. עבור כל $u \in V$, רשימת הסמיכות $Adj[u]$ מכילה מצביעים לכל הקדקודים v שעבורם קיימת קשת $(u, v) \in E$. כלומר, $Adj[u]$ מורכבת מכל הקדקודים הסמוכים ל־u ב־G. בדרך כלל, הקדקודים בכל רשימת סמיכות מאוחסנים בסדר שרירותי. איור 23.1(ii) הוא ייצוג על־ידי רשימות סמיכות של הגרף הבלתי־מכוון שבאיור 23.1(i). בצורה דומה, איור 23.2(ii) הוא ייצוג על־ידי רשימות סמיכות של הגרף המכוון שבאיור 23.2(i).

אם G הוא גרף מכוון, הסכום של אורכי כל רשימות הסמיכות הוא $|E|$, שכן קשת מן הצורה (u, v) מיוצגת על־ידי כך ש־v מופיע ב־$Adj[u]$ אם G הוא גרף בלתי־מכוון, הסכום של אורכי כל רשימות הסמיכות הוא $2|E|$, שכן אם (u, v) היא קשת בלתי־מכוונת, אזי u מופיע ברשימת הסמיכות של v, ולהיפך. אחד מיתרונותיו של הייצוג על־ידי רשימות סמיכות הוא שכמות הזיכרון הדרושה עבורו, הן לייצוג גרפים מכוונים והן לייצוג גרפים בלתי־מכוונים, היא $O(\max(V, E)) = O(V + E)$.

רשימות סמיכות ניתן להתאים בקלות לייצוג **גרפים משוקללים** (weighted graphs), דהיינו, גרפים שבהם לכל קשת מיוחס **משקל** (weight), הנתון בדרך כלל על־ידי **פונקציית משקל** (weight function) $w : E \to \mathbf{R}$. לדוגמה, יהי $G = (V, E)$ גרף משוקלל עם פונקציית משקל w. המשקל $w(u, v)$ של הקשת $(u, v) \in E$ פשוט מאוחסן עם הקדקוד v ברשימת הסמיכות של u. הייצוג על־ידי רשימות סמיכות חסין למדי, במובן זה שניתן להתאימו לתמיכה בטיפוסים רבים אחרים של גרפים.

חיסרון אפשרי של הייצוג על־ידי רשימות סמיכות טמון בכך שאין דרך מהירה יותר לקבוע אם קשת נתונה (u, v) שייכת לגרף מאשר חיפוש אחר v ברשימת הסמיכות $Adj[u]$. על חיסרון זה אפשר להתגבר על־ידי ייצוג באמצעות מטריצת סמיכויות, במחיר שימוש בנפח זיכרון גדול יותר אסימפטוטית.

עבור **ייצוג על־ידי מטריצת סמיכויות** (adjacency-matrix representation) של גרף $G = (V, E)$, אנו מניחים שהקדקודים ממוספרים $1, 2, \ldots, |V|$ בסדר שרירותי. מטריצת הסמיכויות המייצגת גרף G היא מטריצה $A = (a_{ij})$ שממדיה $|V| \times |V|$ וערכי איבריה:

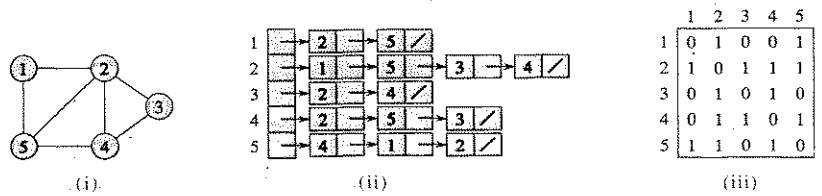
$$a_{ij} = \begin{cases} 1 & \text{אם } (i, j) \in E \\ 0 & \text{אחרת} \end{cases}$$

באיורים 23.1(iii) ו־23.2(iii) מוצגות מטריצות הסמיכויות של הגרף הבלתי־מכוון והגרף המכוון מאיורים 23.1(i) ו־23.2(i), בהתאמה. למטריצת סמיכויות של גרף דרוש $\Theta(V^2)$ מקום בזיכרון, ללא תלות במספר הקשתות בגרף.

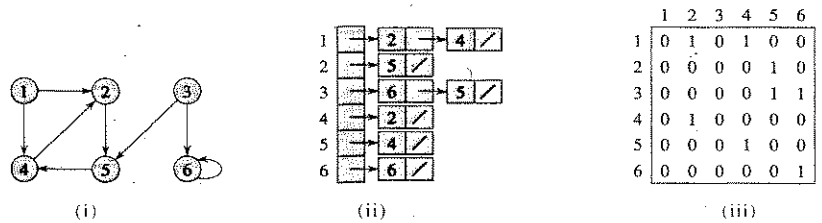
נשים לב לסימטרייה סביב האלכסון הראשי במטריצת הסמיכויות שבאיור 23.1(iii). נגדיר את **המטריצה המוחלפת** (transpose) של מטריצה $A = (a_{ij})$ כמטריצה $A^{\mathrm{T}} = (a_{ij}{}^{\mathrm{T}})$, הנתונה על־ידי $a_{ij}{}^{\mathrm{T}} = a_{ji}$. מכיוון שבגרף בלתי־מכוון (u, v) ו־(v, u) מייצגים אותה קשת, מטריצת הסמיכויות A של גרף בלתי־מכוון היא המטריצה המוחלפת של עצמה: $A = A^{\mathrm{T}}$. ביישומים מסוימים משתלם לאחסן רק את האיברים של מטריצת הסמיכויות הנמצאים באלכסון הראשי ומעליו, ובכך לצמצם כמעט למחצית את כמות הזיכרון הדרושה לאחסון הגרף.

בדומה לייצוג גרף על־ידי רשימות סמיכות, גם ייצוג על־ידי מטריצת סמיכויות יכול לשמש לייצוג גרפים משוקללים. לדוגמה, אם $G = (V, E)$ הוא גרף משוקלל עם פונקציית משקל־קשתות w, המשקל $w(u, v)$ של הקשת $(u, v) \in E$ פשוט מאוחסן כערך התא שבשורה u ובעמודה v במטריצת הסמיכויות. אם קשת אינה קיימת, ניתן לאחסן את הערך NIL בתא המתאים במטריצה, אם כי בביעות רבות נוח להשתמש בערך כגון 0 או ∞.

אף כי ייצוג על־ידי רשימות סמיכות יעיל אסימפטוטית לפחות כמו ייצוג על־ידי מטריצת סמיכויות, הרי שבשל פשטותה, מטריצת סמיכויות עשויה להיות הייצוג המועדף עבור גרפים קטנים דיים. יתר על כן, אם הגרף אינו משוקלל, יש יתרון נוסף לייצוג על־ידי מטריצת סמיכויות מבחינת כמות הזיכרון הדרושה: במקום להשתמש במילה אחת של זיכרון מחשב עבור כל איבר במטריצה, מטריצת הסמיכויות מכילה סיבית אחת בלבד לכל איבר.



איור 23.1 שני ייצוגים של גרף בלתי־מכוון. **(i)** גרף בלתי־מכוון G המכיל חמישה קדקודים ושבע קשתות. **(ii)** ייצוג של G על־ידי רשימות סמיכות. **(iii)** הייצוג של G על־ידי מטריצת סמיכויות.



איור 23.2 שני ייצוגים של גרף מכוון. **(i)** גרף מכוון G המכיל שישה קדקודים ושמונה קשתות. **(ii)** ייצוג של G על־ידי רשימות סמיכות. **(iii)** הייצוג של G על־ידי מטריצת סמיכויות.

תרגילים

23.1-1

בהינתן ייצוג של גרף מכוון על־ידי רשימות סמיכות, כמה זמן דרוש לחישוב דרגת היציאה של כל קדקוד? כמה זמן דרוש לחישוב דרגות הכניסה?

23.1-2

תאר ייצוג על־ידי רשימות סמיכות של עץ בינרי שלם בן 7 קדקודים. תאר ייצוג שקול על־ידי מטריצת סמיכויות. הנח שהקדקודים ממוספרים מ־1 עד 7 כמו בערימה בינרית.

23.1-3

הגרף המוחלף (transpose) של גרף מכוון $G = (V, E)$ הוא הגרף $G^T = (V, E^T)$, כאשר $E^T = \{(v, u) \in V \times V : (u, v) \in E\}$. כלומר, G^T מתקבל מ־G על־ידי היפוך כיווניהן של כל הקשתות ב־G. תאר אלגוריתמים יעילים לחישוב G^T בהינתן G, הן עבור ייצוג G על־ידי רשימות סמיכות והן עבור ייצוג G על־ידי מטריצת סמיכויות. נתח את זמני הריצה של האלגוריתמים שלך.

23.1-4

בהינתן ייצוג על־ידי רשימות סמיכות של מולטי־גרף (multigraph, ראה סעיף 5.4) $G = (V, E)$, תאר אלגוריתם שרץ בזמן $O(V + E)$ לבניית הייצוג על־ידי רשימות סמיכות של הגרף הבלתי־מכוון ה"שקול" $G' = (V, E')$, שבו E' מתקבלת מ־E על־ידי החלפת כל המולטי־קשתות בין שני קדקודים בקשת יחידה, וסילוק כל הלולאות העצמיות.

23.1-5

הריבוע (square) של גרף מכוון $G = (V, E)$ הוא הגרף $G^2 = (V, E^2)$ אשר בו $(u, w) \in E^2$ אם ורק אם קיים $v \in V$ שעבורו $(u, v) \in E$ וגם $(v, w) \in E$. כלומר, G^2 מכיל קשת בין u ו־w כאשר G מכיל מסלול בין u ו־w המורכב משתי קשתות בדיוק. תאר אלגוריתמים יעילים לחישוב G^2 מ־G, הן עבור ייצוג G על־ידי רשימות סמיכות והן עבור ייצוג G על־ידי מטריצת סמיכויות. נתח את זמני הריצה של האלגוריתמים שלך.

23.1-6

כאשר משתמשים בייצוג על־ידי מטריצת סמיכויות, רוב האלגוריתמים על גרפים רצים בזמן $\Theta(V^2)$, אולם ישנם כמה יוצאים מן הכלל. הראה שניתן לקבוע בזמן $O(V)$ אם גרף מכוון מכיל **בור** (sink) — קדקוד בעל דרגת כניסה $|V| - 1$ ודרגת יציאה 0 — גם אם משתמשים בייצוג על־ידי מטריצת סמיכויות.

23.1-7

מטריצת הקשירויות (incidence matrix) של גרף מכוון $G = (V, E)$ היא מטריצה $B = (b_{ij})$ ממדיה $|V| \times |E|$ וערכי איבריה:

$$b_{ij} = \begin{cases} -1 & \text{אם הקשת } j \text{ יוצאת מהקדקוד } i \\ 1 & \text{אם הקשת } j \text{ נכנסת לקדקוד } i \\ 0 & \text{אחרת} \end{cases}$$

תאר מה מייצגים איבריה של מטריצת המכפלה BB^T, כאשר B^T היא המטריצה המוחלפת של B.

23.2 חיפוש לרוחב

חיפוש לרוחב־תחילה (breadth-first search, ובקיצור: חיפוש לרוחב) הוא אחד האלגוריתמים הפשוטים ביותר לסריקת גרף ואב־טיפוס של אלגוריתמים חשובים רבים עבור גרפים. האלגוריתם של Dijkstra למציאת מסלולים קצרים ביותר ממקור יחיד (פרק 25) והאלגוריתם של Prim למציאת עץ פורש מינימלי (סעיף 24.2) מיישמים רעיונות דומים לאלה המיושמים בחיפוש לרוחב.

בהינתן גרף $G = (V, E)$ וקדקוד מסוים s המשמש כ**מקור** (source), חיפוש לרוחב בוחן בשיטתיות את הקשתות ב־G כדי "לגלות" כל קדקוד שניתן להגיע אליו מ־s. הוא מחשב את המרחק (מספר הקשתות המינימלי) מ־s לכל הקדקודים שניתן להגיע אליהם מ־s, וכן בונה "עץ רוחב", ששורשו s והוא מכיל את כל הקדקודים הללו. עבור כל קדקוד v שניתן להגיע אליו מ־s, המסלול מ־s ל־v בעץ הרוחב מקביל ל"מסלול קצר ביותר" מ־s ל־v ב־G, דהיינו, מסלול המכיל מספר מינימלי של קשתות. האלגוריתם פועל על גרפים מכוונים ובלתי־מכוונים כאחד.

חיפוש לרוחב מכונה כך מפני שהוא מקדם את הגבול בין קדקודים שהתגלו לקדקודים שטרם התגלו באופן אחיד לרוחב הגבול. כלומר, האלגוריתם מגלה את כל הקדקודים הנמצאים במרחק k מ־s לפני שהוא מגלה איזשהו קדקוד הנמצא במרחק $k+1$ מ־s.

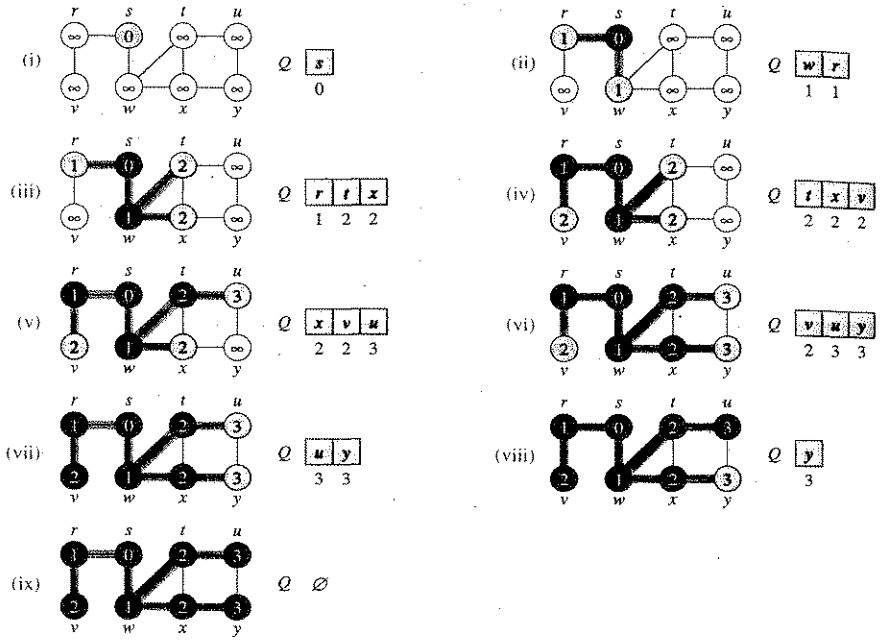
כדי לעקוב אחר התקדמותו, חיפוש לרוחב צובע כל קדקוד בלבן, באפור או בשחור. בתחילה, כל הקדקודים הם לבנים ועשויים להיצבע מאוחר יותר באפור ולאחר מכן בשחור. קדקוד **מתגלה** בפעם הראשונה שנתקלים בו במהלך החיפוש, וברגע זה הוא הופך להיות לא־לבן. קדקודים אפורים ושחורים הם אפוא קדקודים שהתגלו, אולם חיפוש לרוחב מבחין ביניהם כדי להבטיח שהחיפוש אכן יתקדם לרוחב. אם $(u, v) \in E$ והקדקוד u הוא שחור, אזי הקדקוד v הוא אפור או שחור; כלומר, כל הקדקודים הסמוכים לקדקודים שחורים כבר התגלו. לקדקוד אפור עשויים להיות כמה קדקודים סמוכים לבנים; קדקודים אפורים מייצגים אפוא את הגבול בין קדקודים שהתגלו לקדקודים שטרם התגלו.

חיפוש לרוחב בונה עץ רוחב, המכיל בתחילה רק את השורש, שהוא קדקוד המקור s. בכל פעם שמתגלה קדקוד לבן v במהלך הסריקה של רשימת הסמיכות של קדקוד u שכבר התגלה, הקדקוד v והקשת (u, v) נוספים לעץ. אנו אומרים ש־u הוא **הקודם** (predecessor) ל־v או **האב** (parent) של v בעץ הרוחב. מכיוון שקדקוד מתגלה לכל היותר פעם אחת, יש לו לכל היותר אב אחד. היחסים אב קדמון וצאצא בעץ הרוחב מוגדרים, כמו תמיד, יחסית לשורש s: אם u נמצא על מסלול בעץ מהשורש s לקדקוד v, אזי u הוא אב קדמון של v ו־v הוא צאצא של u.

השגרה BFS שלהלן, המבצעת חיפוש לרוחב של גרף, מניחה שגרף הקלט $G = (V, E)$ מיוצג על־ידי רשימות סמיכות. היא מתחזקת עבור כל קדקוד בגרף כמה מבני נתונים נוספים: צבעו של כל קדקוד $u \in V$ מאוחסן במשתנה $color[u]$. הקדקוד הקודם ל־u מאוחסן במשתנה $\pi[u]$. אם ל־u אין קדקוד הקודם לו (למשל, אם $u = s$ או אם u לא התגלה), אזי $\pi[u] = \text{NIL}$. המרחק מהמקור s לקדקוד u שמחשב האלגוריתם מאוחסן ב־$d[u]$. האלגוריתם משתמש גם בתור Q, כלומר, רשימה מהסוג "נכנס ראשון יוצא ראשון" (ראה סעיף 11.1), לניהול קבוצת הקדקודים האפורים.

```
BFS(G, s)
 1  for each vertex u ∈ V[G] − {s}
 2      do color[u] ← WHITE
 3         d[u] ← ∞
 4         π[u] ← NIL
 5  color[s] ← GRAY
 6  d[s] ← 0
 7  π[s] ← NIL
 8  Q ← {s}
 9  while Q ≠ ϕ
10      do u ← head[Q]
11         for each v ∈ Adj[u]
12             do if color[v] = WHITE
13                then color[v] ← GRAY
14                     d[v] ← d[u] + 1
15                     π[v] ← u
16                     ENQUEUE(Q, v)
17         DEQUEUE(Q)
18         color[u] ← BLACK
```


איור 23.3 מציג את התקדמותה של BFS על גרף לדוגמה.



איור 23.3 פעולתה של BFS על גרף בלתי־מכוון. הקשתות באיור הופכות להיות מוצללות עם הוספתן על־ידי BFS לעץ הרוחב. בתוך כל קדקוד u רשום $d[u]$. התור Q מוצג בתחילת כל איטרציה של לולאת ה־**while** שבשורות 9-18. מתחת לכל קדקוד בתור רשום מרחקו מקדקוד המקור s.

השגרה BFS פועלת כדלקמן: שורות 1-4 צובעות כל קדקוד בלבן, מציבות ב־$d[u]$ את הערך ∞ עבור כל קדקוד u, ומציבות באביו של כל קדקוד את הערך NIL. שורה 5 צובעת באפור את קדקוד המקור s, שכן עם תחילת השגרה הוא נחשב לקדקוד שהתגלה. שורה 6 מאתחלת את $d[s]$ ל־0, ושורה 7 מציבה NIL בקדקוד הקודם לקדקוד המקור s. שורה 8 מאתחלת את התור Q כך שיכיל רק את הקדקוד s; מכאן ואילך, Q מכיל תמיד את קבוצת הקדקודים האפורים.

הלולאה העיקרית בתכנית היא לולאת ה־**while** בשורות 9-18. הלולאה חוזרת ומתבצעת כל עוד נותרו קדקודים אפורים, שהם קדקודים שהתגלו אשר רשימות הסמיכות שלהם טרם נבחנו במלואן. שורה 10 קובעת מהו הקדקוד האפור u הנמצא בראש התור Q. לולאת ה־**for** בשורות 11-16 בוחנת כל קדקוד v ברשימת הסמיכות של u. אם v לבן, הרי ש־v טרם התגלה, והאלגוריתם מגלה אותו על־ידי ביצוע שורות 13-16: תחילה v נצבע באפור ובמשתנה המרחק שלו $d[v]$ מוצב הערך $d[u] + 1$. לאחר מכן, u נרשם כאביו של v. לבסוף, v מוכנס לסוף התור Q. לאחר שנבחנו כל הקדקודים ברשימת הסמיכות שלו, u נמחק מן התור ונצבע בשחור בשורות 17-18.

ניתוח

בטרם נוכיח את תכונותיו השונות של חיפוש לרוחב, ניטול על עצמנו משימה קלה יותר: ניתוח זמן הריצה של האלגוריתם על גרף קלט $G = (V, E)$. לאחר האתחול, אף קדקוד לעולם שוב אינו

1. בפרקים 25 ו־26 נכיר אלגוריתמים המחשבים את כל המסלולים הקצרים ביותר בין כל זוג קודקודים.

יהם מספר סופי זה.

מקבלים:

האי־שוויון המשולש, נתבונן בקודקוד ביניים x שמתקבלה במהלך החיפוש מקודקוד u. מהנחת האינדוקציה נובע כי $d[u] \geq \delta(s,u)$ ו־$d[x] \geq \delta(s,x)$. אבל 23.1 ומכאן

$$\delta(s,x) + \delta(x,u) \geq \delta(s,u) \geq d[u]$$

לכן נכונות אי־השוויון $d[x] \geq \delta(s,x)$ נובעת בשורה 14 ומלמה 23.1 אבל

$$d[x] + \delta(x,u) \geq d[u]$$

מתקבלת גם כן. ■

מסקנות

מסקנה 23.4 (נכונותו של חיפוש לרוחב)

עתה נוכל להוכיח כי חיפוש לרוחב מחשב בהכרח מרחקים של מסלולים קצרים ביותר.

תהי $G = (V, E)$ גרף מכוון או בלתי־מכוון, ויהי $s \in V$ קודקוד כלשהו. אז, עבור כל קשת $(u, v) \in E$ מתקיים:

$$\delta(s,v) \leq \delta(s,u) + 1$$

/

למה 23.1

יהי $G = (V, E)$ גרף מכוון או בלתי־מכוון, ויהי $s \in V$ קודקוד כלשהו. אז, עבור כל קשת $(u, v) \in E$ מתקיים:

$$\delta(s,v) \leq \delta(s,u) + 1$$

קצר ביותר: מסלול קצר ביותר (shortest path) מ־u ל־v בגרף $G = (V, E)$ הוא מסלול p בעל משקל $w(p) = \delta(u,v)$, כאשר $\delta(u,v) = \infty$ אם לא קיים מסלול מ־u ל־v. מסלול כלשהו p מ־u ל־v נקרא **מסלול קצר ביותר** (shortest-path distance) מ־u ל־v. במסמך הנוכחי כל משקלות המסלולים נתונים ביחידות של קשתות (כלומר, משקל כל קשת הוא 1). בפרק 24 נגדיר את המושג מסלול קצר ביותר במונחים של משקלות קשתות כלליים יותר.

מסלולים קצרים ביותר

אלגוריתם חיפוש לרוחב מבטיח שבעת גילוי כל קודקוד v, הוא מוצא מסלול קצר ביותר מקודקוד המקור s אל v, אם קיים מסלול כזה. אלגוריתם חיפוש לרוחב פועל גם על גרפים מכוונים או בלתי־מכוונים. נתחיל בכך שנגדיר את המושג מרחק מקודקוד המקור s לכל קודקוד $v \in V$ בגרף $G = (V, E)$, שהוא מספר הקשתות במסלול קצר ביותר (כלומר, קצר ביותר במובן של מספר הקשתות) מ־s ל־v. אם אין מסלול מ־s ל־v, אז נגדיר את המרחק להיות ∞. נוכיח בהמשך פרק זה שחיפוש לרוחב מחשב את המרחק הזה.

למה 23.2

יהי $G = (V, E)$ גרף מכוון או בלתי־מכוון, ויהי $s \in V$ קודקוד כלשהו. אז, עבור כל קשת $(u, v) \in E$ מתקיים:

$$\delta(s,v) \leq \delta(s,u) + 1$$

הוכחה אם u נגיש מ־s, אז גם v נגיש, ומשפט המשולש מבטיח את אי־השוויון. אם u אינו נגיש מ־s, אז $\delta(s,u) = \infty$, ואי־השוויון מתקיים. ■

למה 23.3

יהי $G = (V, E)$ גרף מכוון או בלתי־מכוון, ונניח שאלגוריתם חיפוש לרוחב מופעל על G מקודקוד מקור נתון $s \in V$. אז, עם סיום הריצה, עבור כל קודקוד $v \in V$, התכונה $d[v]$ שחושב על־ידי BFS מקיימת $d[v] \geq \delta(s,v)$.

הוכחה נוכיח את הטענה באינדוקציה על מספר הפעמים שקודקוד מוצב בתור Q. הנחת האינדוקציה היא כי $d[v] \geq \delta(s,v)$ עבור כל $v \in V$.

בסיס האינדוקציה הוא המצב מיד לאחר ש־s מוצב ב־Q בשורה 8 של BFS. הנחת האינדוקציה מתקיימת מאז שבתך $d[s] = 0 = \delta(s,s) \leq \delta(s,v) \leq \infty$ וכן $d[v] = \infty$ עבור כל $v \in V - \{s\}$.

לחלק האינדוקציה, נתבונן בקודקוד v שמתגלה במהלך החיפוש מקודקוד u. מהנחת האינדוקציה עבור הקודקוד u שהוצא מ־Q, $d[u] \geq \delta(s,u)$. מן ההשמה המבוצעת בשורה 14 ומלמה 23.1 אנו יודעים כי

$$d[v] = d[u] + 1 \geq \delta(s,u) + 1 \geq \delta(s,v)$$

ומכאן נובע שהערך שמושם ב־$d[v]$ עבור כל $v \in V$ לאחר גילויו הוא חסם עליון על $\delta(s,v)$. החלק הבא של ההוכחה, שבו נראה שערכי d הם לא יותר מאשר המרחקים, יוכיח שמתקיים שוויון עבור כל קודקוד.

נצפה בקבוץ, ולכן הזמן הכולל שמושקע בשורה 12 מבטיחה שעבור קודקוד יוכנס לתור פחות או יותר פעם אחת לכל היותר, ולכן מצפה בקבוץ, ולכן הזמן הכולל שמושקע בשורה 12 מבטיחה שעבור קודקוד יוכנס לתור פחות או יותר פעם אחת לכל היותר, ולכן נסיבות שבהן קודקוד מוכנס לתור יותר מפעם אחת. יחד עם זאת, כל קודקוד שמוכנס לתור מוצא ממנו בסופו של דבר, ולכן הזמן הכולל שמושקע בפעולות על התור הוא $O(V)$. מאחר שרשימת השכנות של כל קודקוד נסרקת רק כאשר הקודקוד מוצא מהתור, לכל היותר פעם אחת, סך כל רשימות השכנות הנסרקות הוא $O(E)$. מאחר שהזמן הנוסף שמושקע באתחול הוא $O(V)$, זמן הריצה של BFS הוא $O(V+E)$. כלומר, אלגוריתם חיפוש לרוחב רץ בזמן לינארי ביחס לגודלה של רשימת השכנות המתארת את G.

23.2 חיפוש לרוחב

בהינתן גרף $G = (V, E)$ ומקור s, חיפוש לרוחב סורק באופן שיטתי את הקשתות של G כדי "לגלות" כל קודקוד הנגיש מ־s. הוא מחשב את המרחק (מספר הקשתות המינימלי) מ־s לכל קודקוד הנגיש מ־s. הוא מייצר עץ חיפוש לרוחב עם השורש s שמכיל את כל הקודקודים הנגישים. עבור כל קודקוד v הנגיש מ־s, המסלול בעץ חיפוש לרוחב מתאים למסלול קצר ביותר מ־s ל־v בגרף G. אלגוריתם חיפוש לרוחב עובד על גרפים מכוונים ועל גרפים בלתי־מכוונים.


```
BFS(G, s)
1  for each vertex u ∈ V[G] − {s}
2      do color[u] ← WHITE
3         d[u] ← ∞
4         π[u] ← NIL
5  color[s] ← GRAY
6  d[s] ← 0
7  π[s] ← NIL
8  Q ← ∅
9  ENQUEUE(Q, s)
10 while Q ≠ ∅
11     do u ← DEQUEUE(Q)
12        for each v ∈ Adj[u]
13            do if color[v] = WHITE
14                  then color[v] ← GRAY
15                       d[v] ← d[u] + 1
16                       π[v] ← u
17                       ENQUEUE(Q, v)
18        color[u] ← BLACK
```

איור 23.3 הפעולה של BFS על גרף לא־מכוון. הקשתות של עץ החיפוש לרוחב מוצגות בהדגשה, וכל קודקוד u מוצג עם הערכים $d[u]$ ו־$\pi[u]$ שלו. הקודקודים מוצגים בסדר שבו הם מוצאים מהתור Q בשורה 11.

$d[v]$ של d.

ובמקרה שבו v נגיש מ־s, אז $d[v]$ שווה לאורך של מסלול קצר ביותר מ־s ל־v, שכן המסלול הקצר ביותר שמגיע ל־v לפני כל הקודקודים האחרים שמרוחקים ממנו, אם קיים כזה, חייב להכיל קודם כל מסלול קצר ביותר מ־s ל־v. ■

$$\begin{aligned} d[v] &= d[u] + 1 \\ &\geq \delta(s,u) + 1 \\ &\geq \delta(s,v) \end{aligned}$$

החלק העיקרי של ההוכחה מתייחס לקדקודים שניתן להגיע אליהם מ־s. נסמן ב־V_k את קבוצת הקדקודים שמרחקם מ־s הוא k; כלומר, $V_k = \{ v \in V : \delta(s, v) = k \}$. ההוכחה נעשית באינדוקציה על k. הנחת האינדוקציה היא שעבור כל קדקוד $v \in V_k$, קיימת בדיוק נקודה אחת במהלך ביצועה של BFS שבה:

- v נצבע באפור,
- $d[v]$ מקבל את הערך k,
- אם $v \neq s$, אזי $\pi[v]$ מקבל את הערך u עבור איזשהו $u \in V_{k-1}$,
- v מוכנס לתור Q.

כאמור, קיימת בוודאות לכל היותר נקודה אחת כזאת.

הבסיס הוא עבור $k = 0$. אנו מקבלים $V_0 = \{ s \}$, שכן המקור s הוא הקדקוד היחיד שמרחקו מ־s הוא 0. במהלך האתחול, s נצבע באפור, $d[s]$ מקבל את הערך 0, ו־s מוכנס ל־Q, כך שהנחת האינדוקציה מתקיימת.

עבור צעד האינדוקציה, נתחיל בכך שנשים לב כי התור Q אינו מתרוקן אף פעם עד אשר האלגוריתם עוצר, וכמו כן, מרגע שקדקוד u מוכנס לתור, $d[u]$ ו־$\pi[u]$ אינם משתנים עוד לעולם. אם כן, לפי למה 23.3, אם קדקודים מוכנסים לתור במהלך ביצוע האלגוריתם בסדר $v_1, v_2, \ldots, v_r$, אזי סדרת המרחקים היא מונוטונית עולה: $d[v_i] \leq d[v_{i+1}]$ עבור $i = 1, 2, \ldots, r-1$.

הבה נתבונן עתה בקדקוד כלשהו $v \in V_k$ עבור $k \geq 1$. מתכונת המונוטוניות, מכך ש־$d[v] \geq k-1$ (על פי למה 23.2) ומהנחת האינדוקציה, נובע שאם v מתגלה בכלל, הרי שבהכרח הוא מתגלה רק לאחר שכל הקדקודים השייכים ל־V_{k-1} כבר הוכנסו לתור.

מכיוון ש־$\delta(s, v) = k$, קיים מסלול בן k קשתות מ־s ל־v, ולכן קיים קדקוד $u \in V_{k-1}$ כך ש־$(u, v) \in E$. בלי הגבלת הכלליות, יהי u קדקוד ראשון כזה הנצבע באפור, צביעה החייבת להתרחש, שכן, באינדוקציה, כל הקדקודים ב־V_{k-1} נצבעים באפור. הקוד של BFS מכניס לתור כל קדקוד שנצבע באפור, ומכאן שבסופו של דבר u חייב להופיע כראש התור בעת ביצוע שורה 10. כאשר u מופיע כראש התור, רשימות השמיכות שלו נסרקות ו־v מתגלה. (הקדקוד v לא יכול היה להתגלות קודם לכן, שכן הוא אינו סמוך לשום קדקוד השייך ל־V_j, עבור $j < k-1$, אחרת v לא היה יכול להשתייך ל־V_k, ולפי ההנחה, u הוא הקדקוד הראשון המתגלה ב־V_{k-1} אשר v סמוך אליו.) שורה 13 צובעת את v באפור, שורה 14 מציבה $d[v] = d[u] + 1 = k$, שורה 15 מציבה ב־$\pi[v]$ את u, ושורה 16 מכניסה את v לתור. מאחר ש־v הוא קדקוד כלשהו ב־V_k, הרי שבכך הוכחה הנחת האינדוקציה.

להשלמת הוכחת המשפט, נשים לב שאם $v \in V_k$, אזי, על פי מה שראינו זה עתה, $\pi[v] \in V_{k-1}$. לכן נוכל לקבל מסלול קצר ביותר מ־s ל־v על־ידי כך שניקח מסלול קצר ביותר מ־s ל־$\pi[v]$ ונצרף לו את הקשת $(\pi[v], v)$. ■

עצי רוחב

השגרה BFS בונה במהלך סריקת הגרף עץ רוחב, כמודגם באיור 23.3. העץ מיוצג על־ידי השדה π של כל קדקוד. באופן פורמלי יותר, עבור גרף $G = (V, E)$ עם מקור s, אנו מגדירים את **תת־גרף הקודמים** (predecessor subgraph) של G כגרף $G_\pi = (V_\pi, E_\pi)$, כאשר:

$$V_\pi = \{ v \in V : \pi[v] \neq \text{NIL} \} \cup \{ s \}$$

וכן,

$$E_\pi = \{ (\pi[v], v) \in E : v \in V_\pi - \{ s \} \}$$

תת־גרף הקודמים G_π הוא **עץ רוחב** (breadth-first tree) אם V_π מורכבת מן הקדקודים שניתן להגיע אליהם מ־s, ועבור כל $v \in V_\pi$, קיים ב־G_π מסלול פשוט יחיד מ־s ל־v שהוא גם מסלול קצר ביותר מ־s ל־v ב־G. עץ רוחב הוא אכן עץ, שכן הוא קשיר ו־$|E_\pi| = |V_\pi| - 1$ (ראה משפט B.2). הקשתות ב־E_π נקראות **קשתות עץ** (tree edges).

הלמה הבאה מראה שלאחר הרצה של BFS על גרף G ממקור s, תת־גרף הקודמים הוא עץ רוחב.

למה 23.5

כאשר השגרה BFS מופעלת על גרף $G = (V, E)$, מכוון או בלתי־מכוון, היא בונה את השדות π כך שתת־גרף הקודמים $G_\pi = (V_\pi, E_\pi)$ הוא עץ רוחב.

הוכחה שורה 15 של BFS מציבה $\pi[v] = u$ רק אם $(u, v) \in E$ וגם $\delta(s, v) < \infty$ — כלומר, רק אם v ניתן להגעה מ־s — ולכן V_π מורכבת מן הקדקודים ב־V שניתן להגיע אליהם מ־s. מכיוון ש־G_π מהווה עץ, הוא מכיל מסלול יחיד מ־s לכל קדקוד ב־V_π. על־ידי הפעלה אינדוקטיבית של משפט 23.4 אנו מסיקים שכל מסלול כזה הוא מסלול קצר ביותר. ■

השגרה הבאה מדפיסה את הקדקודים לאורכו של מסלול קצר ביותר מ־s ל־v, בהנחה שכבר התבצעה הרצה של BFS לחישוב עץ המסלולים הקצרים ביותר.

```
PRINT-PATH( G, s, v )
1  if v = s
2    then print s
3    else if π[v] = NIL
4           then print "no path from" s "to" v "exists"
5           else PRINT-PATH( G, s, π[v] )
6                print v
```

זמן הריצה של השגרה לינארי במספר הקדקודים על המסלול המודפס, שכן כל קריאה רקורסיבית מתבצעת על מסלול הקצר בקדקוד אחד מזה של הקריאה הקודמת.

תרגילים

23.2-1

הראה את תוצאת ההרצה של חיפוש לרוחב על הגרף המכוון מאיור 23.2(i), כאשר הקדקוד 3 משמש כמקור.

23.2-2

הראה את תוצאת ההרצה של חיפוש לרוחב על הגרף הבלתי־מכוון מאיור 23.3, כאשר הקדקוד u משמש כמקור.

23.2-3
מהו זמן הריצה של BFS כאשר גרף הקלט שלה מיוצג על־ידי מטריצת סמיכויות והאלגוריתם מותאם לאופן ייצוג זה של הקלט?

23.2-4
הראה שבחיפוש לרוחב, הערך $d[u]$ הניתן לקדקוד u אינו תלוי בסדר שבו מופיעים הקדקודים ברשימת הסמיכות.

23.2-5
הבא דוגמה לגרף מכוון $G = (V, E)$, קדקוד מקור $s \in V$, וקבוצה של קשתות עץ $E_\pi \subseteq E$, כך שעבור כל קדקוד $v \in V$, המסלול היחיד ב־E_π מ־s ל־v הוא מסלול קצר ביותר ב־G, אולם קבוצת הקשתות E_π אינה יכולה להתקבל מהרצת BFS על G, ואין זה משנה כיצד מסודרים הקדקודים בכל רשימת סמיכות.

23.2-6
כתוב אלגוריתם יעיל הקובע אם גרף בלתי־מכוון הוא דו־צדדי.

23.2-7 *
הקוטר (diameter) של עץ $T = (V, E)$ נתון על־ידי:

$$\max_{u,v \in V} \delta(u, v)$$

כלומר, הקוטר הוא הגדול מבין מרחקי המסלולים הקצרים ביותר בעץ. כתוב אלגוריתם יעיל לחישוב קוטרו של עץ, ונתח את זמן הריצה של האלגוריתם שלך.

23.2-8
יהי $G = (V, E)$ גרף בלתי־מכוון. כתוב אלגוריתם שזמן ריצתו $O(V + E)$ לחישוב מסלול ב־G אשר עובר בכל קשת ב־E בדיוק פעם אחת בכל כיוון. תאר כיצד תוכל למצוא את דרך היציאה ממבוך אם תינתן לך כמות גדולה של מטבעות.

23.3 חיפוש לעומק

האסטרטגיה הנקטת ב**חיפוש לעומק־תחילה** (depth-first search, ובקיצור: חיפוש לעומק) היא, כפי שמרמז שמו, לחפש "עמוק יותר" בגרף ככל שהדבר אפשרי. בחיפוש לעומק נבדקות הקשתות היוצאות מן הקדקוד שהוא אחרון הקדקודים שהתגלו שעדיין יש לו קשתות יוצאות ממנו וטרם נבדקו. לאחר שנבדקו כל הקשתות היוצאות מ־v, החיפוש "נסוג" וממשיך בבדיקת הקשתות היוצאות מהקדקוד שממנו התגלה v. תהליך זה נמשך עד שמתגלים כל הקדקודים שניתן להגיע אליהם מקדקוד המקור המקורי. אם נותרו קדקודים שטרם התגלו, אחד מהם נבחר כמקור חדש, והחיפוש נמשך ממקור זה. התהליך כולו חוזר על עצמו עד שמתגלים כל הקדקודים.

כמו בחיפוש לרוחב, בכל פעם שמתגלה קדקוד v במהלך הסריקה של רשימת הסמיכות של קדקוד u שכבר התגלה, החיפוש לעומק מציין אירוע זה על־ידי הצבת u, הקודם של v, בשדה $\pi[v]$. שלא כמו חיפוש לרוחב, אשר תת־גרף הקודמים שלו יוצר עץ, תת־גרף הקודמים הנוצר על־ידי חיפוש לעומק עשוי להיות מורכב מכמה עצים, שכן החיפוש עשוי להתנהל מכמה מקורות.

לכן, **תת־גרף הקודמים** (predecessor subgraph) של חיפוש לעומק מוגדר בצורה שונה במקצת מזה של חיפוש לרוחב. נסמן אותו על־ידי $G_\pi = (V, E_\pi)$, כאשר:

$$E_\pi = \{(\pi[v], v) : \pi[v] \neq \text{NIL} \text{ וגם } v \in V\}$$

תת־גרף הקודמים של חיפוש לעומק יוצר **יער עומק** (depth-first forest) המורכב מכמה **עצי עומק** (depth-first trees). הקשתות ב־E_π נקראות **קשתות עץ** (tree edges).

כמו בחיפוש לרוחב, גם בחיפוש לעומק נצבעים קדקודים במהלך החיפוש כדי לציין את מצבם. כל קדקוד צבוע בתחילה לבן, הוא נצבע באפור כאשר הוא **מתגלה** במהלך החיפוש, והוא נצבע בשחור כאשר הטיפול בו **מסתיים**, כלומר, כאשר רשימת הסמיכות שלו נבדקה במלואה. שיטת פעולה זו מבטיחה שכל קדקוד יימצא לבסוף בדיוק בעץ עומק אחד, ולכן העצים האלה זרים.

בנוסף ליצירת יער עומק, חיפוש לעומק שומר בכל קדקוד **חותמות זמן** (timestamps). לכל קדקוד v יש שתי חותמות זמן: הראשונה, $d[v]$, מציינת מתי התגלה v לראשונה (ונצבע באפור); השנייה, $f[v]$, מציינת מתי סיים החיפוש לבחון את רשימת הסמיכות של v (וצבע אותו בשחור). חותמות זמן אלה משמשות באלגוריתמי גרפים רבים, ובאופן כללי מסייעות בניתוח התנהגותו של חיפוש לעומק.

השגרה DFS שלהלן שומרת את מועד הגילוי של קדקוד u במשתנה $d[u]$ ואת מועד סיום הטיפול בקדקוד u במשתנה $f[u]$. חותמות זמן אלה הן ערכים שלמים בין 1 ל־$2|V|$, שכן כל אחד מ־$|V|$ הקדקודים מתגלה פעם אחת ומטופל בו מסתיים פעם אחת. לכל קדקוד u מתקיים:

$$d[u] < f[u] \tag{23.1}$$

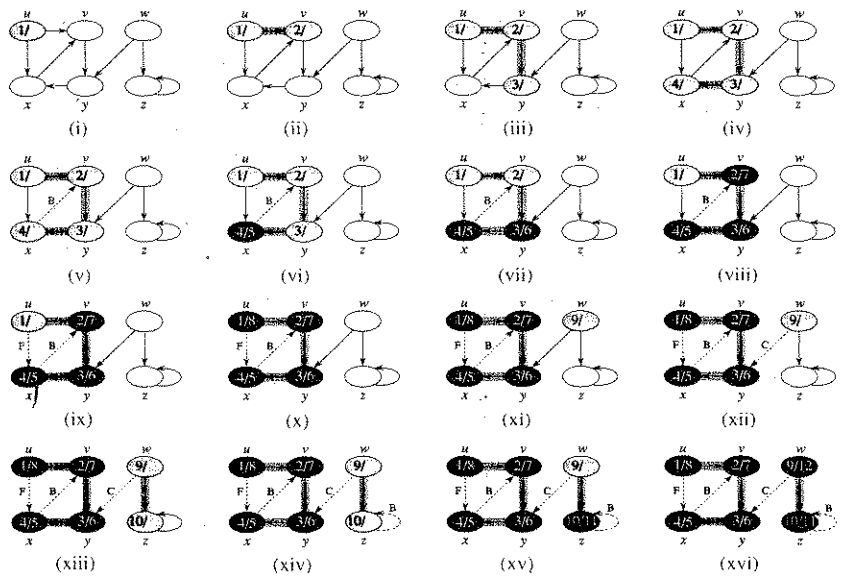
קדקוד u הוא לבן (WHITE) לפני הזמן $d[u]$, אפור (GRAY) בין הזמן $d[u]$ לזמן $f[u]$, ושחור (BLACK) אחרי הזמן $f[u]$.

הפסאדו־קוד שלהלן הוא האלגוריתם הבסיסי לחיפוש לעומק. גרף הקלט G יכול להיות מכוון או בלתי־מכוון. המשתנה *time* הוא משתנה גלובלי המשמש לניהול חותמות־הזמן.

```
DFS(G)
1  for each vertex u ∈ V[G]
2      do color[u] ← WHITE
3         π[u] ← NIL
4  time ← 0
5  for each vertex u ∈ V[G]
6      do if color[u] = WHITE
7         then DFS-VISIT(u)

DFS-VISIT(u)
1  color[u] ← GRAY          ▷ White vertex u has just been discovered.
2  d[u] ← time ← time + 1
3  for each v ∈ Adj[u]      ▷ Explore edge (u, v).
4      do if color[v] = WHITE
5         then π[v] ← u
6              DFS-VISIT(v)
7  color[u] ← BLACK         ▷ Blacken u; it is finished.
8  f[u] ← time ← time + 1
```

איור 23.4 מתאר את התקדמות אלגוריתם החיפוש לעומק DFS על הגרף מאיור 23.2.



איור 23.4 התקדמותו של אלגוריתם החיפוש לעומק DFS על גרף מכוון. עם בדיקתן על־ידי האלגוריתם, הקשתות מוצגות באיור כשהן מוצללות (אם הן קשתות עץ) או מקווקות (אחרת). קשתות שאינן שייכות לעצים מסומנות על־ידי B, C או F, בהתאם להיותן קשתות אחורה (back), קשתות חוצות (cross) או קשתות קדימה (forward). בתוך הקדקודים מופיעות חותמות־הזמן: מועד הגילוי משמאל ומועד הסיום מימין.

השגרה DFS פועלת באופן הבא: שורות 1-3 צובעות את כל הקדקודים בלבן ומאתחלות את השדות π שלהם לערך NIL. שורה 4 מאפסת את מונה הזמן הגלובלי. שורות 5-7 בודקות כל קדקוד ב־V בתורו, וכאשר נמצא קדקוד לבן, נערך בו "ביקור" באמצעות DFS-Visit. בכל פעם שמתבצעת בשורה 7 קריאה ל־DFS-Visit(u), u הופך לשורש של עץ חדש ביער העומק. כאשר DFS חוזרת, בכל קדקוד u כבר הוצב מועד גילוי $d[u]$ ומועד סיום $f[u]$.

בכל קריאה ל־DFS-Visit(u), הקדקוד u הוא בתחילה לבן. שורה 1 של DFS-Visit צובעת את u באפור, ושורה 2 מחשבת את מועד הגילוי של u על־ידי הוספת 1 למשתנה הגלובלי *time*, ומציבה אותו ב־$d[u]$. שורות 3-6 בוחנות כל קדקוד v הסמוך ל־u ואם v לבן, ערוכות בו ביקור באופן רקורסיבי. בכל פעם שקדקוד $v \in Adj[u]$ נבחן בשורה 3, אנו אומרים שהקשת (u, v) **נבדקת** על־ידי החיפוש לעומק. לבסוף, לאחר שנבדקו כל הקשתות היוצאות מ־u, שורות 7-8 צובעות את u בשחור, ומציבות ב־$f[u]$ את מועד הסיום.

מהו זמן הריצה של DFS? הלולאות שבשורות 1-3 ובשורות 5-7 של DFS מתבצעות בזמן $\Theta(V)$, לא כולל הזמן הדרוש לביצוע הקריאות ל־DFS-Visit. השגרה DFS-Visit נקראת בדיוק פעם אחת עבור כל קדקוד $v \in V$, שכן היא נקראת רק על קדקודים לבנים, ופעולתה הראשונה היא לצבוע את הקדקוד באפור. במהלך ביצועה של DFS-Visit(v), הלולאה שבשורות 3-6 מתבצעת $|Adj[v]|$ פעמים. מאחר שמתקיים:

$$\sum_{v \in V} |Adj[v]| = \Theta(E)$$

העלות הכוללת לביצוע שורות 3-6 של DFS-Visit היא $\Theta(E)$. זמן הריצה של DFS הוא אפוא $\Theta(V+E)$.

תכונות של חיפוש לעומק

חיפוש לעומק מניב מידע רב על מבנהו של גרף. התכונה הבסיסית ביותר אולי של חיפוש לעומק היא שתת־גרף הקודמים G_π אכן יוצר יער של עצים, שכן מבנה עצי העומק משקף במדויק את מבנה הקריאות הרקורסיביות ל־DFS-Visit. כלומר, $u = \pi[v]$ אם ורק אם השגרה DFS-Visit$[v]$ נקראת במהלך סריקה של רשימת הסמיכות של u.

תכונה חשובה אחרת של חיפוש לעומק היא שמועדי הגילוי והסיום יוצרים **מבנה סוגריים** (parenthesis structure). אם נייצג את גילויו של קדקוד u על־ידי סוגר שמאלי, "$(u$", ואת סיום הטיפול בו על־ידי סוגר ימני, "$u)$", אזי ההיסטוריה של הגילויים והסיומים תיצור ביטוי בנוי היטב, במובן זה שהסוגריים בביטוי יהיו מקוננים כהלכה. לדוגמה, קיימת התאמה בין החיפוש לעומק המוצג באיור 23.5(i) לבין מבנה הסוגריים בביטוי המופיע באיור 23.5(ii). דרך אחרת לניסוח התנאי של מבנה סוגריים מובאת במשפט הבא.

משפט 23.6 (משפט הסוגריים)
בכל חיפוש לעומק של גרף $G = (V, E)$ (מכוון או בלתי־מכוון), עבור כל שני קדקודים u ו־v, מתקיים בדיוק אחד משלושת התנאים הבאים:

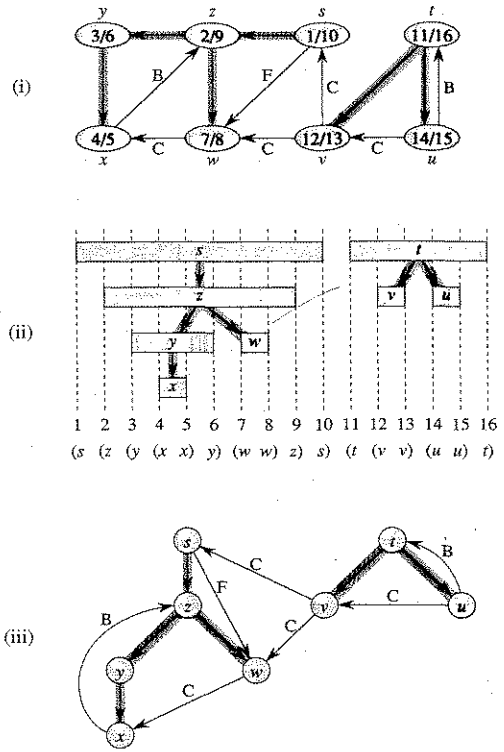
- הקטעים $[d[u], f[u]]$ ו־$[d[v], f[v]]$ זרים לחלוטין;
- הקטע $[d[u], f[u]]$ מוכל כולו בקטע $[d[v], f[v]]$, ו־u הוא צאצא של v בעץ העומק;
- הקטע $[d[v], f[v]]$ מוכל כולו בקטע $[d[u], f[u]]$, ו־v הוא צאצא של u בעץ העומק.

הוכחה נתחיל במקרה שבו $d[u] < d[v]$. יש לבחון שני תת־מקרים: האחד שבו $d[v] < f[u]$ והאחר שבו $d[v] > f[u]$. בתת־מקרה הראשון, $d[v] < f[u]$, ופירוש הדבר ש־v התגלה כאשר u היה עדיין אפור. מכך נובע ש־v הוא צאצא של u. יתר על כן, מכיוון ש־v התגלה אחרי u, כל הקשתות היוצאות ממנו נבדקות והטיפול בו מסתיים לפני שהחיפוש חוזר ל־u ומסיים את הטיפול בו. לכן במקרה זה, הקטע $[d[v], f[v]]$ מוכל כולו בקטע $[d[u], f[u]]$. בתת־מקרה השני, $f[u] < d[v]$, ומאי־שוויון (23.1) נובע שהקטעים $[d[u], f[u]]$ ו־$[d[v], f[v]]$ זרים.

המקרה שבו $d[u] > d[v]$ דומה, ובו מחליפים את תפקידיהם של u ו־v בטיעון שלעיל. ■

מסקנה 23.7 (קינון של קטעי צאצאים)
קדקוד v הוא צאצא ממש של קדקוד u ביער עומק של גרף G (מכוון או בלתי־מכוון) אם ורק אם $d[u] < d[v] < f[v] < f[u]$.

הוכחה ההוכחה נובעת ישירות ממשפט 23.6. ■



איור 23.5 תכונות של חיפוש לעומק. **(i)** תוצאות החיפוש לעומק של גרף מכוון. לקדקודים יש חותמות זמן, וסוגי הקשתות מסומנים כמו באיור 23.4. **(ii)** קיימת התאמה בין הקטעים התחומים על־ידי מועדי הגילוי והסיום של כל קדקוד לבין מבנה הסוגריים המוצג. כל מלבן נמתח לאורך הקטע הנתון על־ידי מועדי הגילוי והסיום של הקדקוד המתאים. באיור מופיעות קשתות העץ. אם שני קטעים חותכים זה את זה, אזי אחד מהם מקונן בתוך האחר, והקדקוד המתאים לקטע הקטן יותר הוא צאצא של הקדקוד המתאים לקטע הגדול יותר. **(iii)** הגרף מחלק (i) משורטט בשניות כך שכל קשתות העץ והקשתות קדימה פונות כלפי מטה בעץ העומק וכל הקשתות אחורה פונות כלפי מעלה, מצאצא לאב קדמון.

המשפט הבא מביא אפיון חשוב נוסף המאפשר לקבוע אם קדקוד אחד הוא צאצא של קדקוד אחר ביער העומק.

משפט 23.8 (משפט המסלול הלבן)

ביער עומק של גרף $G = (V, E)$ (מכוון או בלתי־מכוון), קדקוד v הוא צאצא של קדקוד u אם ורק אם בזמן $d[u]$ שבו מגלה החיפוש את u, ניתן להגיע מ־u ל־v לאורך מסלול המורכב כולו מקדקודים לבנים.

הוכחה $\Leftarrow$: נניח כי v הוא צאצא של u. יהי w קדקוד כלשהו על המסלול מ־u ל־v בעץ העומק, כך ש־w הוא צאצא של u. על פי מסקנה 23.7, $d[u] < d[w]$, ולכן w הוא לבן בזמן $d[u]$.

$\Rightarrow$: נניח כי בזמן $d[u]$ ניתן להגיע מ־u לקדקוד v לאורך מסלול המורכב מקדקודים לבנים בלבד, אולם v אינו נעשה צאצא של u בעץ העומק. נניח, בלי הגבלת הכלליות, שכל קדקוד אחר

לאורך המסלול הופך לצאצא של u (אחרת, יהי v הקדקוד הקרוב ביותר ל־u לאורך המסלול שאינו נעשה צאצא של u). יהי w הקודם של v במסלול, כך ש־w הוא צאצא של u (למעשה, w ו־u עשויים להיות אותו קדקוד). על פי מסקנה 23.7, $f[w] \leq f[u]$. נשים לב כי v חייב להתגלות לאחר ש־u מתגלה, אולם לפני סיום הטיפול ב־w. לפיכך, $d[u] < d[v] < f[w] \leq f[u]$. ממשפט 23.6 נובע שהקטע $[d[v], f[v]]$ מוכל כולו בקטע $[d[u], f[u]]$. על פי מסקנה 23.7, v חייב אפוא להיות צאצא של u. ■

סיווג קשתות

תכונה מעניינת אחרת של חיפוש לעומק היא שניתן להשתמש בו לסיווג הקשתות בגרף הקלט $G = (V, E)$. באמצעות סיווג זה של הקשתות ניתן ללקט מידע חשוב על הגרף. לדוגמה, בסעיף הבא נראה שגרף מכוון הוא ללא מעגלים אם ורק אם חיפוש לעומק אינו מגלה בו קשתות "אחורה" (למה 23.10).

ניתן להגדיר ארבעה סוגים של קשתות במונחי יער העומק G_π הנוצר במהלך חיפוש לעומק של G:

1. **קשתות עץ** (tree edges) הן קשתות השייכות ליער העומק G_π. קשת (u, v) היא קשת עץ אם v התגלה לראשונה על־ידי בדיקת הקשת (u, v).
2. **קשתות אחורה** (back edges) הן אותן קשתות (u, v) המחברות קדקוד u לאב קדמון v בעץ עומק. לולאות עצמיות, שעשויות להופיע בגרפים מכוונים, נחשבות לקשתות אחורה.
3. **קשתות קדימה** (forward edges) הן אותן קשתות (u, v) שאינן קשתות עץ, אשר מחברות קדקוד u לצאצא v בעץ עומק.
4. **קשתות חוצות** (cross edges) הן כל הקשתות האחרות. הן עשויות לקשר קדקודים באותו עץ עומק, כל עוד קדקוד אחד אינו צאצא של האחר, או לקשר קדקודים בעצי עומק שונים.

באיורים 23.4 ו־23.5 רשום ליד כל קשת הסוג שלה. איור 23.5(iii) גם מראה כיצד ניתן לשרטט את הגרף מאיור 23.5(i) כך שכל קשתות העץ והקשתות קדימה פונות כלפי מטה בעץ העומק וכל הקשתות אחורה פונות כלפי מעלה. כל גרף ניתן להצגה באופן זה.

את האלגוריתם DFS ניתן לשנות כך שיסווג קשתות כאשר הוא נתקל בהן. רעיון המפתח הוא שכל קשת (u, v) ניתן לסווג על פי צבעו של הקדקוד שמגיעים אליו כאשר הקשת נבדקת לראשונה (פרט לכך שלא ניתן להבחין בין קשתות קדימה לקשתות חוצות):

1. לבן (WHITE) מצביע על קשת עץ;
2. אפור (GREY) מצביע על קשת אחורה;
3. שחור (BLACK) מצביע על קשת קדימה או קשת חוצה.

המקרה הראשון נובע ישירות מאפיון האלגוריתם. עבור המקרה השני, נשים לב כי קדקודים אפורים יוצרים תמיד שרשרת לינארית של צאצאים המתאימה למחסנית הקריאות הפעילות עדיין ל־DFS-Visit; מספר הקדקודים האפורים גדול ב־1 מעומקו ביער העומק של הקדקוד האחרון שהתגלה. בדיקת קשתות מתקדמת תמיד מן הקדקוד האפור העמוק ביותר, כך שקשת שמגיעה לקדקוד אפור נוסף מגיעה לאב קדמון. המקרה השלישי מתייחס לאפשרות הנותרת; ניתן להראות שקשת (u, v) כזאת היא קשת קדימה אם $d[u] < d[v]$, וקשת חוצה אם $d[u] > d[v]$ (ראה תרגיל 23.3-4).

בגרף בלתי־מכוון עלולה להתעורר דו־משמעות מסוימת בסיווג הקשתות, שכן (u, v) ו־(v, u) הן למעשה אותה קשת. במקרה כזה, סוג הקשת נקבע כסוג *הראשון* ברשימת הסיווג החל על הקשת.

באופן שקול (ראה תרגיל 23.3-5), סוג הקשת נקבע על פי הכיוון (דהיינו, (u, v) או (v, u)) שבו נתקלים בה לראשונה במהלך ביצוע האלגוריתם.

נראה עתה שבחיפוש לעומק של גרף בלתי־מכוון לא מתקבלות לעולם קשתות קדימה וקשתות חוצות.

משפט 23.9
בחיפוש לעומק של גרף בלתי־מכוון G, כל קשת ב־G היא קשת עץ או שהיא קשת אחורה.

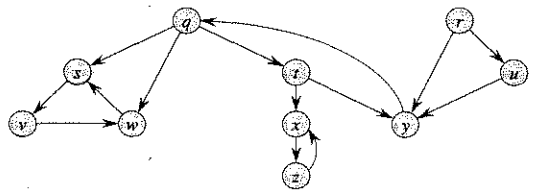
הוכחה תהי (u, v) קשת כלשהי ב־G ונניח, בלי הגבלת הכלליות, כי $d[u] < d[v]$. אזי, הגילוי של v והטיפול בו חייבים להסתיים לפני סיום הטיפול ב־u, שכן v שייך לרשימת הסמיכות של u. אם הקשת (u, v) נבדקת תחילה בכיוון מ־u ל־v, אזי (u, v) נעשית קשת עץ. אם (u, v) נבדקת תחילה בכיוון מ־v ל־u, אזי (u, v) היא קשת אחורה, שכן u עדיין אפור בעת שהקשת נבדקת לראשונה. ■

בסעיפים הבאים נראה יישומים רבים של משפטים אלה.

תרגילים

23.3-1
הכן טבלה של 3×3, שבה כותרות השורות והעמודות הן: לבן, אפור, שחור. בכל משבצת (i, j) ציין אם בנקודה כלשהי במהלך חיפוש לעומק של גרף מכוון יכולה להיות קשת מקדקוד בצבע i לקדקוד בצבע j. עבור כל קשת אפשרית, ציין לאילו סוגים היא עשויה להשתייך. ערוך טבלה שנייה כזאת עבור חיפוש לעומק של גרף בלתי־מכוון.

23.3-2
הראה כיצד פועל חיפוש לעומק על הגרף שבאיור 23.6. הנח כי לולאת ה־**for** בשורות 5-7 של DFS בוחנת את הקדקודים בסדר אלפביתי וכי סדר הקדקודים בכל רשימת סמיכות גם הוא אלפביתי. הראה את מועדי הגילוי והסיום של כל קדקוד ואת הסוג של כל קשת.



איור 23.6 גרף מכוון עבור תרגילים 23.3-2 ו־23.3-5.

23.3-3
הצג את מבנה הסוגריים של החיפוש לעומק מאיור 23.4.

23.3-4
הראה שקשת (u, v) היא -

א. קשת עץ או קשת קדימה אם ורק אם $d[u] < d[v] < f[v] < f[u]$.

ב. קשת אחורה אם ורק אם $d[v] < d[u] < f[u] < f[v]$.

ג. קשת חוצה אם ורק אם $d[u] > f[u] > d[v] > f[v]$.

23.3-5
הראה כי בגרף בלתי־מכוון, סיווג קשת (u, v) כקשת עץ או קשת אחורה על פי הכיוון (דהיינו, (u, v) או (v, u)) שבו נתקלים בה לראשונה במהלך חיפוש לעומק, שקול לסיווגה על פי סדר הופעת הסוגים ברשימת הסיווג.

23.3-6
הבא דוגמה נגדית לטענה שאם קיים מסלול מ־u ל־v בגרף מכוון G, ואם $d[u] < d[v]$ בחיפוש לעומק של G, אזי v הוא צאצא של u ביער העומק הנוצר.

23.3-7
שנה את הפסאודו־קוד של חיפוש לעומק כך שידפיס כל קשת בגרף מכוון G ביחד עם סוגה. הראה אילו שינויים, אם בכלל, יש להכניס אם G הוא גרף בלתי־מכוון.

23.3-8
הסבר כיצד ייתכן שקדקוד u השייך לגרף מכוון יימצא בסופו של דבר בעץ עומק המכיל את u בלבד, אף כי ל־u יש קשתות הנכנסות אליו וגם קשתות היוצאות ממנו.

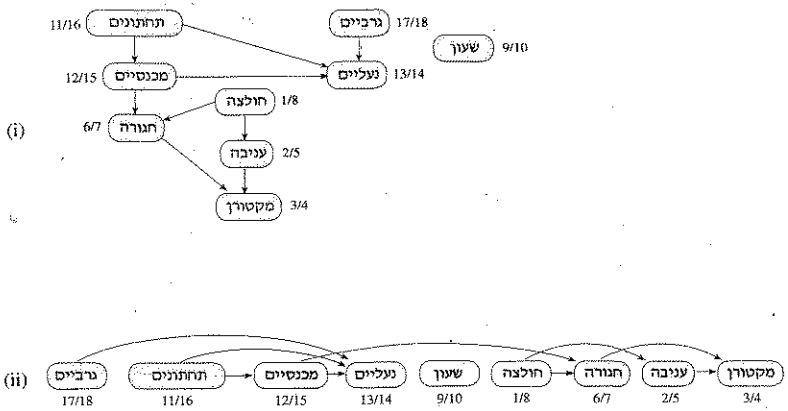
23.3-9
הראה כי ניתן להשתמש בחיפוש לעומק של גרף בלתי־מכוון G לזיהוי הרכיבים הקשירים של G וכי מספר העצים ביער העומק שווה למספר הרכיבים הקשירים ב־G. ביתר דיוק, הראה כיצד יש לשנות חיפוש לעומק כך שלכל קדקוד v תוצמד תווית $cc[v]$ המקבלת ערכים שלמים בין 1 ל־k, כאשר k הוא מספר הרכיבים הקשירים של G, כך ש־$cc[u] = cc[v]$ אם ורק אם u ו־v שייכים לאותו רכיב קשיר.

* 23.3-10
גרף מכוון $G = (V, E)$ הוא **חד־קשיר** (singly-connected) אם $u \rightsquigarrow v$ גורר שקיים לכל היותר מסלול פשוט אחד מ־u ל־v, עבור כל הקדקודים $u, v \in V$. כתוב אלגוריתם יעיל אשר קובע אם גרף מכוון הוא חד־קשיר או לא.

23.4 מיון טופולוגי

בסעיף זה נראה כיצד ניתן להשתמש בחיפוש לעומק לביצוע מיונים טופולוגיים של גרף מכוון ללא מעגלים, או גמ"ל, כפי שהוא מכונה לעתים. **מיון טופולוגי** (topological sort) של גמ"ל $G = (V, E)$ הוא סידור ליניארי של כל קדקודי G כך שאם G מכיל קשת (u, v), אזי u מופיע לפני v בסידור זה. (אם הגרף מכיל מעגלים, לא ניתן לסדר את קדקודיו בשום סדר לינארי.) ניתן לראות מיון טופולוגי של גרף כסידור של קדקודיו לאורך קו אופקי כך שכל הקשתות המכוונות פונות משמאל לימין. מיון טופולוגי שונה אפוא מהמיונים הרגילים שעסקנו בהם בחלק II.

גרפים מכוונים ללא מעגלים משמשים ביישומים רבים לציון קדימויות בתוך קבוצת מאורעות. הדוגמה המוצגת באיור 23.7 מתארת כיצד מתלבש פרופסור בומסטד בבוקר. הפרופסור חייב ללבוש בכל סדר שהוא (למשל, גרביים ומכנסיים). בגמ"ל שבאיור 23.7(i), קשת מכוונת (u, v) פירושה שיש ללבוש את הפריט u לפני הפריט v. מיון טופולוגי של גמ"ל זה נותן אפוא את הסדר שבו יש להתלבש. איור 23.7(ii) מראה את הגמ"ל הממוין טופולוגית כסידור של קדקודים לאורך קו אופקי כך שכל הקשתות המכוונות פונות משמאל לימין.



איור 23.7 (i) פרופסור בומסטד ממיין את בגדיו מיון טופולוגי בעת שהוא מתלבש. כל קשת מכוונת (u, v) פירושה שיש ללבוש את בגד u לפני בגד v. מועדי הגילוי והסיום המתקבלים במהלך חיפוש לעומק רשומים ליד כל קדקוד. **(ii)** אותו גרף לאחר שמוין מיון טופולוגי. קדקודיו מסודרים משמאל לימין בסדר יורד של מועדי סיום. אפשר לראות שכל הקשתות המכוונות פונות משמאל לימין.

האלגוריתם הפשוט הבא ממיין גמ"ל מיון טופולוגי.

TOPOLOGICAL-SORT(G)
1 call DFS(G) to compute finishing times $f[v]$ for each vertex v
2 as each vertex is finished, insert it onto the front of a linked list
3 **return** the linked list of vertices

איור 23.7(ii) מראה כיצד הקדקודים הממוינים מיון טופולוגי מופיעים בסדר הפוך של מועדי הסיום שלהם.

אפשר לבצע מיון טופולוגי בזמן $\Theta(V+E)$, שכן חיפוש לעומק מתבצע בזמן $\Theta(V+E)$ והכנסת כל אחד מ־$|V|$ הקדקודים לראש הרשימה המקושרת מתבצעת בזמן $O(1)$.

נוכיח את נכונותו של אלגוריתם זה בעזרת למת המפתח הבאה, המאפיינת גרפים מכוונים ללא מעגלים.

למה 23.10
גרף מכוון G אינו מכיל מעגלים אם ורק אם חיפוש לעומק של G אינו מניב קשתות אחורה.

הוכחה $\Leftarrow$: נניח שקיימת קשת אחורה (u, v). אזי, הקדקוד v הוא אב קדמון של הקדקוד u ביער העומק. קיים אפוא ב־G מסלול מ־v ל־u, והקשת אחורה (u, v) סוגרת מעגל.

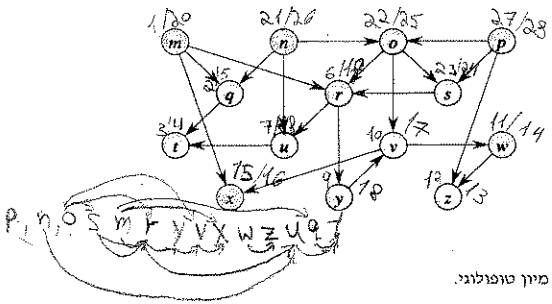
$\Rightarrow$: נניח ש־G מכיל מעגל c. נראה שחיפוש לעומק של G מניב קשת אחורה. יהי v הקדקוד הראשון המתגלה ב־c ותהי (u, v) הקשת הקודמת ב־c. בזמן $d[v]$ קיים מסלול של קדקודים לבנים מ־v ל־u. על פי משפט המסלול הלבן, הקדקוד u הופך להיות צאצא של הקדקוד v ביער העומק. לכן, (u, v) היא קשת אחורה. ■

משפט 23.11
האלגוריתם TOPOLOGICAL-SORT(G) יוצר מיון טופולוגי של גרף מכוון ללא מעגלים G.

הוכחה נניח שמריצים את DFS על גמ"ל נתון $G = (V, E)$ כדי לקבוע את מועדי הסיום של קדקודיו. די להראות שעבור זוג כלשהו של קדקודים שונים $u, v \in V$, אם קיימת ב־G קשת מ־u ל־v, אזי $f[u] > f[v]$. נתבונן בקשת כלשהי (u, v) הנבדקת על־ידי DFS(G). כאשר זו נבדקת, v אינו יכול להיות אפור, שכן אז הוא היה הופך לאב קדמון של u ר־(u, v) היתה אז קשת אחורה, בסתירה ללמה 23.10. לכן, v חייב להיות לבן או שחור. אם v לבן, הוא נעשה צאצא של u, ואז $f[u] > f[v]$. אם v שחור, גם אז $f[u] > f[v]$. לכן, לכל קשת (u, v) בגמ"ל מתקיים $f[u] > f[v]$, ובכך הוכח המשפט. ■

תרגילים

23.4-1
הראה את סידור הקדקודים הנוצר על־ידי TOPOLOGICAL-SORT כשמריצים אותו על הגמ"ל מאיור 23.8, באותן הנחות כמו בתרגיל 23.3-2.



איור 23.8 גמ"ל עבור מיון טופולוגי.

23.4-2
כתוב אלגוריתם אשר קובע אם גרף בלתי־מכוון נתון $G = (V, E)$ מכיל מעגל או לא. האלגוריתם שלך צריך לרוץ בזמן $O(V)$, ללא תלות ב־$|E|$.

23.4-3
הוכח או הפרך: אם גרף מכוון G מכיל מעגלים, אזי TOPOLOGICAL-SORT(G) יוצר סידור קדקודים הממזער את מספר הקשתות ה"רעות" שאינן קונסיסטנטיות עם הסידור המתקבל.

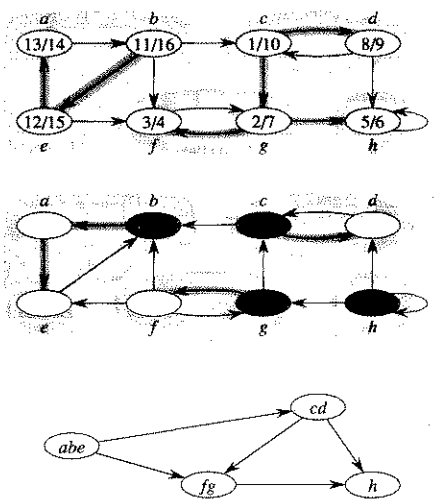
23.4-4
דרך אחרת לביצוע מיון טופולוגי של גרף מכוון ללא מעגלים $G = (V, E)$ היא למצוא פעם אחר פעם קדקוד בעל דרגת כניסה 0, להוציאו לפלט, ולהסיר אותו ואת כל הקשתות היוצאות ממנו מן הגרף. הסבר כיצד ניתן לממש רעיון זה כך שיתבצע בזמן $O(V+E)$. מה קורה לאלגוריתם זה אם G מכיל מעגלים?

23.5 רכיבים קשירים היטב

נתבונן עתה ביישום קלאסי של חיפוש לעומק: פירוק גרף מכוון לרכיביו הקשירים היטב. בסעיף זה נראה כיצד מבצעים פירוק כזה באמצעות שני חיפושים לעומק. אלגוריתמים רבים לעבודה עם גרפים מכוונים מתחילים את פעולתם בפירוק כזה; גישה זו מאפשרת לעתים קרובות לחלק את הבעיה המקורית לתת־בעיות — אחת לכל רכיב קשיר היטב. צירוף הפתרונות לתת־בעיות נעשה על־פי מבנה הקשרים בין רכיבים קשירים היטב; את המבנה הזה אפשר לייצג על־ידי גרף הידוע כגרף "הרכיבים", והוא יוגדר בתרגיל 23.5-4.

כאמור בפרק 5, רכיב קשיר היטב של גרף מכוון $G = (V, E)$ הוא קבוצה מקסימלית של קדקודים $U \subseteq V$ כך שעבור כל זוג קדקודים u ו־v ב־U, מתקיים $u \rightsquigarrow v$ וגם $v \rightsquigarrow u$; דהיינו, הקדקודים u ו־v ניתנים להגעה הדדית. באיור 23.9 מוצגת דוגמה.

האלגוריתם שלנו למציאת רכיבים קשירים היטב של גרף $G = (V, E)$ משתמש בגרף המוחלף של G, המוגדר בתרגיל 23.1-3 כגרף $G^T = (V, E^T)$, כאשר $E^T = \{(u, v) : (v, u) \in E\}$. כלומר, E^T מורכבת מהקשתות של G כאשר כיווניהן הפוכים. בהינתן ייצוג של G על־ידי רשימות סמיכות, הזמן ליצירת G^T הוא $O(V + E)$. מעניין לציין כי ל־G ול־G^T יש בדיוק אותם רכיבים קשירים היטב: u ו־v ניתנים להגעה הדדית ב־G אם ורק אם הם ניתנים להגעה הדדית ב־G^T. באיור 23.9(ii) מוצג הגרף המוחלף של הגרף מאיור 23.9(i), כאשר הרכיבים הקשירים היטב מוצללים.



איור 23.9 **(i)** גרף מכוון G. הרכיבים הקשירים היטב של G מוצגים באיור כאזורים מוצללים. בתוך כל קדקוד רשומים מועדי הגילוי והסיום שלו. קשתות־עץ מופיעות כשהן מוצללות. **(ii)** הגרף G^T — הגרף המוחלף של G. יער העומק המחושב בשורה 3 של STRONGLY-CONNECTED-COMPONENTS מוצג עם קשתות עץ מוצללות. כל רכיב קשיר היטב מתאים לעץ עומק אחד. הקדקודים b, c, g ו־h, המופיעים באיור בצבע שחור, הם אבות קדומים ביותר של כל קדקוד ברכיב הקשיר היטב שלהם; קדקודים אלה הם גם שורשים של עצי העומק הנוצרים על־ידי חיפוש לעומק של G^T. **(iii)** גרף הרכיבים חסר המעגלים G^{SCC} המתקבל על־ידי כך שמכווצים כל רכיב קשיר היטב של G לקדקוד יחיד.

האלגוריתם הבא, שזמן ריצתו לינארי (דהיינו, $\Theta(V + E)$), מחשב את הרכיבים הקשירים היטב של גרף מכוון $G = (V, E)$ באמצעות שני חיפושים לעומק, אחד על G ואחד על G^T.

```
STRONGLY-CONNECTED-COMPONENTS(G)
1  call DFS(G) to compute finishing times f[u] for each vertex u
2  compute G^T
3  call DFS(G^T), but in the main loop of DFS, consider the vertices
       in order of decreasing f[u] (as computed in line 1)
4  output the vertices of each tree in the depth-first forest of step 3 as a
       separate strongly connected component
```

במבט ראשון נראה כאילו אין קשר בין אלגוריתם פשוט למראה זה לבין רכיבים קשירים היטב. את יתרונו של סעיף זה נקדיש לפענוח חידת מבנהו של האלגוריתם ולהוכחת נכונותו. נתחיל בשתי הבחנות שימושיות.

למה 23.12

אם שני קדקודים נמצאים באותו רכיב קשיר היטב, אזי כל מסלול המחבר ביניהם לעולם אינו יוצא מהרכיב הקשיר היטב.

הוכחה יהיו u ו־v שני קדקודים באותו רכיב קשיר היטב. על פי ההגדרה של רכיב קשיר היטב ישנם מסלולים מ־u ל־v ומ־v ל־u. יהי w קדקוד על מסלול כלשהו $v \rightsquigarrow w \rightsquigarrow u$, כך ש־$w$ ניתן להגעה מ־u. יתר על כן, מאחר שקיים מסלול $u \rightsquigarrow v$, אנו יודעים כי u ניתן להגעה מ־w באמצעות המסלול $u \rightsquigarrow v \rightsquigarrow w$. לפיכך, u ו־w נמצאים באותו רכיב קשיר היטב. מכיוון ש־w נבחר אקראית, המשפט הוכח. ■

משפט 23.13

בכל חיפוש לעומק, כל הקדקודים השייכים לאותו רכיב קשיר היטב ממוקמים באותו עץ עומק.

הוכחה מבין הקדקודים הנמצאים ברכיב הקשיר היטב, יהי r הראשון שמתגלה. מאחר ש־r הוא הראשון להתגלות, הרי שבעת גילויו, הקדקודים האחרים ברכיב הקשיר היטב צבועים בלבן. קיימים מסלולים מ־r לכל קדקוד אחר ברכיב הקשיר היטב; מכיוון שמסלולים אלה לעולם אינם יוצאים מהרכיב הקשיר היטב (על פי למה 23.12), כל הקדקודים עליהם הם לבנים. לכן, על פי משפט המסלול הלבן, כל קדקוד ברכיב הקשיר היטב הופך להיות צאצא של r בעץ העומק. ■

בשאריתו של סעיף זה, הסימונים $d[u]$ ו־$f[u]$ מתייחסים למועדי הגילוי והסיום כפי שהם מחושבים על־ידי החיפוש לעומק הראשון בשורה 1 של STRONGLY-CONNECTED-COMPONENTS. בצורה דומה, הסימון $u \rightsquigarrow v$ מתייחס לקיומו של מסלול ב־G, ולא ב־G^T.

לצורך הוכחת נכונותה של השגרה STRONGLY-CONNECTED-COMPONENTS, נציג את המושג **אב קדום ביותר** (forefather) $\phi(u)$ של קדקוד u, שהוא הקדקוד w הניתן להגעה מ־u, אשר סיים אחרון בחיפוש לעומק שבשורה 1. במילים אחרות:

$\phi(u) =$ אותו קדקוד w שעבורו מתקיים $u \rightsquigarrow w$ ו־$f[w]$ מקסימלי.

יש לשים לב כי $\phi(u) = u$ אפשרי, שכן ניתן להגיע מ־u ל־u, ולכן:

$$f[u] \le f[\phi(u)] \tag{23.2}$$

כמו כן, ניתן להראות כי $\phi(\phi(u)) = \phi(u)$, באמצעות הטיעון שלהלן. עבור קדקודים כלשהם $u, v \in V$ מתקיים:

יש להם אותו אב קדום ביותר בעץ מכוון G.

משפט 23.16

בגרף מכוון $G = (V, E)$, שני קדקודים $u, v \in V$ נמצאים באותו רכיב קשיר חיטב אם ורק אם המשפט הבא נותן תוצאה חזקה יותר לגבי הקשר בין אבות קדומים ביותר לכיבים קשירים חיטב.

משפט 23.17

המשפט הבא נותן תיאור מדויק של עץ מכוון G.

$$u \rightsquigarrow v \Rightarrow f[\phi(v)] \leq f[\phi(u)] \tag{23.3}$$

מעיד על כך שאם $\phi(u) = \phi(v)$ אז u ו־v נמצאים באותו רכיב קשיר חיטב של G, זהו השורש של עץ העמק. עתה נכיח תכונות אלה.

משפט 23.14

בגרף מכוון $G = (V, E)$, יהא $\phi(u)$ הפסגה בעלת הזמן הגילוי הגדול ביותר של קדקוד $u \in V$ בלשונו בהגדרת העמק G. אז לכל קדקוד $v \in V$ שאליו אפשר להגיע מ־u ב־G, מתקיים $f[\phi(v)] \leq f[\phi(u)]$.

המשפט הראשון מצדיק את כינויו של $\phi(u)$ כ־"אב קדום ביותר" של u.

תוכחה אם $u = v$ אז $\phi(u) = \phi(v)$, ולכן $f[\phi(v)] = f[\phi(u)]$. אחרת, v הוא צאצא של u בעץ העמק, ולכן $d[u] < d[v] < f[v] < f[u]$ (משפט 23.8). אם $\phi(u)$ הוא אב קדמון של u, אז $\phi(u)$ הוא אב קדמון של v, ולכן $f[\phi(v)] \leq f[\phi(u)]$ (משפט 23.3).

נותר לברר שאם $\phi(u)$ הוא הפסגה בעלת הזמן הגילוי הגדול ביותר שאליה אפשר להגיע מ־u ב־G, אז $\phi(u)$ הוא אב קדמון של u. נתבונן בשני מקרים, תלוי בצבע שבו נצבע $\phi(u)$ כאשר u נצבע לראשונה. אם $\phi(u)$ נצבע לבן, אז הוא הופך לצאצא של u, ולכן $\phi(u) = \phi(\phi(u))$. אם $\phi(u)$ נצבע אפור, אז הוא הופך לצאצא של u בעץ הגילוי של DFS שבו u הוא השורש, ולכן $\phi(u) = \phi(\phi(u))$. בכל מקרה, $\phi(u)$ הוא אב קדמון של u.

מסקנה 23.15

בכל רכיב קשיר חיטב של גרף מכוון $G = (V, E)$, עבור כל $u \in V$, הקדקודים u ו־$\phi(u)$ שייכים לאותו רכיב קשיר חיטב.

תוכחה $u \rightsquigarrow \phi(u)$ על פי התגדרה של אב קדום ביותר, ו־$\phi(u) \rightsquigarrow u$ שכן $\phi(u)$ הוא אב קדמון של u. ■

המשפט הבא נותן תיאור מדויק של עץ העמק.

משפט 23.17

יהא $G = (V, E)$ גרף מכוון. נניח ש־DFS מופעל על G ומחשב את $f[u]$ לכל קדקוד $u \in V$. נניח שהרשימות השכנים נסרקות בסדר יורד של $f[u]$ בעת הפעלת DFS-$Visit$. אז G הוא עץ העמק של G.

תוכחה $\Leftarrow$: נניח כי $\phi(v) = \phi(u)$ עבור $u \neq v$. אז u ו־v נמצאים באותו רכיב קשיר חיטב של G (מסקנה 23.15). לכן, u ו־v נמצאים באותו עץ DFS (משפט 23.6), ו־u הוא אב קדמון של v או להפך. נניח בלי הגבלת הכלליות ש־u הוא אב קדמון של v. אז $f[u] > f[v]$ (משפט 23.8). אבל $\phi(u)$ הוא הפסגה בעלת הזמן הגילוי הגדול ביותר שאליה אפשר להגיע מ־u ב־G, ולכן $f[u] \leq f[\phi(u)]$. לכן, $f[\phi(u)] > f[v]$, ולכן קדקוד $\phi(u)$ נצבע לבן כאשר v מתגלה לראשונה. לפי משפט המקסלי (משפט 23.3), $\phi(u)$ הוא צאצא של v בעץ DFS. אבל אז $\phi(u)$ ו־v נמצאים באותו רכיב קשיר חיטב, בסתירה להנחה. לפיכך, $\phi(u) = \phi(v)$.

$\Rightarrow$: נניח כי $\phi(u) = \phi(v)$. אז u ו־v נמצאים באותו רכיב קשיר חיטב של G, ולכן $u \rightsquigarrow v$ ב־G. נותר להוכיח כי $v \rightsquigarrow u$ ב־G. מאחר ש־$\phi(u) = \phi(v)$, נובע כי $f[\phi(u)] = f[\phi(v)]$. נניח בלי הגבלת הכלליות ש־u הוא הראשון מבין קדקודי הרכיב הקשיר החיטב שמתגלה, ומקדקוד האחרון מבין קדקודי הרכיב הקשיר החיטב שמתגלים. בתיפוש לעומק של G^T, זהו השורש של עץ העמק. עתה נכיח כי שראה מיד, בכל רכיב קשיר חיטב יש קדקוד אחד שהוא האב הקדום ביותר של כל קדקוד אחר ברכיב. זה, אם קדום ביותר זה הוא "קדקוד נציג" של הרכיב הקשיר החיטב שבתיפוש לעומק של G, זהו הקדקוד הראשון מבין קדקודי הרכיב הקשיר החיטב שמתגלה, ומקדקוד האחרון מבין קדקודי הרכיב, זה, אם קדום ביותר זה הוא "קדקוד נציג" של הרכיב הקשיר החיטב שבתיפוש לעומק של G. תהיינה אלה.

תכונות אלה.

לשלמות, נוכיח גם משפט שנוכל להזדקק לו בהמשך.

משפט 23.18

בגרף מכוון $G = (V, E)$, יהא $\phi(u)$ הפסגה בעלת הזמן הגילוי הגדול ביותר של קדקוד $u \in V$ בלשונו בהגדרת העמק G. אז לכל קדקוד $v \in V$ שאליו אפשר להגיע מ־u ב־G, מתקיים $f[\phi(v)] \leq f[\phi(u)]$.

תוכחה אם $u = v$ אז $\phi(u) = \phi(v)$, ולכן $f[\phi(v)] = f[\phi(u)]$. אחרת, $u \neq v$ ו־v נצבע מ־u ב־G, ולכן $v \in \{w : u \rightsquigarrow w\}$. מכאן נובע כי $\{w : v \rightsquigarrow w\} \subseteq \{w : u \rightsquigarrow w\}$. מכאן נובע כי $f[\phi(v)] \leq f[\phi(u)]$ על פי אי־שוויון (23.3).

עתה, בשבילנו משפט 23.16 אפשר להבין בצורה קלה את מבנהו של האלגוריתם לחישוב רכיבים קשירים היטב. הרכיבים קשירים היטב הם קבוצות של קדקודים שכל אחת מהן שייכת לאותו רכיב קשיר חיטב. אלגוריתם $STRONGLY$-$CONNECTED$-$COMPONENTS$ מחשב בתחילה את $\phi(u)$ לכל קדקוד u ב־G. לאחר מכן הוא בונה את G^T ומחשב את $f[\phi(u)]$ לכל קדקוד u ב־G^T. לבסוף, הוא מבצע חיפוש לעומק על G^T תוך שהלולאה הראשית של DFS עוברת על הקדקודים בסדר יורד של $f[\phi(u)]$. כל עץ DFS שנוצר בשלב זה הוא רכיב קשיר חיטב. נכיח את נכונות האלגוריתם.

משפט 23.19

אלגוריתם $STRONGLY$-$CONNECTED$-$COMPONENTS$ מחשב בהלבה את הרכיבים הקשירים היטב של גרף מכוון G.

תוכחה נכיח את המשפט על ידי אינדוקציה על מספר העצים שנוצרים בעץ DFS שנוצר בשלב השני של האלגוריתם. נוכיח כי כל עץ DFS מייצג רכיב קשיר חיטב בנפרד. בשלב האינדוקציה, כאשר נוצר עץ DFS כלשהו, כל קדקוד v שאליו אפשר להגיע מהשורש של העץ נמצא ברכיב קשיר חיטב שאינו מכיל קדקודים שכבר נמצאו בעצים קודמים. מדוע? נניח ש־v נמצא ברכיב קשיר חיטב C, ושהקדקוד u שייך לרכיב קשיר חיטב שכבר נמצא. אז, לפי מסקנה 23.15, u ו־v נמצאים באותו רכיב קשיר חיטב. לפי משפט 23.18, $f[\phi(v)] \leq f[\phi(u)]$. אבל u הוא הקדקוד הראשון שנמצא ברכיב קשיר חיטב שבו נמצא v, ולכן $\phi(u) = \phi(v)$. לפיכך, u ו־v נמצאים באותו רכיב קשיר חיטב, בסתירה להנחה. לפיכך, כל קדקוד שאליו אפשר להגיע מ־u ב־G נמצא ברכיב קשיר חיטב שבו נמצא u.

■

הוכחה נוכיח באינדוקציה על מספר עצי העומק המתגלים בחיפוש לעומק של G^T כי הקדקודים בכל עץ יוצרים רכיב קשיר היטב. כל צעד בטיעון האינדוקטיבי מוכיח שעץ הנוצר בחיפוש לעומק של G^T הוא רכיב קשיר היטב, בהנחה שכל העצים שנוצרו לפניו הם רכיבים קשירים היטב. בסיס האינדוקציה טריוויאלי, שכן לעץ הראשון שנוצר אין עצים שנוצרו לפניו, ולכן ההנחה נכונה באופן טריוויאלי.

נתבונן בעץ עומק T בעל שורש r הנוצר בחיפוש לעומק של G^T. נסמן ב־$C(r)$ את קבוצת הקדקודים ש־r הוא האב הקדום ביותר שלהם:

$$C(r) = \{v \in V : \phi(v) = r\}$$

עתה נוכיח שקדקוד u מוצב ב־T אם ורק אם $u \in C(r)$.

$\Longleftarrow$: ממשפט 23.13 נובע שכל הקדקודים ב־$C(r)$ יימצאו בסופו של דבר באותו עץ עומק, מכיוון ש־$r \in C(r)$ ו־r הוא השורש של T, כל איבר ב־$C(r)$ יימצא בסופו של דבר ב־T.

$\Longrightarrow$: נראה כי כל קדקוד w שעבורו $f[\phi(w)] > f[r]$ או $f[\phi(w)] < f[r]$ אינו מוצב ב־T, על־ידי כך שנתבונן בנפרד בשני המקרים. באינדוקציה על מספר העצים המתגלים, כל קדקוד w שעבורו $f[\phi(w)] > f[r]$ אינו מוצב בעץ T, שכן בזמן ש־r נבחר, w כבר מוצב בעץ ששורשו $\phi(w)$. כל קדקוד w שעבורו $f[\phi(w)] < f[r]$ אינו יכול להיות מוצב ב־T, מכיוון שהצבה כזאת היתה גוררת $r \rightsquigarrow w$; לכן, על פי נוסחה (23.3) והתכונה כי $r = \phi(r)$, נקבל כי $f[\phi(w)] \geq f[\phi(r)] = f[r]$, בסתירה לכך ש־$f[\phi(w)] < f[r]$.

לכן, T מכיל בדיוק אותם קדקודים u שעבורם $\phi(u) = r$. כלומר, T שווה בדיוק לרכיב הקשיר היטב $C(r)$, ובכך נשלמה ההוכחה באינדוקציה. ■

תרגילים

23.5-1

כיצד יכול להשתנות מספר הרכיבים הקשירים היטב בגרף אם מוסיפים קשת חדשה?

23.5-2

הראה כיצד פועלת השגרה STRONGLY-CONNECTED-COMPONENTS על הגרף שבאיור 23.6. בפרט, הצג את מועדי הסיום המחושבים בשורה 1 ואת היער הנוצר בשורה 3. הנח שהלולאה בשורות 5-7 של DFS בוחנת קדקודים בסדר אלפביתי ושרשימות הסמיכות מסודרות בסדר אלפביתי.

23.5-3

פרופסור דיוור טוען שניתן לפשט את האלגוריתם למציאת רכיבים קשירים היטב על־ידי שימוש בגרף המקורי (ולא בגרף המוחלף) בחיפוש לעומק השני וסריקת הקדקודים בסדר *עולה* של זמני סיום. האם הפרופסור צודק?

23.5-4

את **גרף הרכיבים** (component graph) של גרף מכוון $G = (V, E)$ אנו מסמנים על־ידי $G^{SCC} = (V^{SCC}, E^{SCC})$, כאשר V^{SCC} מכילה קדקוד אחד עבור כל רכיב קשיר היטב ב־G ו־E^{SCC} מכילה את הקשת (u, v) אם מקדקוד ברכיב הקשיר היטב של G המתאים ל־u קיימת

קשת מכוונת לקדקוד ברכיב הקשיר היטב של G המתאים ל־v. באיור 23.9(iii) מוצגת דוגמה. הוכח כי G^{SCC} הוא גמ"ל.

23.5-5

כתוב אלגוריתם שזמן ריצתו $O(V + E)$ לחישוב גרף הרכיבים של גרף מכוון $G = (V, E)$. הבטח שתהיה לכל היותר קשת אחת בין שני קדקודים בגרף הרכיבים שיוצר האלגוריתם שלך.

23.5-6

בהינתן גרף מכוון $G = (V, E)$, הסבר כיצד ליצור גרף אחר $G' = (V, E')$ המקיים: (i) הרכיבים הקשירים היטב של G' זהים לאלה של G, (ii) גרף הרכיבים של G' זהה לזה של G, ו־(iii) E' קטנה ככל האפשר. תאר אלגוריתם מהיר ליצירת G'.

23.5-7

גרף מכוון $G = (V, E)$ נקרא **קשיר־למחצה** (semiconnected) אם, עבור כל זוגות הקדקודים $u, v \in V$, מתקיים $u \rightsquigarrow v$ או $v \rightsquigarrow u$. כתוב אלגוריתם יעיל הקובע אם G קשיר־למחצה או לא. הוכח את נכונותו של האלגוריתם שלך ונתח את זמן הריצה שלו.

בעיות

23-1 סיווג קשתות על־ידי חיפוש לרוחב

יער עומק מסווג את קשתותיו של גרף לקשתות עץ, קשתות אחורה, קשתות קדימה וקשתות חוצות. ניתן להשתמש גם בחיפוש לרוחב כדי לסווג את הקשתות שניתן להגיע אליהן ממקור החיפוש לאותן ארבע קטגוריות.

א. הוכח שבחיפוש לרוחב של גרף בלתי־מכוון מתקיימות התכונות הבאות:

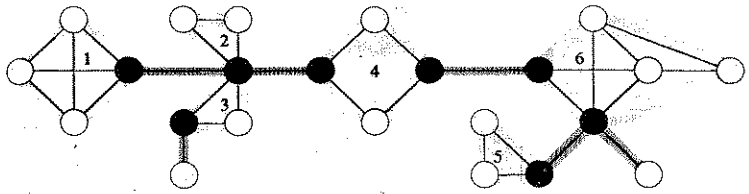
1. לא קיימות קשתות אחורה וקשתות קדימה.
2. לכל קשת עץ (u, v) מתקיים $d[v] = d[u] + 1$.
3. לכל קשת חוצה (u, v) מתקיים $d[v] = d[u]$ או $d[v] = d[u] + 1$.

ב. הוכח שבחיפוש לרוחב של גרף מכוון מתקיימות התכונות הבאות:

1. לא קיימות קשתות קדימה.
2. לכל קשת עץ (u, v) מתקיים $d[v] = d[u] + 1$.
3. לכל קשת חוצה (u, v) מתקיים $d[v] \leq d[u] + 1$.
4. לכל קשת אחורה (u, v) מתקיים $d[v] < d[u] \leq 0$.

23-2 נקודות הפרדה, גשרים, ורכיבים דו־קשירים

יהי $G = (V, E)$ גרף בלתי־מכוון קשיר. **נקודת הפרדה** (articulation point) של G היא קדקוד אשר סילוקו הופך את G לבלתי קשיר. **גשר** (bridge) ב־G הוא קשת אשר סילוקה הופך את G לבלתי קשיר. **רכיב דו־קשיר** (biconnected component) של G הוא קבוצה מקסימלית של קשתות כך שכל שתי קשתות בקבוצה נמצאות על מעגל פשוט משותף. איור 23.10 מדגים הגדרות אלה. ניתן לקבוע נקודות הפרדה, גשרים ורכיבים דו־קשירים באמצעות חיפוש לעומק.



איור 23.10 נקודות ההפרדה, הגשרים והרכיבים הדו־קשירים של גרף בלתי־מכוון קשיר, לשימוש בבעיה 23-2. נקודות ההפרדה הן הקדקודים המושחרים, הגשרים הם הקשתות עם הצללית הכהה, והרכיבים הדו־קשירים הם הקשתות באזורים המוצללים. האיור כולל מספור של הרכיבים הדו־קשירים.

יהי $G_\pi = (V, E_\pi)$ עץ עומק של G.

א. הוכח שהשורש של G_π הוא נקודת הפרדה של G אם ורק אם יש לו לפחות שני בנים ב־G_π.

ב. יהי v קדקוד ב־G_π שאינו השורש. הוכח ש־v הוא נקודת הפרדה של G אם ורק אם ל־v יש בן s כך שלא קיימת קשת אחורה מ־s או מצאצא כלשהו של s לאב קדמון ממש של v.

ג. נגדיר:

$$low[v] = \min \begin{cases} d[v], \\ \{d[w] : \text{היא קשת אחורה עבור איזשהו } u \text{ צאצא של } v \text{ } (u, w)\} \end{cases}$$

הראה כיצד ניתן לחשב את $low[v]$ עבור כל הקדקודים $v \in V$ בזמן $O(E)$.

ד. הראה כיצד ניתן לחשב את כל נקודות ההפרדה בזמן $O(E)$.

ה. הוכח שקשת ב־G היא גשר אם ורק אם היא אינה נמצאת על מעגל פשוט כלשהו ב־G.

ו. הראה כיצד ניתן לחשב את כל הגשרים ב־G בזמן $O(E)$.

ז. הוכח שהרכיבים הדו־קשירים של G יוצרים חלוקה של הקשתות ב־G שאינן גשרים.

ח. כתוב אלגוריתם שזמן ריצתו $O(E)$ אשר מסמן כל קשת e ב־G במספר שלם חיובי $bcc[e]$, כך ש־$bcc[e] = bcc[e']$ אם ורק אם e ו־e' שייכות לאותו רכיב דו־קשיר.

23-3 מסלול אוילר

מסלול אוילר (Euler tour) של גרף מכוון קשיר $G = (V, E)$ הוא מעגל העובר בכל קשת ב־G פעם אחת בדיוק, למרות שהוא עשוי לבקר בקדקוד יותר מפעם אחת.

א. הראה שב־G קיים מסלול אוילר אם ורק אם עבור כל קדקוד $v \in V$, דרגת הכניסה של v שווה לדרגת היציאה שלו.

ב. תאר אלגוריתם שזמן ריצתו $O(E)$ למציאת מסלול אוילר של G אם קיים כזה. (**רמז:** מזג מעגלים זרים־בקשתותיהם.)

הערות ביבליוגרפיות לפרק 23

Even [65] ו־Tarjan [188] הם מקורות מצוינים עבור אלגוריתמים על גרפים.

חיפוש לרוחב התגלה על־ידי Moore [150] בהקשר של מציאת מסלולים במבוכים. Lee [134] גילה אותו אלגוריתם באופן בלתי תלוי בהקשר של ניתוב תילים על לוחות של מעגלים חשמליים.

Hopcroft & Tarjan [102] הציגו את יתרונותיו של השימוש בייצוג על־ידי רשימות סמיכות על השימוש בייצוג על־ידי מטריצת סמיכויות עבור גרפים דלילים, והיו הראשונים שהכירו בחשיבות האלגוריתמית של חיפוש לעומק. חיפוש לעומק היה בשימוש נרחב מאז שנות החמישים המאוחרות, במיוחד בתכניות אינטליגנציה מלאכותית.

Tarjan [185] הציג אלגוריתם זמן־לינארי למציאת הרכיבים הקשירים היטב. האלגוריתם עבור רכיבים קשירים היטב שהובא בסעיף 23.5 נלקח מ־Aho, Hopcroft & Ullman [5] בשינויים מסוימים. מחברים אלה זוקפים אותו לזכותם של S.R. Kosaraju ו־M. Sharir. Knuth [121] היה הראשון שהציג אלגוריתם זמן־לינארי למיון טופולוגי.

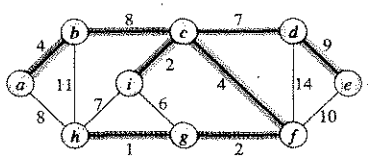
24 עצים פורשים מינימליים

בתכנון מעגלים חשמליים יש לעתים קרובות צורך לדאוג לכך שההדקים ("רגליים") של כמה רכיבים יהיו נקודות שקולות מבחינה חשמלית. כדי לחבר קבוצה של n הדקים, אפשר להשתמש בסידור של $n-1$ חוטי חשמל, או תילים, אשר כל אחד מהם מחבר שני הדקים. מבין כל הסידורים האפשריים האלה, הסידור הרצוי ביותר הוא בדרך כלל זה שבו משתמשים בכמות התיל הקטנה ביותר.

ניתן לייצג בעיית חיווט זו באמצעות גרף בלתי־מכוון קשיר $G = (V, E)$ כאשר V היא קבוצת ההדקים, E היא קבוצת החיבורים האפשריים בין זוגות הדקים, וכל קשת $(u, v) \in E$ היא בעלת משקל $w(u, v)$ המציין את העלות של חיבור u ו־v (דהיינו, את כמות התיל הנדרשת). אם כן, ברצוננו למצוא תת־קבוצה ללא מעגלים $T \subseteq E$ המחברת את כל הקדקודים ואשר משקלה הכולל:

$$w(T) = \sum_{(u,v) \in T} w(u, v)$$

הוא מינימלי. מכיוון ש־T מחברת את כל הקדקודים ואינה מכילה מעגלים, הרי שבהכרח היא יוצרת עץ, המכונה **עץ פורש** (spanning tree), שכן הוא "פרוש" על הגרף G כולו. את הבעיה של מציאת העץ T אנו מכנים **בעיית העץ הפורש המינימלי** (minimum-spanning-tree problem)[1]. איור 24.1 מביא דוגמה של גרף קשיר והעץ הפורש המינימלי שלו.



איור 24.1 עץ פורש מינימלי של גרף קשיר. ליד הקשתות רשומים המשקלות שלהן, והקשתות השייכות לעץ הפורש המינימלי מוצללות. המשקל הכולל של העץ המוצג הוא 37. העץ אינו יחיד: על־ידי החלפת הקשת (b, c) בקשת (a, h) מתקבל עץ פורש מינימלי נוסף שמשקלו 37.

בפרק זה נבחן שני אלגוריתמים הפותרים את בעיית העץ הפורש המינימלי: האלגוריתם של קרוסקל (Kruskal) והאלגוריתם של פרים (Prim). ניתן להגיע בנקל לכך שכל אחד מאלגוריתמים אלה ירוץ בזמן $O(E \lg V)$ על־ידי שימוש בערימות בינריות רגילות. על־ידי שימוש

1. המונח "עץ פורש מינימלי" הוא קיצור של המונח המדויק יותר "עץ פורש בעל משקל מינימלי". איננו ממיינים, למשל, את מספר הקשתות ב־T, שכן על פי משפט 5.2, כל העצים הפורשים מכילים בדיוק $|V| - 1$ קשתות.

בערימות פיבונאצ'י ניתן לקצר את זמן הריצה של האלגוריתם של פרים ל־$O(E + V \lg V)$, המהווה שיפור אם $|V|$ קטן בהרבה מ־$|E|$.

שני האלגוריתמים גם מדגימים היוריסטיקת אופטימיזציה הנקראת "הגישה החמדנית". בכל צעד באלגוריתם יש לבחור באחת מבין כמה אפשרויות. הגישה החמדנית מכתיבה בחירה באפשרות הטובה ביותר באותו רגע. ככלל, גישה זו אינה מבטיחה מציאתם של פתרונות אופטימליים כוללים לבעיות. אולם במקרה של בעיית העץ הפורש המינימלי ניתן להוכיח שאסטרטגיות חמדניות מסוימות אכן מניבות עץ פורש בעל משקל מינימלי. גישות חמדניות נדונות באריכות בפרק 17. ניתן אמנם לקרוא את הפרק הנוכחי באופן בלתי תלוי בפרק 17, אולם הגישות החמדניות המוצגות כאן מהוות יישום קלאסי של הרעיונות התיאורטיים שהוצגו שם.

סעיף 24.1 מציג אלגוריתם "כללי" למציאת עץ פורש מינימלי, אשר "מצמיח" עץ פורש על־ידי הוספת קשת אחת בכל פעם. סעיף 24.2 מציג שתי דרכים למימוש האלגוריתם הכללי. האלגוריתם הראשון, של קרוסקל, דומה לאלגוריתם למציאת רכיבים קשירים שהובא בסעיף 22.1. האלגוריתם השני, של פרים, דומה לאלגוריתם של דייקסטרה (Dijkstra) למציאת מסלולים קצרים ביותר (סעיף 25.2).

24.1 "הצמחת" עץ פורש מינימלי

נניח שנתונים לנו גרף בלתי־מכוון קשיר $G = (V, E)$ עם פונקציית משקל $w : E \to \mathbf{R}$, וברצוננו למצוא עץ פורש מינימלי עבור G. שני האלגוריתמים שנבחן בפרק זה משתמשים בגישה חמדנית לבעיה, אולם הם נבדלים זה מזה בדרך שבה הם מיישמים גישה זו.

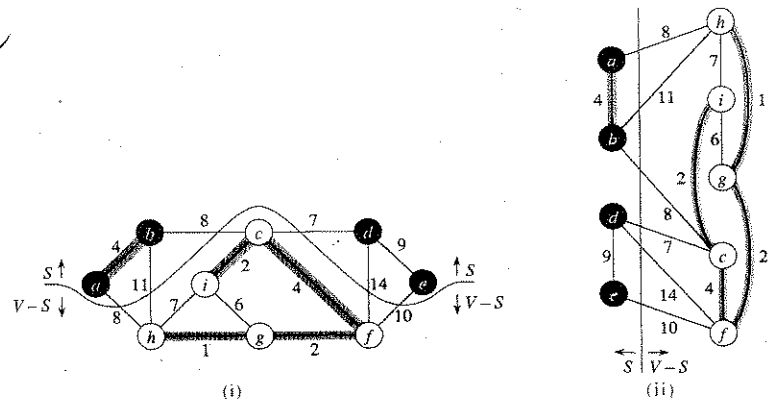
אסטרטגיה חמדנית זו עומדת בבסיסו של האלגוריתם ה"כללי" שלהלן, אשר "מצמיח" עץ פורש מינימלי על־ידי הוספת קשת אחת בכל פעם. האלגוריתם מנהל קבוצה A שהיא תמיד תת־קבוצה של עץ פורש מינימלי כלשהו. בכל צעד, האלגוריתם מוצא קשת (u, v) שניתן להוסיפה ל־A בלי להפר אינווריאנטה זו, במובן זה ש־$A \cup \{(u, v)\}$ גם היא תת־קבוצה של עץ פורש מינימלי. אנו קוראים לקשת כזאת **קשת בטוחה** (safe edge), שכן ניתן להוסיפה ל־A בבטחה בלי להפר את האינווריאנטה.

```
GENERIC-MST(G, w)
1  A ← φ
2  while A does not form a spanning tree
3      do find an edge (u, v) that is safe for A
4         A ← A ∪ {(u, v)}
5  return A
```

נשים לב לכך שלאחר שורה 1, הקבוצה A מקיימת באופן טריוויאלי את האינווריאנטה ש־A היא תת־קבוצה של עץ פורש מינימלי. הלולאה בשורות 2–4 שומרת על האינווריאנטה. לכן, כאשר הקבוצה A מוחזרת בשורה 5, A היא בהכרח עץ פורש מינימלי. החלק הקשה הוא, כמובן, מציאת קשת בטוחה בשורה 3. קשת כזאת בהכרח קיימת, שכן בעת ביצועה של שורה 3, האינווריאנטה מכתיבה את קיומו של עץ פורש T שעבורו $A \subseteq T$, ואם קיימת קשת $(u, v) \in T$ כך ש־$(u, v) \notin A$, אזי (u, v) בטוחה עבור A.

ביתרתו של סעיף זה נציג כלל (משפט 24.1) לזיהוי קשתות בטוחות. בסעיף הבא נתאר שני אלגוריתמים המשתמשים בכלל זה כדי למצוא ביעילות קשתות בטוחות.

תחילה דרושות לנו כמה הגדרות. **חתך** $(S, V-S)$ (cut) של גרף בלתי־מכוון $G=(V,E)$ הוא חלוקה של V. איור 24.2 מתאר מושג זה. אנו אומרים שקשת $(u,v)\in E$ **חוצה** את החתך $(S, V-S)$ אם אחת מנקודות הקצה שלה נמצאת ב־S והאחרת ב־$V-S$. אנו אומרים שחתך **מכבד** קבוצת קשתות A אם לא קיימת ב־A קשת החוצה את החתך. קשת היא **קשת קלה** החוצה חתך אם משקלה מינימלי מבין משקליהן של כל הקשתות החוצות את החתך. יש לשים לב שתיתכן יותר מקשת קלה אחת החוצה חתך, במקרה של שוויון משקלים. באופן כללי יותר, אנו אומרים שקשת היא **קשת קלה** המקיימת תכונה מסוימת אם משקלה מינימלי מבין משקליהן של כל הקשתות המקיימות את התכונה.



איור 24.2 שתי דרכים לראיית חתך $(S, V-S)$ של הגרף מאיור 24.1. **(i)** הקדקודים השחורים באיור שייכים לקבוצה S והלבנים שייכים ל־$V-S$. הקשתות החוצות את החתך הן אלה המחברות קדקודים לבנים עם קדקודים שחורים. הקשת (d,c) היא הקשת הקלה היחידה החוצה את החתך. תת־קבוצה A של הקשתות מוצללת באיור; אפשר לראות שהחתך $(S, V-S)$ מכבד את A, שכן שום קשת ב־A אינה חוצה את החתך. **(ii)** אותו גרף כאשר הקדקודים השייכים ל־S נמצאים משמאל והקדקודים השייכים ל־$V-S$ נמצאים מימין. קשת חוצה את החתך אם היא מחברת קדקוד המופיע משמאל עם קדקוד המופיע מימין.

הכלל שלנו לזיהוי קשתות בטוחות נתון במשפט הבא.

משפט 24.1

יהי $G=(V,E)$ גרף בלתי־מכוון קשיר עם פונקציית משקל w המוגדרת על E ומקבלת ערכים ממשיים. תהי A תת־קבוצה של E המוכלת בעץ פורש מינימלי כלשהו של G, יהי $(S, V-S)$ חתך כלשהו של G המכבד את A, ותהי (u,v) קשת קלה החוצה את $(S, V-S)$. אזי, הקשת (u,v) בטוחה עבור A.

הוכחה יהי T עץ פורש מינימלי המכיל את A, ונניח כי T אינו מכיל את הקשת הקלה (u,v), שכן אם הוא מכיל אותה, סיימנו. נבנה עץ פורש מינימלי אחר T' המכיל את $A\cup\{(u,v)\}$ על־ידי שימוש בטכניקה של "חיתוך והדבקה", ובכך נראה כי (u,v) היא קשת בטוחה עבור A.

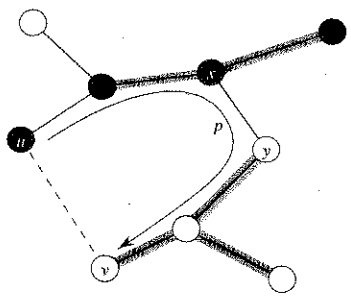
הקשת (u,v) יוצרת מעגל ביחד עם הקשתות על המסלול p מ־u ל־v ב־T, כמודגם באיור 24.3. מכיוון ש־u ו־v נמצאים משני צידי החתך $(S, V-S)$, קיימת לפחות קשת אחת ב־T על המסלול p החוצה גם היא את החתך. תהי (x,y) קשת כזאת. הקשת (x,y) אינה שייכת ל־A, שכן החתך מכבד את A. מאחר ש־(x,y) נמצאת על המסלול היחיד ב־T מ־u ל־v, סילוקה של (x,y) מפרק את T לשני רכיבים. הוספת (u,v) מחברת מחדש את שני הרכיבים לעץ פורש מינימלי חדש $T'=T-\{(x,y)\}\cup\{(u,v)\}$.

נראה עתה כי T' הוא עץ פורש מינימלי. מאחר ש־(u,v) היא קשת קלה החוצה את $(S, V-S)$ ו־(x,y) חוצה גם היא חתך זה, הרי שמתקיים $w(u,v)\le w(x,y)$. לכן:

$$\begin{aligned} w(T') &= w(T)-w(x,y)+w(u,v) \\ &\le w(T) \end{aligned}$$

אולם T הוא עץ פורש מינימלי, ולכן $w(T)\le w(T')$; לפיכך, T' חייב גם הוא להיות עץ פורש מינימלי.

נותר להראות כי (u,v) היא אכן קשת בטוחה עבור A. ידוע לנו כי $A\subseteq T'$, שכן $A\subseteq T$ ו־$(x,y)\notin A$; לכן, $A\cup\{(u,v)\}\subseteq T'$. כתוצאה מכך, מאחר ש־$T'$ הוא עץ פורש מינימלי, (u,v) בטוחה עבור A. ■



איור 24.3 הוכחת משפט 24.1. הקדקודים השייכים ל־S שחורים, והקדקודים השייכים ל־$V-S$ לבנים. הקשתות השייכות לעץ הפורש המינימלי T מופיעות באיור, אך הקשתות השייכות לגרף G אינן מופיעות בו. הקשתות השייכות ל־A מוצללות, ו־(u,v) היא קשת קלה החוצה את החתך $(S, V-S)$. הקשת (x,y) היא קשת על p, המסלול היחיד ב־T מ־u ל־v. עץ פורש מינימלי T' המכיל את (u,v) מתקבל על־ידי הוצאת הקשת (x,y) מ־T והוספת הקשת (u,v).

משפט 24.1 מבהיר לנו את פעולתו של האלגוריתם GENERIC-MST על גרף קשיר $G=(V,E)$. במהלך ביצועו של האלגוריתם, הקבוצה A היא תמיד חסרת מעגלים; אחרת, עץ פורש מינימלי המכיל את A היה מכיל מעגל, וזוהי סתירה. בכל נקודה במהלך ביצוע האלגוריתם, הגרף $G_A=(V,A)$ הוא יער, וכל אחד מן הרכיבים הקשירים של G_A הוא עץ. (אחדים מן העצים עשויים להכיל קדקוד אחד בלבד, כפי שקורה, למשל, עם תחילת ביצוע האלגוריתם: הקבוצה A ריקה והיער מכיל $|V|$ עצים, אחד עבור כל קדקוד.) יתירה מזו, כל קשת בטוחה (u,v) עבור A מחברת רכיבים זרים של G_A, שכן $A\cup\{(u,v)\}$ חייבת להיות חסרת מעגלים.

הלולאה בשורות 2-4 של GENERIC-MST מבוצעת $|V|-1$ פעמים, עם קביעתה של כל אחת מ־$|V|-1$ הקשתות של עץ פורש מינימלי. במצב ההתחלתי, כאשר $A=\phi$, G_A מורכב מ־$|V|$ עצים, וכל איטרציה מפחיתה ב־1 את מספר העצים. כאשר היער מכיל עץ אחד בלבד, האלגוריתם עוצר.

שני האלגוריתמים שיובאו בסעיף 24.2 עושים שימוש במסקנה הבאה הנובעת ממשפט 24.1.

מסקנה 24.2

יהי $G=(V,E)$ גרף בלתי־מכוון קשיר עם פונקציית משקל w המוגדרת על E ומקבלת ערכים ממשיים. תהי A תת־קבוצה של E המוכלת בעץ פורש מינימלי כלשהו של G, ויהי C רכיב קשיר

(עץ) ביער $G_A = (V, A)$. אם (u, v) היא קשת קלה המחברת את C לרכיב אחר כלשהו ב־G_A, אזי (u, v) בטוחה עבור A.

הוכחה החתך $(C, V - C)$ מכבד את A, ולכן (u, v) היא קשת קלה עבור חתך זה. ■

תרגילים

24.1-1

תהי (u, v) קשת בעלת משקל מינימלי בגרף G. הראה כי (u, v) שייכת לעץ פורש מינימלי כלשהו של G.

24.1-2

פרופסור סבטייה טוען לקיומו של המשפט ההפוך למשפט 24.1: יהי $G = (V, E)$ גרף בלתי־מכוון קשיר עם פונקציית משקל w המוגדרת על E ומקבלת ערכים ממשיים. תהי A תת־קבוצה של E המוכלת בעץ פורש מינימלי כלשהו של G, יהי $(S, V - S)$ חתך כלשהו של G המכבד את A, ותהי (u, v) קשת בטוחה עבור A החוצה את $(S, V - S)$. אזי, (u, v) היא קשת קלה עבור החתך. הראה שטענתו של הפרופסור אינה נכונה על־ידי מציאת דוגמה נגדית.

24.1-3

הראה שאם קשת (u, v) מוכלת בעץ פורש מינימלי כלשהו, אזי היא קשת קלה החוצה חתך כלשהו של הגרף.

24.1-4

הבא דוגמה פשוטה לגרף שבו קבוצת כל הקשתות שהן קשתות קלות החוצות חתך כלשהו של הגרף אינה יוצרת עץ פורש מינימלי.

24.1-5

תהי e קשת בעלת משקל מקסימלי במעגל כלשהו בגרף $G = (V, E)$. הוכח שקיים עץ פורש מינימלי של $G' = (V, E - \{e\})$ שהוא גם עץ פורש מינימלי של G.

24.1-6

הראה שגרף הוא בעל עץ פורש מינימלי יחיד אם, עבור כל חתך של הגרף, קיימת קשת קלה יחידה החוצה את החתך. הראה שההיפך אינו נכון על־ידי מציאת דוגמה נגדית.

24.1-7

הראה שאם כל המשקלות של קשתות גרף הם חיוביים, אזי כל תת־קבוצה של קשתות המחברת את כל הקדקודים ואשר משקלה הכולל מינימלי היא בהכרח עץ. מצא דוגמה המראה שמסקנה זו אינה נובעת אם אנו מרשים שחלק מן המשקלות יהיו אי־חיוביים.

24.1-8

יהי T עץ פורש מינימלי של גרף G, ותהי L הרשימה הממוינת של משקלות הקשתות של T. הראה שעבור כל עץ פורש מינימלי אחר T' של G, הרשימה L היא גם הרשימה הממוינת של משקלות הקשתות של T'.

24.1-9

יהי T עץ פורש מינימלי של גרף $G = (V, E)$, ותהי V' תת־קבוצה של V. יהי T' התת־גרף של T המושרה על־ידי V', ויהי G' התת־גרף של G המושרה על־ידי V'. הראה שאם T' קשיר, אזי T' הוא עץ פורש מינימלי של G'.

24.2 האלגוריתמים של קרוסקל ושל פרים

שני האלגוריתמים למציאת עץ פורש מינימלי שנתאר בסעיף זה הם עיבודים של האלגוריתם הכללי. כל אחד מהם משתמש בכלל מסוים למציאת קשת בטוחה בשורה 3 של GENERIC-MST. באלגוריתם של קרוסקל, הקבוצה A היא יער. הקשת הבטוחה המצורפת ל־A היא תמיד קשת בעלת משקל מינימלי בגרף אשר מחברת שני רכיבים זרים. באלגוריתם של פרים, הקבוצה A מהווה עץ יחיד. הקשת הבטוחה המצורפת ל־A היא תמיד קשת בעלת משקל מינימלי המחברת את העץ לקדקוד שאינו שייך לעץ.

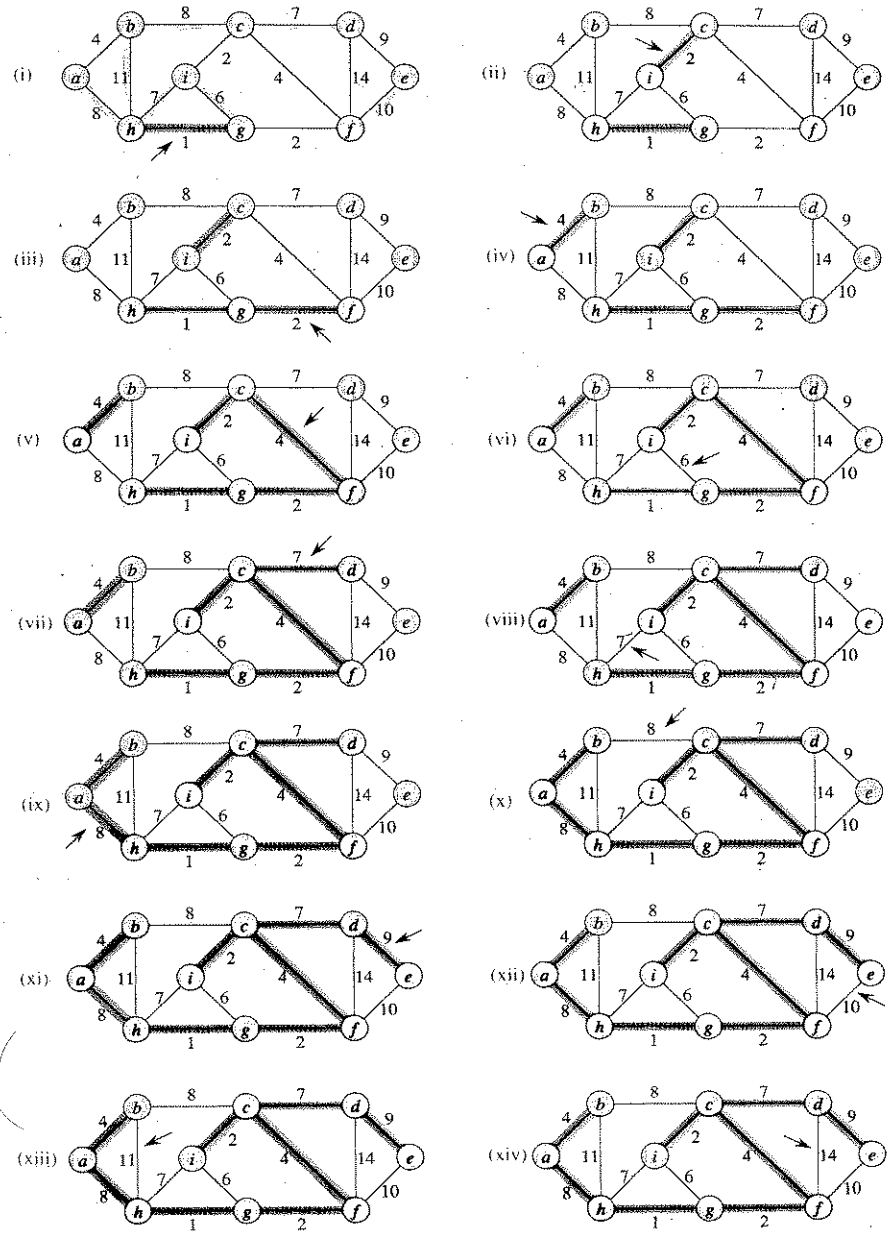
האלגוריתם של קרוסקל (Kruskal)

האלגוריתם של קרוסקל מבוסס ישירות על האלגוריתם הכללי למציאת עץ פורש מינימלי שהוצג בסעיף 24.1. הוא מוצא קשת בטוחה שניתן להוסיפה ליער המתהווה על־ידי כך שהוא מוצא קשת (u, v) בעלת משקל מינימלי מבין כל הקשתות המחברות שני עצים כלשהם ביער. נסמן ב־C_1 וב־C_2 את שני העצים שמחברת הקשת (u, v). מכיוון ש־(u, v) היא בהכרח קשת קלה המחברת את C_1 לעץ אחר כלשהו, הרי שממסקנה 24.2 נובע כי (u, v) היא קשת בטוחה עבור C_1. האלגוריתם של קרוסקל הוא אלגוריתם חמדן, משום שבכל צעד הוא מוסיף ליער קשת בעלת משקל קטן ככל האפשר.

המימוש שלנו לאלגוריתם של קרוסקל דומה לאלגוריתם למציאת רכיבים קשירים מסעיף 22.1. הוא משתמש במבנה נתונים לקבוצות זרות כדי לתחזק כמה קבוצות זרות של איברים. כל קבוצה מכילה את קדקודיו של עץ אחד ביער הנוכחי. הפעולה FIND-SET(u) מחזירה איבר נציג מן הקבוצה המכילה את u. באופן זה ניתן לקבוע אם שני קדקודים u ו־v שייכים לאותו עץ על־ידי כך שבודקים אם FIND-SET(u) שווה ל־FIND-SET(v). צירוף העצים מתבצע על־ידי השגרה UNION.

```
MST-KRUSKAL(G, w)
 1  A ← φ
 2  for each vertex v ∈ V[G]
 3      do MAKE-SET(v)
 4  sort the edges of E by nondecreasing weight w
 5  for each edge (u, v) ∈ E, in order by nondecreasing weight
 6      do if FIND-SET(u) ≠ FIND-SET(v)
 7            then A ← A ∪ {(u, v)}
 8                 UNION(u, v)
 9  return A
```

האלגוריתם של קרוסקל פועל כמתואר באיור 24.4. שורות 1-3 מאתחלות את הקבוצה A לקבוצה הריקה ויוצרות $|V|$ עצים אשר כל אחד מהם מכיל קדקוד אחד. שורה 4 ממיינת את הקשתות ב־E לפי סדר לא יורד של משקלות. לולאת ה־**for** בשורות 5-8 בודקת, עבור כל קשת (u, v), אם הקדקודים בקצותיה, u ו־v, שייכים לאותו עץ. אם אכן כך הדבר, אזי לא ניתן להוסיף את הקשת (u, v) ליער בלי ליצור מעגל, ולפיכך אין משתמשים בה. אחרת, שני הקדקודים שייכים לעצים שונים, ואז מוסיפים את הקשת (u, v) ל־A בשורה 7, ובשורה 8 מתבצע מיזוג של הקדקודים שבשני העצים.



איור 24.4 הרצת האלגוריתם של קרוסקל על הגרף מאיור 24.1. קשתות מוצללות שייכות ליער המתהווה A. הקשתות נבחנות על־ידי האלגוריתם על פי סדר המשקלות שלהן. החץ מצביע על הקשת הנבחנת הנוכחית בכל צעד של האלגוריתם. אם הקשת מחברת שני עצים זרים ביער, היא מצורפת אל היער, ובכך ממוזגים שני העצים.

זמן הריצה של האלגוריתם של קרוסקל על גרף $G = (V, E)$ תלוי במימוש מבנה הנתונים לקבוצות זרות. אנו נניח מימוש על־ידי יער הקבוצות הזרות שהוצג בסעיף 22.3 עם ההיוריסטיקות של איחוד על פי דרגה וכיווץ מסלולים, מאחר שזהו המימוש המהיר ביותר אסימפטוטית הידוע היום. האתחול מתבצע בזמן $O(V)$, והזמן הדרוש למיון הקשתות בשורה 4 הוא $O(E \lg E)$. ישנן $O(E)$ פעולות על יער הקבוצות הזרות, והזמן הכולל לביצוען הוא $O(E\,\alpha(E, V))$, כאשר α היא הפונקציה ההופכית לפונקציית אקרמן שהוגדרה בסעיף 22.4. מאחר שמתקיים $\alpha(E, V) = O(\lg E)$, זמן הריצה הכולל של האלגוריתם של קרוסקל הוא $O(E \lg E)$.

האלגוריתם של פרים (Prim)

כמו האלגוריתם של קרוסקל, גם האלגוריתם של פרים הוא מקרה פרטי של האלגוריתם הכללי למציאת עץ פורש מינימלי מסעיף 24.1. האלגוריתם של פרים פועל במידה רבה כמו האלגוריתם של דייקסטרה למציאת מסלולים קצרים ביותר בגרף (ראה סעיף 25.2). האלגוריתם של פרים הוא בעל התכונה שהקשתות בקבוצה A יוצרות תמיד עץ יחיד. כמודגם באיור 24.5, העץ מתחיל מקדקוד שורש שרירותי r, והוא הולך וגדל עד שהוא פורש את כל קדקודי V. בכל צעד מוסיפים לעץ A קשת קלה המחברת את A עם קדקוד מבודד ב־$G_A = (V, A)$. על פי מסקנה 24.2, כלל זה מבטיח שנוסיף ל־A רק קשתות שהינן בטוחות עבור A; לכן, כאשר האלגוריתם עוצר, הקשתות ב־A יוצרות עץ פורש מינימלי. אסטרטגיה זו היא "חמדנית", שכן בכל צעד מוסיפים לעץ קשת התורמת את המינימום האפשרי למשקלו.

המפתח למימוש יעיל של האלגוריתם של פרים טמון במציאת דרך קלה לבחירת הקשת החדשה שתצורף לעץ שיוצרות הקשתות ב־A. בפסאודו־קוד שלהלן, האלגוריתם מקבל כקלט את הגרף הקשיר G ואת השורש r של העץ הפורש המינימלי שהאלגוריתם צריך לבנות. במהלך ביצוע האלגוריתם, כל הקדקודים אשר אינם שייכים לעץ נשמרים בתור קדימויות Q המבוסס על שדה מפתח (key). עבור כל קדקוד v, $key[v]$ מכיל את המשקל המינימלי מבין משקלי הקשתות המחברות את v לקדקודים השייכים לעץ; כמקובל, נציב $key[v] = \infty$ אם לא קיימת קשת כזו. השדה $\pi[v]$ מזהה את ה"אב" של v בעץ. במהלך ביצוע האלגוריתם, הקבוצה A המופיעה ב־GENERIC-MST מנוהלת במובלע כקבוצה:

$$A = \{(v, \pi[v]) : v \in V - \{r\} - Q\}$$

כאשר האלגוריתם עוצר, תור הקדימויות Q ריק; העץ הפורש המינימלי A עבור G הוא אפוא:

$$A = \{(v, \pi[v]) : v \in V - \{r\}\}$$

```
MST-PRIM(G, w, r)
 1  Q ← V[G]
 2  for each u ∈ Q
 3      do key[u] ← ∞
 4  key[r] ← 0
 5  π[r] ← NIL
 6  while Q ≠ φ
 7      do u ← EXTRACT-MIN(Q)
 8         for each v ∈ Adj[u]
 9             do if v ∈ Q and w(u, v) < key[v]
10                   then π[v] ← u
11                        key[v] ← w(u, v)
```

איור 24.5 הדגמת האלגוריתם של דייקסטרה על הגרף מאייר 24.1. השורש הוא הקדקוד a. קשתות מצללות מציינות ערכים $\pi[v]$ עבור קדקודים v שנמצאים עדיין ב־Q. בכל שלב נבחר הקדקוד שבתוך העיגול ומוסר מ־Q. הקדקודים שבתוך העיגולים השחורים אינם ב־Q לאחר הסרת הקדקוד שלהם.

נניח שמשקלות של כל הקשתות בגרף הם מספרים שלמים בין 1 ל־W עבור ערך W כלשהו. נוכל אז להשתמש במבנה נתונים שנקרא *דלי* (bucket) כדי לשפר את זמן הריצה של האלגוריתם של דייקסטרה. מבנה הנתונים מכיל רשימות מקושרות של קדקודים, כאשר כל רשימה מכילה את כל הקדקודים שהמפתח שלהם הוא מספר מסוים. הדלי i מכיל את כל הקדקודים שהמפתח שלהם הוא i. הדלי הראשון מכיל את כל הקדקודים שהמפתח שלהם הוא 0, הדלי הבא מכיל את כל הקדקודים שהמפתח שלהם הוא 1, וכן הלאה. המספר המרבי של דלאים שאנחנו צריכים הוא $W(V-1)+1$, שכן זה המספר המרבי האפשרי של מרחקים שונים מהשורש, כאשר כל מרחק הוא לכל היותר $W(V-1)$.

24.2-5

נניח שמשקלות של כל הקשתות בגרף הם מספרים שלמים בין 1 ל־$|V|$. במבנה תוכל לקצר את זמן הריצה עבור W כלשהו.

24.2-4

פיבונאצי יהיה מהיר יותר מאשר אסימפטוטית אלגוריתם שמשתמש בערימה בינארית? עבור $|E| = \Theta(V)$ איזה ניחוש ערוך לטובת בין $|E|$ ל־$|V|$ כדי שמשתמש בערימה פיבונאצי יהיה מהיר יותר מאשר אסימפטוטית אלגוריתם שמשתמש בערימה בינארית? עבור $|E| = \Theta(V)$ ומה בדבר גרף צפוף, שבו $|E| = \Theta(V^2)$?

24.2-3

האם מימוש האלגוריתם של פרים באמצעות ערימת פיבונאצי מהיר יותר אסימפטוטית ממימוש באמצעות ערימה בינארית עבור גרפים דלילים $G = (V, E)$, שבו $|E| = \Theta(V)$? ומה בדבר גרף צפוף, שבו $|E| = \Theta(V^2)$?

24.2-2

נניח שהתחלנו לייצג גרף $G = (V, E)$ שעבורו G מייצג מסלול פשוט באורך $|V| - 1$ שמכיל את כל הקדקודים של V, אך E מכיל קשתות נוספות. תארו מימוש יעיל של פרים שמשתמש בייצוג זה.

24.2-1

הוכיחו שהאלגוריתם של דייקסטרה פועל נכון גם כאשר קשתות עם משקל שלילי מותרות, בתנאי שאין מעגל בעל משקל שלילי המקשר בין השורש לבין קדקוד כלשהו שנמצא ב־Q.

תרגילים

השווה אסימפטוטית ביצועי האלגוריתם של פרים ל־$O(E + V \lg V)$ שהתקבל לאלגוריתם של קרוסקל.

הוכיחו שזמן הריצה הכולל של האלגוריתם של פרים הוא $O(E \lg V)$, ותנו זמן ריצה של האלגוריתם של פרים במונחים של $O(\lg V)$ עבור כל אחת מהפעולות שמבוצעות על הערימה. ופעולה זו נקראת DECREASE-KEY. אם כך, זמן הריצה הכולל של האלגוריתם של פרים הוא $O(E \lg V + E \lg V) = O(E \lg V)$, ותנו זמן משוערך של $O(\lg V)$ עבור כל אחת מהפעולות שמבוצעות על הערימה. ופעולה זו נקראת EXTRACT-MIN. אם כך, זמן הריצה הכולל של האלגוריתם של פרים הוא $O(E \lg V + V \lg V) = O(E \lg V)$.

בשורות 8-11 מבוצעת $O(E)$ פעמים בסך הכל, שכן סכום אורכי כל הרשימות המקושרות הוא $|E|$. בסך הכל נדרשת $O(\lg V)$ עבודה עבור כל קדקוד שנוסף ל־Q בשורה 9 ועבור כל קדקוד שמוסר מ־Q בשורה 11, כך שסך הכל נדרשת $O(E \lg V)$ עבודה עבור פעולות Q. הזמן הכולל לביצוע כל הקריאות ל־EXTRACT-MIN הוא $O(V \lg V)$. לפיכך זמן הריצה הכולל של האלגוריתם הוא $O(E \lg V)$.

בהינתן גרף צפוף, שבו $|E| = \Theta(V^2)$, ניתן לבצע את פעולות האלגוריתם בזמן $O(V^2)$ בעזרת ערימה פשוטה שבה האיבר ה־u הוא פשוט הקדקוד u. כל פעולת EXTRACT-MIN דורשת זמן $O(V)$ כדי לחפש את האיבר המינימלי, ופעולת DECREASE-KEY דורשת זמן $O(1)$, כך שהזמן הכולל הוא $O(V^2)$. בגרפים דלילים ומאזנים שבהם $|E| = \Omega(V \lg V)$, המימוש המבוסס ערימה בינארית של האלגוריתם של פרים רץ מהר יותר מאשר המימוש המבוסס מערך. בגרפים גדולים יותר שבהם $|E| = \Theta(V^2/\lg V)$, המימוש המבוסס ערימת פיבונאצ'י שנדון בפרק 21 מהיר יותר מאשר המימוש המבוסס ערימה בינארית. אולם ניתן למצוא גרפים שעבורם המימוש המבוסס ערימת פיבונאצ'י איטי יותר מהמימוש המבוסס ערימה בינארית.

24.2-6

תאר אלגוריתם יעיל, אשר בהינתן לו גרף בלתי־מכוון G, מוצא עץ פורש של G אשר המשקל המקסימלי של קשת בו הוא מינימלי על פני כל העצים הפורשים של G.

* 24.2-7

נניח שמשקלות הקשתות בגרף מתפלגים באופן אחיד על פני הקטע הפתוח למחצה $[0, 1)$. את זמן הריצה של איזה אלגוריתם, של קרוסקל או של פרים, תוכל לשפר?

* 24.2-8

נניח שכבר בנינו עץ פורש מינימלי של גרף G. באיזו מהירות ניתן לעדכן את העץ הפורש המינימלי אם מצרפים ל־G קדקוד חדש וקשתות הקשורות אליו?

בעיות

24-1 עץ פורש בעל משקל מינימלי "שני־הכי־טוב"

יהי $G = (V, E)$ גרף בלתי־מכוון קשיר עם פונקציית משקל $w : E \to \mathbf{R}$, ונניח כי $|E| \geq |V|$.

א. יהי T עץ פורש מינימלי של G. הוכח שקיימות קשתות $(u, v) \in T$ ו־$(x, y) \notin T$ כך ש־$T - \{(u, v)\} \cup \{(x, y)\}$ הוא עץ פורש מינימלי "שני־הכי־טוב" של G, כלומר, שמשקלו הוא השני הקטן ביותר מבין העצים הפורשים של G.

ב. יהי T עץ פורש של G. עבור שני קדקודים כלשהם $u, v \in V$, תהי $max[u, v]$ קשת בעלת משקל מקסימלי על המסלול היחיד ב־T בין u ו־v. תאר אלגוריתם שזמן ריצתו $O(V^2)$ אשר, בהינתן T, מחשב את $max[u, v]$ עבור כל הזוגות $u, v \in V$.

ג. כתוב אלגוריתם יעיל לחישוב עץ פורש מינימלי "שני־הכי־טוב" של G.

24-2 עץ פורש מינימלי של גרף דליל

עבור כל גרף קשיר דליל $G = (V, E)$, ניתן לשפר את זמן הריצה $O(E + V \lg V)$ של האלגוריתם של פרים המומש באמצעות ערימות פיבונצ'י על־ידי ביצוע "עיבוד מקדים" על G, כדי להקטין את מספר הקדקודים לפני הרצת האלגוריתם של פרים. השגרה שלהלן מקבלת כקלט גרף משוקלל G ומחזירה גרסה "מכווצת" של G, לאחר שהוסיפה כמה קשתות ל־T, העץ הפורש המינימלי הנבנה. במצב ההתחלתי, עבור כל קשת $(u, v) \in E$, אנו מניחים כי $orig[u, v] = (u, v)$ וכי $w[u, v]$ הוא משקל הקשת.

```
MST-REDUCE(G, T)
 1  for each v ∈ V[G]
 2      do mark[v] ← FALSE
 3         MAKE-SET(v)
 4  for each u ∈ V[G]
 5      do if mark[u] = FALSE
 6            then choose v ∈ Adj[u] such that w[u, v] is minimized
 7                 UNION(u, v)
 8                 T ← T ∪ {orig[u, v]}
 9                 mark[u] ← mark[v] ← TRUE
10  V[G'] ← {FIND-SET(v) : v ∈ V[G]}
11  E[G'] ← φ
12  for each (x, y) ∈ E[G]
```



```
13      do u ← FIND-SET(x)
14         v ← FIND-SET(y)
15         if (u, v) ∉ E[G']
16            then E[G'] ← E[G'] ∪ {(u, v)}
17                 orig[u, v] ← orig[x, y]
18                 w[u, v] ← w[x, y]
19            else if w[x, y] < w[u, v]
20                    then orig[u, v] ← orig[x, y]
21                         w[u, v] ← w[x, y]
22  construct adjacency lists Adj for G'
23  return G' and T
```

א. תהי T קבוצת הקשתות שמחזירה MST-REDUCE, ויהי T' עץ פורש מינימלי של הגרף G' שמחזירה השגרה. הוכח כי $T \cup \{orig[x, y] : (x, y) \in T'\}$ הוא עץ פורש מינימלי של G.

ב. הראה כי $|V[G']| \leq |V|/2$.

ג. הראה כיצד ניתן לממש את MST-REDUCE כך שתרוץ בזמן $O(E)$. (**רמז:** השתמש במבני נתונים פשוטים.)

ד. נניח שמריצים את MST-REDUCE k פעמים, כאשר הגרף הנוצר על־ידי הרצה אחת משמש כקלט להרצה הבאה וב־T מצטברות קשתות. הראה כי זמן הריצה הכולל של k ההרצות הוא $O(kE)$.

ה. נניח שלאחר הרצת MST-REDUCE k פעמים, אנו מריצים את האלגוריתם של פרים על הגרף שהתזירה ההרצה האחרונה. הראה כיצד לבחור את k כך שזמן הריצה הכולל יהיה $O(E \lg \lg V)$. הראה כי ה־k שבחרת ממזער את זמן הריצה האסימפטוטי הכולל.

ו. עבור אילו ערכים של $|E|$ (במונחי $|V|$) זמן הריצה של האלגוריתם של פרים עם עיבוד מקדים טוב יותר אסימפטוטית מזה של האלגוריתם של פרים ללא עיבוד מקדים?

הערות ביבליוגרפיות לפרק 24

Tarjan [188] סוקר את בעיית העץ הפורש המינימלי ומציג חומר מתקדם מצוין. Graham & Hell [92] כתבו היסטוריה של בעיית העץ הפורש המינימלי.

Tarjan מייחס את האלגוריתם הראשון למציאת עץ פורש מינימלי למאמר של O. Boruvka משנת 1926. האלגוריתם של קרוסל הוצג על־ידיו ב־1956 (Kruskal [131]). את האלגוריתם הידוע כאלגוריתם של פרים אכן המציא פרים (Prim [163]), אולם הוא הומצא גם קודם לכן, ב־1930, על־ידי V. Jarník.

הסיבה ליעילותם של אלגוריתמים חמדנים במציאת עצים פורשים מינימליים היא העובדה שקבוצת היערות של גרף יוצרת מטרואיד גרפי (ראה סעיף 17.4).

האלגוריתם המהיר ביותר כיום למציאת עץ פורש מינימלי כאשר $|E| = \Omega(V \lg V)$ הוא האלגוריתם של פרים המומש באמצעות ערימות פיבונצ'י. קיים גם אלגוריתם של Gabow, Galil, Spencer & Tarjan משנת 1986, שזמן ריצתו $O(E \lg \beta(|E|, |V|))$, כאשר $\beta(|E|, |V|) = \min\{i : \lg^{(i)} |V| \leq |E|/|V|\}$. אלגוריתם זה משפר את זמן הריצה במקרים של גרפים דלילים יותר, למשל, אם $|E| \leq |V|$.

25 מסלולים קצרים ביותר ממקור יחיד

נהג רוצה למצוא את הדרך הקצרה ביותר משיקגו לבוסטון. כאשר נתונה לנו מפת דרכים של ארה"ב, שעליה רשומים המרחקים בין כל זוג הצטלבויות דרכים סמוכות, כיצד נוכל לקבוע איזהו נתיב הנסיעה הקצר ביותר?

אחת הדרכים האפשריות היא להכין רשימה של כל נתיבי הנסיעה משיקגו לבוסטון, לחשב את אורכו של כל נתיב, ולבחור את הקצר מכולם. אולם אפשר לראות מיד שגם אם איננו מרשים נתיבים המכילים מעגלים, קיימים מיליוני נתיבים אפשריים, שאת רובם אין כלל טעם לבחון. לדוגמה, נתיב היוצא משיקגו, עובר דרך יוסטון ומגיע לבוסטון הוא ללא ספק בחירה גרועה, משום שיוסטון שוכנת כאלף מייל מחוץ לדרך.

בפרק זה ובפרק 26 נראה כיצד ניתן לפתור ביעילות בעיות מסוג זה. **בעיה של מסלולים קצרים ביותר** (shortest-paths problem), נתונים לנו גרף מכוון משוקלל $G = (V, E)$ עם פונקציית משקל $w : E \to \mathbf{R}$ הממפה קשתות אל משקלות בעלי ערכים ממשיים. **המשקל** (weight) של מסלול $p = \langle v_0, v_1, \ldots, v_k \rangle$ הוא סכום משקלות הקשתות המרכיבות את המסלול:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

נגדיר **משקל של מסלול קצר ביותר** (shortest-path weight) מ־u אל v על־ידי:

$$\delta(u, v) = \begin{cases} \min\{ w(p) : u \overset{p}{\rightsquigarrow} v \} & \text{אם קיים מסלול מ-} u \text{ אל } v \\ \infty & \text{אחרת} \end{cases}$$

מסלול קצר ביותר (shortest path) מקדקוד u אל קדקוד v מוגדר אז כמסלול p כלשהו שמשקלו $w(p) = \delta(u, v)$.

בדוגמת הנתיב משיקגו לבוסטון, ניתן לייצג את מפת הדרכים באמצעות גרף: קדקודים בגרף מייצגים הצטלבויות דרכים, קשתות מייצגות קטעי דרך בין הצטלבויות, ומשקלות הקשתות מייצגים מרחקים. משימתנו היא למצוא מסלול קצר ביותר מהצטלבות דרכים נתונה בשיקגו להצטלבות דרכים נתונה בבוסטון.

את משקלות הקשתות ניתן לפרש לא רק כמרחקים אלא גם כמטריקות אחרות. משקלות הקשתות משמשים לעתים קרובות לייצוג זמן, עלות, קנסות, הפסדים, או כל כמות אחרת אשר מצטברת באופן לינארי לאורך מסלול וברצוננו למזער אותה.

אלגוריתם החיפוש לרוחב מסעיף 23.2 הוא אלגוריתם למציאת מסלולים קצרים ביותר בגרפים לא־משוקללים, כלומר, בגרפים שבהם ניתן לחשוב על כל קשת כאילו היא בעלת משקל של יחידה. מכיוון שרבים מן המושגים שהצגנו כאשר עסקנו בחיפוש לרוחב מופיעים גם בחקר מסלולים קצרים ביותר בגרפים משוקללים, מומלץ שהקורא יחזור ויעיין בסעיף 23.2 קודם שימשיך בקריאת פרק זה.

גרסאות שונות

בפרק זה נתמקד בבעיה של **מציאת מסלולים קצרים ביותר ממקור יחיד** (single-source shortest-paths problem): בהינתן גרף $G = (V, E)$, ברצוננו למצוא מסלול קצר ביותר מקדקוד **מקור** (source) נתון $s \in V$ לכל קדקוד $v \in V$. באמצעות האלגוריתם עבור גרסת המקור היחיד ניתן לפתור בעיות רבות אחרות, כולל הגרסאות הבאות לבעיית המסלולים הקצרים ביותר:

הבעיה של מסלולים קצרים ביותר ליעד יחיד (single-destination shortest-paths problem): מציאת מסלול קצר ביותר מכל קדקוד v אל קדקוד **יעד** (destination) נתון t. על־ידי היפוך כיוונה של כל קשת בגרף, ניתן להפוך בעיה זו לבעיה של מציאת מסלולים קצרים ביותר ממקור יחיד.

הבעיה של מסלול קצר ביותר בין זוג קדקודים אחד (single-pair shortest-path problem): מציאת מסלול קצר ביותר מקדקוד נתון u אל קדקוד נתון v. אם נפתור את בעיית המקור היחיד עם u כמקור, נפתור בכך גם בעיה זו. יתירה מזו, לא ידועים היום אלגוריתמים לבעיה זו שזמן ריצתם במקרה הגרוע מהיר יותר אסימפטוטית מזה של האלגוריתמים הטובים ביותר עבור גרסת המקור היחיד.

הבעיה של מסלולים קצרים ביותר בין כל זוגות הקדקודים (all-pairs shortest-paths problem): מציאת מסלול קצר ביותר מ־u אל v, עבור כל זוג קדקודים u ו־v. בעיה זו ניתן לפתור על־ידי הרצת האלגוריתם עבור בעיית המקור היחיד פעם אחת עבור כל קדקוד כמקור; אולם בדרך כלל ניתן לפתור בעיה זו מהר יותר, ומבנה הבעיה מעניין כשלעצמו. בפרק 26 נטפל בפירוט בגרסת כל זוגות הקדקודים.

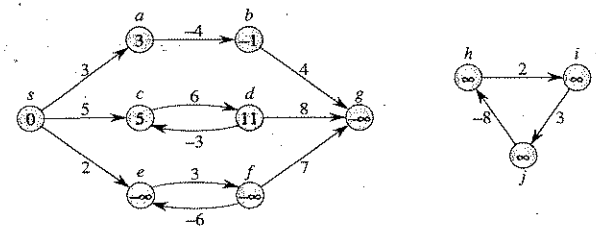


קשתות בעלות משקלות שליליים

במופעים מסוימים של בעיית המסלולים הקצרים ביותר ממקור יחיד עשויות להיות קשתות בעלות משקלות שליליים. אם הגרף $G = (V, E)$ אינו מכיל מעגלים בעלי משקל שלילי שניתן להגיע אליהם מן המקור s, אזי עבור כל $v \in V$, משקל המסלול הקצר ביותר $\delta(s, v)$ עדיין מוגדר היטב, גם אם ערכו שלילי. אולם אם קיים מעגל בעל משקל שלילי שניתן להגיע אליו מ־s, אזי המשקלים של מסלולים קצרים ביותר אינם מוגדרים היטב. אף מסלול מ־s אל קדקוד על המעגל אינו יכול להיות מסלול קצר ביותר — תמיד ניתן למצוא מסלול בעל משקל נמוך יותר שעובר לאורכו של המסלול "הקצר ביותר" המוצע ואז מקיף את המעגל בעל המשקל השלילי. אם קיים מעגל בעל משקל שלילי על מסלול כלשהו מ־s אל v, אנו מגדירים $\delta(s, v) = -\infty$.

איור 25.1 מדגים את השפעתם של משקלות שליליים על משקליהם של מסלולים קצרים ביותר. מכיוון שקיים מסלול אחד בלבד מ־s אל a (המסלול $\langle s, a \rangle$), אזי $\delta(s, a) = w(s, a) = 3$. בדומה לכך, קיים מסלול אחד בלבד מ־s אל b, ולכן, $\delta(s, b) = w(s, a) + w(a, b) = 3 + (-4) = -1$. קיים מספר אינסופי של מסלולים מ־s אל c: $\langle s, c \rangle$, $\langle s, c, d, c \rangle$, $\langle s, c, d, c, d, c \rangle$, וכן הלאה. מכיוון שמשקלו של המעגל $\langle c, d, c \rangle$ הוא $3 + (-3) = 0 > 0$, המסלול הקצר ביותר מ־s אל c הוא $\langle s, c \rangle$, שמשקלו $\delta(s, c) = 5$. באותו אופן, המסלול הקצר ביותר מ־s אל d הוא $\langle s, c, d \rangle$, שמשקלו $\delta(s, d) = w(s, c) + w(c, d) = 11$. באופן אנלוגי, קיים מספר אינסופי של מסלולים מ־s אל e: $\langle s, e \rangle$, $\langle s, e, f, e \rangle$, $\langle s, e, f, e, f, e \rangle$, וכן הלאה. אולם, מכיוון שמשקלו של המעגל $\langle e, f, e \rangle$ הוא $-3 + (-6) = -3 < 0$, לא קיים מסלול קצר ביותר מ־s אל e. על־ידי מעבר על המעגל $\langle e, f, e \rangle$ מספר פעמים רב כרצוננו, ניתן למצוא מסלולים מ־s אל e בעלי משקל שלילי גדול כרצוננו, ולפיכך $\delta(s, e) = -\infty$. באותו אופן, $\delta(s, f) = -\infty$. מכיוון ש־g ניתן להגעה מ־f, ניתן

גם לבנות מסלולים בעלי משקל שלילי גדול כרצוננו מ־s אל g, ו־$\delta(s, g) = -\infty$. הקדקודים h, i ו־j יוצרים גם הם מעגל בעל משקל שלילי. אולם לא ניתן להגיע אליהם מ־s, ולכן $\delta(s, h) = \delta(s, i) = \delta(s, j) = \infty$.



איור 25.1 גרף מכוון המכיל קשתות בעלות משקלות שליליים. בתוך כל קדקוד רשום משקלו של המסלול הקצר ביותר מהמקור s אל קדקוד זה. מכיוון שהקדקודים e ו־f יוצרים מעגל בעל משקל שלילי שניתן להגיע אליו מ־s, משקל המסלולים הקצרים ביותר אליהם הוא $-\infty$. מכיוון שאל הקדקוד g ניתן להגיע מקדקוד שמשקל המסלול הקצר ביותר אליו הוא $-\infty$, גם משקלו של המסלול הקצר ביותר אל g הוא $-\infty$. אל קדקודים כגון i ,h ו־j לא ניתן להגיע מ־s, ולכן משקלי המסלולים הקצרים ביותר אליהם הם ∞, למרות שהם נמצאים על מעגל בעל משקל שלילי.

אחדים מן האלגוריתמים למציאת מסלולים קצרים ביותר, האלגוריתם של דייקסטרה למשל, מניחים שכל משקלות הקשתות בגרף הקלט הם אי־שליליים, כמו בדוגמה של מפת הדרכים. אלגוריתמים אחרים, אלגוריתם בלמן־פורד למשל, מאפשרים קשתות בעלות משקלות שליליים בגרף הקלט, ומחזירים תשובה נכונה כל עוד לא קיימים בגרף מעגלים בעלי משקל שלילי שניתן להגיע אליהם מקדקוד המקור. אם קיים מעגל בעל משקל שלילי כזה, האלגוריתם מסוגל בדרך כלל לגלות אותו ולדווח על קיומו.

ייצוג מסלולים קצרים ביותר

לעתים קרובות אנו רוצים למצוא לא רק משקלים של מסלולים קצרים ביותר אלא גם את הקדקודים הנמצאים עליהם. הייצוג שנשתמש בו עבור מסלולים קצרים ביותר דומה לזה שהשתמשנו בו עבור עצי רוחב בסעיף 23.2. בהינתן גרף $G = (V, E)$, אנו מתחזקים עבור כל קדקוד $v \in V$ את **הקודם** (predecessor) של v, הנקרא $\pi[v]$, שיכול להיות קדקוד אחר או NIL. האלגוריתמים למציאת מסלולים קצרים ביותר שנציג בפרק זה מציבים בשדות π ערכים כך ששרשרת הקודמים המתחילה בקדקוד v חוזרת אחורה אל s לאורך מסלול קצר ביותר מ־s אל v. לפיכך, בהינתן קדקוד v שעבורו $\pi[v] \neq \text{NIL}$, השגרה PRINT-PATH(G, s, v) מסעיף 23.2 יכולה לשמש להדפסת מסלול קצר ביותר מ־s אל v.

עם זאת, במהלך ביצועו של אלגוריתם למציאת מסלולים קצרים ביותר, ערכי π לא חייבים להצביע על מסלולים קצרים ביותר. כמו בחיפוש לרוחב, אנו נתעניין ב**תת־גרף הקודמים** (predecessor subgraph) $G_\pi = (V_\pi, E_\pi)$ המושרה על־ידי הערכים π. גם כאן, נגדיר את קבוצת הקדקודים V_π כקבוצת הקדקודים מ־G אשר הקודמים להם אינם NIL, בצירוף הקדקוד s:

$$V_\pi = \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}$$

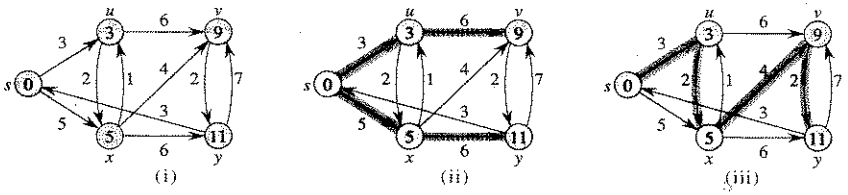
קבוצת הקשתות המכוונות E_π היא קבוצת הקשתות המושרה על־ידי ערכי π של הקדקודים השייכים לקבוצה V_π:

$$E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$$

אנו נוכיח שהערכים π שמפיקים האלגוריתמים שנציג בפרק זה מתאפיינים בכך שעם סיום ההרצה, G_π הוא "עץ מסלולים קצרים ביותר", שהוא, בניסוח לא פורמלי, עץ מושרש המכיל מסלול קצר ביותר ממקור s אל כל קדקוד שניתן להגיע אליו מ־s. עץ מסלולים קצרים ביותר דומה לעץ הרוחב מסעיף 23.2, אולם הוא מכיל מסלולים קצרים ביותר מן המקור המוגדרים במונחים של משקלות הקשתות ולא במונחי מספר הקשתות. ביתר דיוק, יהי $G = (V, E)$ גרף מכוון משוקלל עם פונקציית משקל $w : E \to \mathbf{R}$, ונניח כי G אינו מכיל מעגלים בעלי משקל שלילי שניתן להגיע אליהם מקדקוד המקור $s \in V$, כך שמסלולים קצרים ביותר מוגדרים היטב. **עץ מסלולים קצרים ביותר** (shortest-paths tree) המושרש ב־s הוא תת־גרף מכוון $G' = (V', E')$, כאשר $V' \subseteq V$, $E' \subseteq E$, המקיים:

1. V' היא קבוצת הקדקודים ב־G שניתן להגיע אליהם מ־s;
2. G' יוצר עץ מושרש ששורשו s;
3. עבור כל $v \in V'$, המסלול הפשוט היחיד מ־s אל v ב־G' הוא מסלול קצר ביותר מ־s אל v ב־G.

מסלולים קצרים ביותר אינם בהכרח יחידים וכך גם עצי מסלולים קצרים ביותר. לדוגמה, באיור 25.2 מוצג גרף מכוון משוקלל ושני עצי מסלולים קצרים ביותר בעלי אותו שורש.



איור 25.2 **(i)** גרף מכוון משוקלל עם משקלי מסלולים קצרים ביותר מהמקור s. **(ii)** הקשתות המוצללות יוצרות עץ מסלולים קצרים ביותר המושרש במקור s. **(iii)** עץ מסלולים קצרים ביותר אחר, בעל אותו שורש.

תקציר הפרק

האלגוריתמים למציאת מסלולים קצרים ביותר ממקור יחיד שנציג בפרק זה מבוססים כולם על טכניקה הידועה בשם **הקלה** (relaxation). סעיף 25.1 פותח בהוכחת כמה תכונות חשובות של מסלולים קצרים ביותר באופן כללי, ואחר כך מוכיח כמה עובדות חשובות לגבי אלגוריתמים המבוססים על הקלה. בסעיף 25.2 נציג את האלגוריתם של דייקסטרה (Dijkstra), הפותר את הבעיה של מסלולים קצרים ביותר ממקור יחיד כאשר כל הקשתות הן בעלות משקלות אי־שליליים. בסעיף 25.3 נציג את אלגוריתם בלמן־פורד (Bellman-Ford), המשמש במקרה הכללי יותר, שבו קשתות יכולות להיות בעלות משקלות שליליים. אם הגרף מכיל מעגל בעל משקל שלילי שניתן להגיע אליו מן המקור, אלגוריתם בלמן־פורד מגלה את קיומו. בסעיף 25.4 נציג אלגוריתם זמן־לינארי לחישוב מסלולים קצרים ביותר ממקור יחיד בגרף מכוון ללא מעגלים. לבסוף, בסעיף 25.5 נראה כיצד ניתן להשתמש באלגוריתם בלמן־פורד כדי לפתור מקרה פרטי של "תכנון לינארי".

הניתוח שנערוך ידרוש כמה מוסכמות באשר לפעולות אריתמטיות על גדלים אינסופיים. אנו נניח שעבור כל מספר ממשי $a \neq -\infty$, מתקיים $a + \infty = \infty + a = \infty$. כמו כן, כדי שההוכחות שלנו יהיו תקפות גם בנוכחות מעגלים בעלי משקלים שליליים, נניח שעבור כל מספר ממשי $a \neq \infty$, מתקיים $a + (-\infty) = (-\infty) + a = -\infty$.

25.1 מסלולים קצרים ביותר וטכניקת ההקלה

כדי להבין אלגוריתמים למציאת מסלולים קצרים ביותר ממקור יחיד, כדאי להבין את הטכניקות שבהן הם משתמשים ואת תכונות המסלולים הקצרים ביותר שהם מנצלים. הטכניקה העיקרית שבה משתמשים האלגוריתמים שנציג בפרק זה היא הקלה, שיטה שבה מקטינים פעם אחר פעם חסם עליון על משקלו האמיתי של מסלול קצר ביותר אל כל קדקוד — עד שהחסם העליון שווה למשקל המסלול הקצר ביותר. בסעיף זה נראה כיצד פועלת הקלה ונוכיח באופן פורמלי תכונות אחדות שלה.

בקריאה ראשונה של סעיף זה ניתן לדלג על הוכחות המשפטים — לקרוא את טענותיהם בלבד — ולפנות מיד לאלגוריתמים שבסעיפים 25.2 ו־25.3. אולם יש לשים לב במיוחד ללמה 25.7, שהיא המפתח להבנת האלגוריתמים למציאת מסלולים קצרים ביותר שבפרק זה. כמו כן, אפשר להתעלם בקריאה ראשונה מן הלמות העוסקות בתת־גרף הקודמים ובעצי מסלולים קצרים ביותר (למות 25.8 ו־25.9) ולהתמקד תחת זאת בלמות הקודמות להן, הנוגעות למשקלים של מסלולים קצרים ביותר.

תת־מבנה אופטימלי של מסלול קצר ביותר

אלגוריתמים למציאת מסלולים קצרים ביותר מנצלים בדרך כלל את העובדה שמסלול קצר ביותר בין שני קדקודים מכיל בתוכו מסלולים אחרים קצרים ביותר. תכונה זו של תת־מבנה אופטימלי היא סימן היכר לכך שניתן לפתור את הבעיה באמצעות תכנון דינמי (פרק 16) או בגישה החמדנית (פרק 17). למעשה, האלגוריתם של דייקסטרה הוא אלגוריתם חמדן, ואלגוריתם פלויד־וורשאל (Floyd-Warshall), אשר מוצא מסלולים קצרים ביותר בין כל זוג קדקודים (ראה פרק 26), הוא אלגוריתם תכנון־דינמי. הלמה הבאה ומסקנתה מנסחות באופן מדויק יותר את תכונת תת־מבנה האופטימלי של מסלולים קצרים ביותר.

למה 25.1 (תת־מסלולים של מסלולים קצרים ביותר הם מסלולים קצרים ביותר)
בהינתן גרף מכוון משוקלל $G = (V, E)$ עם פונקציית משקל $w : E \to \mathbf{R}$, יהי $p = \langle v_1, v_2, \ldots, v_k \rangle$ מסלול קצר ביותר מהקדקוד v_1 אל הקדקוד v_k. כמו כן, עבור כל i ו־j כך ש־$1 \le i \le j \le k$, יהי $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ תת־מסלול של p מהקדקוד v_i אל הקדקוד v_j. אזי, p_{ij} הוא מסלול קצר ביותר מ־v_i אל v_j.

הוכחה אם נפרק את המסלול p באופן הבא: $v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$, אזי $w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$. כעת, נניח שקיים מסלול p'_{ij} מ־v_i אל v_j בעל משקל $w(p'_{ij}) < w(p_{ij})$. אזי, $v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p'_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$ הוא מסלול מ־v_1 אל v_k אשר משקלו, $w(p_{1i}) + w(p'_{ij}) + w(p_{jk})$, קטן מ־$w(p)$, בסתירה להנחתנו ש־$p$ הוא מסלול קצר ביותר מ־v_1 אל v_k. ■

כאשר חקרנו חיפוש לרוחב (סעיף 23.2), הוכחנו כלמה 23.1 תכונה פשוטה של מרחקים קצרים ביותר בגרפים לא משוקללים. המסקנה הבאה של למה 25.1 מכלילה תכונה זו גם לגרפים משוקללים.

מסקנה 25.2
יהי $G = (V, E)$ גרף מכוון משוקלל עם פונקציית משקל $w : E \to \mathbf{R}$. נניח שמסלול קצר ביותר p מן המקור s אל קדקוד v ניתן לפירוק באופן הבא: $s \overset{p'}{\rightsquigarrow} u \to v$, עבור איזשהם קדקוד u

ומסלול p'. אזי, משקלו של מסלול קצר ביותר מ־s אל v הוא $\delta(s, v) = \delta(s, u) + w(u, v)$.

הוכחה על פי למה 25.1, תת־מסלול p' הוא מסלול קצר ביותר מן המקור s אל הקדקוד u. לכן:

$$\begin{aligned} \delta(s, v) &= w(p) \\ &= w(p') + w(u, v) \\ &= \delta(s, u) + w(u, v) \end{aligned}$$

■

בלמה הבאה מוצגת תכונה פשוטה אך שימושית של משקליהם של מסלולים קצרים ביותר.

למה 25.3
יהי $G = (V, E)$ גרף מכוון משוקלל עם פונקציית משקל $w : E \to \mathbf{R}$ וקדקוד מקור s. אזי, עבור כל קשת $(u, v) \in E$, מתקיים $\delta(s, v) \le \delta(s, u) + w(u, v)$.

הוכחה משקלו של מסלול קצר ביותר p מן המקור s אל הקדקוד v אינו עולה על משקלו של כל מסלול אחר מ־s אל v. בפרט, משקלו של p אינו עולה על משקלו של המסלול המסוים העובר לאורך מסלול קצר ביותר מ־s אל u ואז על פני הקשת (u, v). ■

טכניקת ההקלה

האלגוריתמים המוצגים בפרק זה משתמשים בטכניקת **ההקלה** (relaxation). עבור כל קדקוד $v \in V$, אנו מתחזקים מאפיין $d[v]$, שהוא חסם עליון על משקלו של מסלול קצר ביותר מ־s אל v. ל־$d[v]$ אנו קוראים **אומדן של מסלול קצר ביותר** (shortest-path estimate). אומדני מסלולים קצרים ביותר וכן הקודמים של קדקודים מאותחלים באמצעות השגרה הבאה:

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ V[G]
2      do d[v] ← ∞
3         π[v] ← NIL
4  d[s] ← 0
```

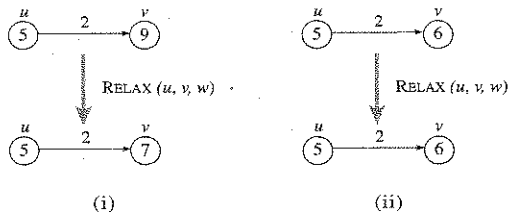
לאחר האתחול, $\pi[v] = \text{NIL}$ עבור כל $v \in V$, $d[v] = 0$ עבור $v = s$, וכן $d[v] = \infty$ עבור כל $v \in V - \{s\}$.

בתהליך **ההקלה**[1] של קשת (u, v) בודקים אם אפשר לשפר את המסלול הקצר ביותר אל v שמצאנו עד כה על־ידי מעבר דרך u, ואם כך הדבר, מעדכנים את $d[v]$ ואת $\pi[v]$. צעד הקלה עשוי להקטין את ערך האומדן של מסלול קצר ביותר $d[v]$ ולעדכן את השדה $\pi[v]$ המכיל את הקודם של v. השגרה הבאה מבצעת צעד הקלה על הקשת (u, v).

1. השימוש במונח "הקלה" לתיאור פעולה אשר דומה מהדקת חסם עליון עשוי להיראות מוזר. לשימוש במונח זה סיבות היסטוריות. את התוצאה של צעד הקלה ניתן לראות כהקלת חדרישה $d[v] \le d[u] + w(u, v)$, אשר, על פי למה 25.3, חייבת להתקיים אם $d[u] = \delta(s, u)$ ו־$d[v] = \delta(s, v)$. כלומר, אם $d[v] > d[u] + w(u, v)$, אין "לחץ" למלא דרישה זו, כך שיש כאן "הקלה" של הדרישה.

```
RELAX(u, v, w)
1  if d[v] > d[u] + w(u, v)
2     then d[v] ← d[u] + w(u, v)
3          π[v] ← u
```

באיור 25.3 מוצגות שתי דוגמאות להקלה של קשת. באחת מהן קטן אומדנו של מסלול קצר ביותר ואילו בשנייה שום אומדן אינו משתנה.



איור 25.3 הקלה של קשת (u, v) שמשקלה $w(u, v) = 2$. בתוך כל קדקוד רשום אומדן המסלול הקצר ביותר אליו. **(i)** מכיוון שלפני ההקלה מתקיים $d[v] > d[u] + w(u, v)$, ערכו של $d[v]$ קטן בעקבות ההקלה. **(ii)** כאן, $d[u] + w(u, v) \leq d[v]$ לפני צעד ההקלה, ולכן $d[v]$ אינו משתנה כתוצאה מן ההקלה.

כל אחד מן האלגוריתמים המוצגים בפרק זה קורא לשגרה INITIALIZE-SINGLE-SOURCE, ולאחר מכן מבצע פעם אחר פעם הקלה של קשתות. יצירה מזו, הקלה היא האמצעי היחיד לשינוי אומדני מסלולים קצרים ביותר וקודמים של קדקודים. האלגוריתמים בפרק זה נבדלים זה מזה במספר הפעמים שבו הם מבצעים הקלות על כל קשת ובסדר שבו הם מבצעים הקלת קשתות. באלגוריתם של דייקסטרה ובאלגוריתם למציאת מסלולים קצרים ביותר בגרף מכוון ללא מעגלים, על כל קשת מתבצעת בדיוק הקלה אחת. באלגוריתם בלמן־פורד, על כל קשת מתבצעות הקלות אחדות.

תכונות של הקלה

נכונותם של האלגוריתמים המוצגים בפרק זה תלויה בתכונות חשובות של הקלת קשתות המסוכמות בלמות אחדות המובאות להלן. רוב הלמות מתארות את התוצאה של ביצוע סדרת צעדי הקלה על קשתות בגרף מכוון משוקלל שאותחל באמצעות INITIALIZE-SINGLE-SOURCE. הלמות הללו, פרט ללמה 25.9, מתקיימות עבור סדרה *כלשהי* של צעדי הקלה, ולא רק עבור סדרות המניבות ערכי מסלולים קצרים ביותר.

למה 25.4
יהי $G = (V, E)$ גרף מכוון משוקלל עם פונקציית משקל $w : E \to \mathbf{R}$, ותהי $(u, v) \in E$. אזי, מיד לאחר הקלת הקשת (u, v) על־ידי ביצוע RELAX(u, v, w), מתקיים $d[v] \leq d[u] + w(u, v)$.

הוכחה אם, ממש לפני הקלת הקשת (u, v), מתקיים $d[v] > d[u] + w(u, v)$, אזי לאחר ההקלה מתקיים $d[v] = d[u] + w(u, v)$. אולם אם ממש לפני ההקלה מתקיים $d[v] \leq d[u] + w(u, v)$, אזי $d[u]$ ו־$d[v]$ אינם משתנים, ולכן $d[v] \leq d[u] + w(u, v)$ גם לאחר ההקלה. ■

למה 25.5
יהי $G = (V, E)$ גרף מכוון משוקלל עם פונקציית משקל $w : E \to \mathbf{R}$. יהי $s \in V$ קדקוד המקור, ונניח כי הגרף אותחל על־ידי INITIALIZE-SINGLE-SOURCE(G, s). אזי, $d[v] \geq \delta(s, v)$ עבור כל $v \in V$, ואינווארינטה זו נשמרת במהלך כל סדרה שהיא של צעדי הקלה על הקשתות של G. יתר על כן, מרגע ש־$d[v]$ מגיע לחסם התחתון שלו $\delta(s, v)$, ערכו שוב אינו משתנה לעולם.

הוכחה האינווארינטה $d[v] \geq \delta(s, v)$ בוודאי מתקיימת לאחר האתחול, שכן $d[s] = 0 \geq \delta(s, s)$ (נשים לב כי $\delta(s, s)$ הוא $-\infty$ אם s נמצא על מעגל בעל משקל שלילי, ו־0 אחרת), ועבור כל $v \in V - \{s\}$, $d[v] = \infty$ גורר $d[v] \geq \delta(s, v)$. אנו נוכיח בדרך השלילה שהאינווארינטה נשמרת במהלך סדרה כלשהי של צעדי הקלה. יהי v הקדקוד הראשון שעבורו צעד הקלה על קשת (u, v) גורם לכך ש־$d[v] < \delta(s, v)$. אזי, מיד לאחר הקלת הקשת (u, v), נקבל:

$$\begin{aligned} d[u] + w(u, v) &= d[v] \\ &< \delta(s, v) \\ &\leq \delta(s, u) + w(u, v) \qquad \text{(על פי למה 25.3)} \end{aligned}$$

ומכאן נובע כי $d[u] < \delta(s, u)$. אולם מאחר שהקלת הקשת (u, v) אינה משנה את $d[u]$, הרי שאי־שוויון זה בהכרח היה נכון ממש לפני הקלת הקשת, בסתירה לבחירת v כקדקוד הראשון שעבורו $d[v] < \delta(s, v)$. אנו מסיקים מכך כי האינווארינטה $d[v] \geq \delta(s, v)$ מתקיימת עבור כל $v \in V$.

כדי להיווכח בכך שערכו של $d[v]$ שוב אינו משתנה לעולם מאותו רגע שבו $d[v] = \delta(s, v)$, נשים לב כי לאחר שערכו של $d[v]$ הגיע לחסם התחתון שלו, הוא אינו יכול לקטון, משום שזה עתה הראינו כי $d[v] \geq \delta(s, v)$, והוא אינו יכול לגדול, מכיוון שצעדי הקלה אינם מגדילים ערכי d. ■

מסקנה 25.6
נניח שבגרף מכוון משוקלל $G = (V, E)$ עם פונקציית משקל $w : E \to \mathbf{R}$, לא קיים שום מסלול המחבר קדקוד מקור $s \in V$ אל קדקוד נתון $v \in V$. אזי, לאחר אתחול הגרף על־ידי INITIALIZE-SINGLE-SOURCE(G, s), אנו מקבלים $d[v] = \delta(s, v)$, ושוויון זה נשמר כאינווארינטה במהלך כל סדרה שהיא של צעדי הקלה על הקשתות של G.

הוכחה על פי למה 25.5, תמיד מתקיים $d[v] \leq \delta(s, v) = \infty$, ולכן $d[v] = \infty = \delta(s, v)$. ■

הלמה הבאה היא מכריעה בהוכחת נכונותם של האלגוריתמים למציאת מסלולים קצרים ביותר המופיעים בהמשך פרק זה. הלמה נותנת תנאים מספיקים לכך שהקלה תגרום לכך שאומדן של מסלול קצר ביותר יתכנס למשקלו של מסלול קצר ביותר.

למה 25.7
יהי $G = (V, E)$ גרף מכוון משוקלל עם פונקציית משקל $w : E \to \mathbf{R}$, יהי $s \in V$ קדקוד מקור, ויהי $s \rightsquigarrow u \to v$ מסלול קצר ביותר ב־G, עבור קדקודים כלשהם $u, v \in V$. נניח כי G מאותחל באמצעות INITIALIZE-SINGLE-SOURCE(G, s), ואז מבוצעת על קשתות G סדרה של צעדי הקלה הכוללת את הקריאה RELAX(u, v, w). אם $d[u] = \delta(s, u)$ בזמן כלשהו לפני הקריאה, אזי $d[v] = \delta(s, v)$ בכל זמן לאחר הקריאה.

$$d[v_k] > d[v_{k-1}] + w(v_{k-1}, v_k)$$

יתכנו גם מקרים שבהם קיים מסלול קצר ביותר מ s אל v אבל אלגוריתם דייקסטרה לא יכול למצוא אותו. לדוגמה, נניח שמסלול קצר ביותר מ s אל v הוא $\langle v_0, v_1, \ldots, v_k \rangle$ כאשר $v_0 = s$ ו $v_k = v$, אבל אלגוריתם דייקסטרה מוצא את המסלול $\langle s, v \rangle$ בגלל שהקשתות (v_i, v_{i+1}) מסומנות בסדר עולה של i ולא בסדר יורד. אם כל הקשתות היוצאות מ s מסומנות בערכים גדולים מאוד, אז ייתכן שמסלול קצר ביותר מ s אל v יהיה הרבה יותר ארוך מ $\langle s, v \rangle$ ולכן $d[v]$ יהיה הרבה יותר גדול מ $\delta(s, v)$.

מסלולים קצרים ביותר ומכוונות ללא מעגלים

אלגוריתם דייקסטרה מוצא מסלולים קצרים ביותר מכל מקור יחיד בגרף מכוון $G = (V, E)$ שבו משקל כל קשת הוא אי-שלילי. נראה כעת שאם G הוא מכוון וללא מעגלים, אז אפשר להשתמש בשיטה דומה כדי למצוא מסלולים קצרים ביותר מכל מקור יחיד. הזמן הנדרש הוא $\Theta(V + E)$. נזכור שבגרף מכוון וללא מעגלים תמיד יש סדר טופולוגי לקודקודים, ולכן נניח שהקשתות של G מסומנות בהתאם לסדר טופולוגי. אפשר למצוא סדר טופולוגי בזמן $\Theta(V + E)$ (משפט 22.12).

האלגוריתם הבא מחשב $d[v]$ ו $\pi[v]$ עבור כל $v \in V$ בגרף מכוון וללא מעגלים $G = (V, E)$ עם פונקציית משקל $w : E \to \mathbf{R}$ ומקור $s \in V$.

```
DAG-SHORTEST-PATHS(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for כל קודקוד u, לפי סדר טופולוגי
3      do for כל קודקוד v ∈ Adj[u]
4          do RELAX(u, v, w)
```

הדוגמה שבאיור 25.5 מראה כיצד פועל האלגוריתם.

משפט 25.6

אם גרף מכוון וללא מעגלים $G = (V, E)$ מכיל קודקוד s ו $s \rightsquigarrow v$ עבור $v \in V$, אז בסיום האלגוריתם $d[v] = \delta(s, v)$ ומתקיים תת-גרף הקודמים G_π הוא עץ מסלולים קצרים ביותר מ s.

הוכחה אם $s \rightsquigarrow v$ עבור $v \in V$, אז s הוא קודקוד ראשון שנבחר ב v בשורה 3. לכן $d[s] = \delta(s, s)$ בסיום האלגוריתם. נניח $s \rightsquigarrow v$ עבור $v \in V$ ונניח $\langle v_0, v_1, \ldots, v_k \rangle$ כאשר $v_0 = s$ ו $v_k = v$, ונניח שהקשתות (v_i, v_{i+1}) מסומנות בסדר עולה של i. נראה כי $d[v_i] = \delta(s, v_i)$ עבור $i = 1, 2, \ldots, k$ לפיכך, עבור $v = v_k$ מתקיים $d[v] = \delta(s, v)$. תכונת שלילי המסלולים בעלי משקל שלילי שניתן להגיע אליהם מן המקור מבטיחה כי $d[v_{i-1}] \leq d[v_i]$ מתקיים עבור $i = 1, 2, \ldots, k$. מכיל מעגלים ו $\pi[v_i] = v_{i-1}$ עבור $i = 1, 2, \ldots, k$. מכיוון שהקשתות (v_{i-1}, v_i) מסומנות בסדר עולה של i, אף אחד מהקודקודים $v_0, v_1, \ldots, v_k$ לא יכול להיות מסומן לאחר v_{i-1}. לפיכך, עבור $i = 1, 2, \ldots, k$, התנאי $d[v_{i-1}] \leq d[v_i]$ מתקיים. אם $d[v_{i-1}] < d[v_i]$ אז $d[v_{i-1}] + w(v_{i-1}, v_i) < d[v_i]$, ולכן $d[v_i]$ השתנה בשורה 4 של RELAX. אם $d[v_{i-1}] = d[v_i]$ אז $d[v_{i-1}] + w(v_{i-1}, v_i) = d[v_i]$, ולכן $d[v_i]$ לא השתנה בשורה 4 של RELAX. מכאן נובע כי $d[v_i] = \delta(s, v_i)$ עבור $i = 1, 2, \ldots, k$. לאחר שקשת (v_{i-1}, v_i) עברה רלקסציה, $d[v_i] = \delta(s, v_i)$ ו $\pi[v_i] = v_{i-1}$ נשארים ללא שינוי.

איור 25.4 תצלום התוכחה לכך שמשקל $\delta(s, v)$ מקיים את אי-השוויון $\delta(s, v) \leq \delta(s, u) + w(u, v)$ אם $s \rightsquigarrow u \rightsquigarrow v$ ו $x \rightsquigarrow y$ מסמל קשת (x, y) שמשקלה $w(x, y)$. מכיוון ש p_2 הוא מסלול קצר ביותר מ s אל v ב G_π, נובע כי p_2 הוא מסלול קצר ביותר מ s אל u ב G_π. לכן $v.d = u.d + w(u, v)$. אם $y = z$ או $\pi[z] = y$ אז $x \rightsquigarrow y \rightsquigarrow z \rightsquigarrow u \rightsquigarrow v$ ו $x \rightsquigarrow z \rightsquigarrow u \rightsquigarrow v$ הם אותו מסלול, ולכן $z.d = y.d$ ומכאן $\pi[z] = y$. אחרת, $\pi[z] = x$ ומכאן $z.d = x.d + w(x, z) < y.d$, בסתירה לכך ש y הוא הראשון במסלול p_1 שמקיים $y.d \geq \delta(s, y)$. לכן $\pi[z] = y$ ומכאן $z.d = y.d$. בכל מקרה $z.d = y.d$. מכאן נובע כי $v.d = u.d + w(u, v)$.

הוכחה על פי למה 25.5, אם $u = \delta(s, u)$ ברגע כלשהו לפני הקלת הקשת (u, v), אזי $d[u] = \delta(s, u)$ שמתקיים מאז מתקיים גם לאחר מכן.

על פי למה 25.5, $d[v] = \delta(s, v)$ מתחזקת, ומכך אנו מסיקים כי $\delta(s, v) = d[v]$ בכל עת.

■

עצי מסלולים קצרים ביותר

עד כה התמקדנו בחישוב משקלי המסלולים הקצרים ביותר. לעתים קרובות אנו מעוניינים לחשב את המסלולים עצמם, ונדון כעת בשיטות שבהן משתמשים. נתון גרף $G = (V, E)$ עם פונקציית משקל $w : E \to \mathbf{R}$, ונניח שאין מעגלים שליליים. עבור כל קודקוד $v \in V$, נחשב מסלול קצר ביותר p מ s אל v. במהלך חישוב $d[v]$ לכל $v \in V$, נחשב עבור כל קודקוד $v \neq s$ את קודמו $\pi[v]$, שהוא קודקוד אחר או NIL. האלגוריתם מבצע חיפוש לאחור מ v ומוצא את v הקודמים לו לאורך מסלול קצר ביותר מ s אל v. לכן, אם $\pi[v] = u$, אז u הוא הקודם האחרון ב s לאורך מסלול קצר ביותר מ s אל v. כלומר, $v.d = u.d + w(u, v)$.

תת-גרף הקודמים G_π שמושרה על ידי ערכי π נקרא **עץ מסלולים קצרים ביותר**. בדוגמה שבאיור 25.5 עץ המסלולים הקצרים ביותר G_π מוצג בחלק (ב) של האיור. נגדיר את G_π ואת ערכי π שהוא מחשב. נגדיר את **גרף הקודמים** של G ביחס ל π כ $G_\pi = (V_\pi, E_\pi)$, כאשר

$$V_\pi = \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}$$

ו

$$E_\pi = \{(\pi[v], v) : v \in V_\pi - \{s\}\}$$

V_π היא קבוצת הקודקודים ב G שהם קודמים של קודקוד כלשהו, יחד עם s. E_π היא קבוצת הקשתות שמסמנות קודקודים ב V_π ל s. נוכיח שאם G הוא גרף מכוון וללא מעגלים, אז π מקיים את התכונה שעץ מסלולים קצרים ביותר מ s אל v מוגדר ב G_π עבור כל $v \in V_\pi$. נתחיל בלמה שמראה שתת-גרף של גרף מכוון וללא מעגלים הוא גרף מכוון וללא מעגלים.

למה 25.8

יהי $G = (V, E)$ גרף מכוון משקל עם פונקציית משקל $w : E \to \mathbf{R}$ וקודקוד מקור $s \in V$, כך ש G אינו מכיל מעגלים בעלי משקל שלילי שניתן להגיע אליהם מ s. יהי G_π תת-גרף הקודמים שנוצר עקב הרצת האלגוריתם INITIALIZE-SINGLE-SOURCE(G, s) ו RELAX על הקשתות של G. אז G_π הוא גרף מכוון וללא מעגלים.

הוכחה נניח שקיים מעגל $c = \langle v_0, v_1, \ldots, v_k \rangle$ כאשר $v_k = v_0$ ו $k > 0$, כך ש $\pi[v_i] = v_{i-1}$ עבור $i = 1, 2, \ldots, k$. נראה כי c הוא מעגל ב G, שסותר את ההנחה כי G אינו מכיל מעגלים. נראה כי $v_i.d \geq v_{i-1}.d$ עבור $i = 1, 2, \ldots, k$, ונקבל $v_k.d \geq v_0.d$. אי-השוויון הקשה הוא עבור $i = 1$. לפני הקריאה הראשונה ל RELAX בשורה 4, $v_1.d = \infty$ או $v_1.d \geq \delta(s, v_1)$. לאחר הקריאה הראשונה, $v_1.d \leq v_0.d + w(v_0, v_1)$. מכאן נובע כי $v_0.d \geq 0$, ולכן $v_1.d \leq v_0.d + w(v_0, v_1)$. עבור $i = 2, 3, \ldots, k$, בפעם הראשונה שבה $v_i.d$ משתנה בשורה 4 של RELAX, $v_i.d$ הופך להיות שווה ל $v_{i-1}.d + w(v_{i-1}, v_i)$. לאחר מכן, $v_{i-1}.d$ לא משתנה, ומכיוון ש $v_{i-1}.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$ נובע כי $v_i.d \geq v_{i-1}.d$ עבור $i = 2, 3, \ldots, k$. לכן

$$v_k.d \geq v_0.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

אבל:

$$\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$$

שכן כל קודקוד על המעגל c מופיע בדיוק פעם אחת בכל סכום. מכאן נובע כי

$$0 > \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

שהוא בדיוק אי-השוויון שקיבלנו לעיל, ובכך הוכחנו שהמעגל c הוא שלילי.

■

נחזור כעת להוכחה כי G_π הוא עץ מסלולים קצרים ביותר. נתחיל בלמה שמראה כי תת-גרף הקודמים מקיים את התכונה הרקורסיבית של מסלולים קצרים ביותר. לאחר שנוכיח כי G_π הוא גרף מכוון וללא מעגלים, נוכיח כי G_π מכיל עץ מסלולים קצרים ביותר מ s. לבסוף נוכיח כי G_π מקיים את אי-השוויון $d[v] \leq \delta(s, v)$ לכל $v \in V$, ומכך נובע כי $d[v] = \delta(s, v)$ לכל $v \in V$.

למה 25.9

יהי $G = (V, E)$ גרף מכוון משקל עם פונקציית משקל $w : E \to \mathbf{R}$ וקודקוד מקור $s \in V$, כך ש G אינו מכיל מעגלים בעלי משקל שלילי שניתן להגיע אליהם מ s. נניח שנקרא INITIALIZE-SINGLE-SOURCE(G, s) ואחר כך מבצעים סדרה של קריאות ל RELAX על הקשתות של G שמביאה לכך ש $v.d = \delta(s, v)$ לכל $v \in V_\pi$. אז תת-גרף הקודמים G_π הוא עץ מסלולים קצרים ביותר עם שורש s ומקיים את התכונה הרקורסיבית של מסלולים קצרים ביותר שמוגדרת בלמה 25.1 (עמוד 1.25).

למה 25.9

יהי $G = (V, E)$ גרף מכוון משוקלל, עם פונקציית משקל $w : E \to \mathbf{R}$ ויהי $s \in V$ קדקוד מקור, ונניח כי G אינו מכיל מעגלים בעלי משקל שלילי שניתן להגיע אליהם מ־s. נקרא ל־INITIALIZE-SINGLE-SOURCE(G, s), ולאחר מכן נבצע סדרה כלשהי של צעדי הקלה על קשתות של G המביאה ל־$d[v] = \delta(s, v)$ עבור כל $v \in V$. אזי, תת־גרף הקודמים G_π הוא עץ מסלולים קצרים ביותר המושרש ב־s.

הוכחה עלינו להוכיח ש־G_π מקיים את שלוש התכונות של עצי מסלולים קצרים ביותר. כדי להוכיח את קיומה של התכונה הראשונה, עלינו להראות כי V_π היא קבוצת הקדקודים שניתן להגיע אליהם מ־s. על פי ההגדרה, משקל של מסלול קצר ביותר $\delta(s, v)$ הוא סופי אם ורק אם v ניתן להגעה מ־s, ולכן הקדקודים שניתן להגיע אליהם מ־s הם בדיוק אלה שהם בעלי ערכי d סופיים. אולם, עבור קדקוד $v \in V - \{s\}$ הוצב ערך סופי בשדה $d[v]$ אם ורק אם $\pi[v] \neq \text{NIL}$. לכן, הקדקודים ב־V_π הם בדיוק אלה שניתן להגיע אליהם מ־s.

התכונה השנייה נובעת ישירות מלמה 25.8.

נותר אפוא להוכיח כי G_π מקיים את התכונה השלישית של עצי מסלולים קצרים ביותר: עבור כל $v \in V_\pi$, המסלול הפשוט היחיד $s \xrightarrow{p} v$ ב־G_π הוא מסלול קצר ביותר מ־s אל v ב־G. יהי $p = \langle v_0, v_1, \ldots, v_k \rangle$, כאשר $v_0 = s$ ו־$v_k = v$. עבור $i = 1, 2, \ldots, k$, מתקיים $d[v_i] = \delta(s, v_i)$ וגם $d[v_i] \geq d[v_{i-1}] + w(v_{i-1}, v_i)$, ומכך אנו מסיקים כי $w(v_{i-1}, v_i) \leq \delta(s, v_i) - \delta(s, v_{i-1})$. נסכם את המשקלים לאורך המסלול p ונקבל:

$$\begin{aligned} w(p) &= \sum_{i=1}^{k} w(v_{i-1}, v_i) \\ &\leq \sum_{i=1}^{k} (\delta(s, v_i) - \delta(s, v_{i-1})) \\ &= \delta(s, v_k) - \delta(s, v_0) \\ &= \delta(s, v_k) \end{aligned}$$

השורה השלישית מתקבלת מן השנייה מכיוון שהסכום בשורה השנייה מצטמצם טלסקופית, והשורה הרביעית נובעת מכך שמתקיים $\delta(s, v_0) = \delta(s, s) = 0$. לכן, $w(p) \leq \delta(s, v_k)$. מאחר ש־$\delta(s, v_k)$ הוא חסם תחתון על משקלו של מסלול כלשהו מ־s אל v_k, אנו מסיקים כי $w(p) = \delta(s, v_k)$, ו־p הוא אפוא מסלול קצר ביותר מ־s אל $v = v_k$. ■



תרגילים

25.1-1

מצא שני עצי מסלולים קצרים ביותר עבור הגרף המכוון מאיור 25.2, השונים מאלה המוצגים באיור.

25.1-2

מצא דוגמה לגרף מכוון משוקלל $G = (V, E)$ עם פונקציית משקל $w : E \to \mathbf{R}$ וקדקוד מקור s המקיים את התכונה הבאה: עבור כל קשת $(u, v) \in E$, קיים עץ מסלולים קצרים ביותר המושרש ב־s אשר מכיל את (u, v), וקיים עץ מסלולים קצרים ביותר אחר המושרש ב־s, אשר אינו מכיל את (u, v).

25.1-3

הרחב את הוכחת למה 25.3 לטיפול במקרים שבהם משקליהם של מסלולים קצרים ביותר הם ∞ או $-\infty$.

25.1-4

יהי $G = (V, E)$ גרף מכוון משוקלל עם קדקוד מקור s, ונניח כי G מאותחל על־ידי השגרה INITIALIZE-SINGLE-SOURCE(G, s). הוכח שאם סדרה של צעדי הקלה מציבה ב־$\pi[s]$ ערך שונה מ־NIL, אזי G מכיל מעגל בעל משקל שלילי.

25.1-5

יהי $G = (V, E)$ גרף מכוון משוקלל שאינו מכיל קשתות בעלות משקל שלילי. יהי $s \in V$ קדקוד מקור, ונגדיר את $\pi[v]$ באופן הרגיל: $\pi[v]$ הוא הקודם של v על מסלול קצר ביותר כלשהו מ־s אל v אם $v \in V - \{s\}$ ניתן להגעה מ־s, ו־$\pi[v] = \text{NIL}$ אחרת. הבא דוגמה לגרף G כזה ולהצבה של ערכי π היוצרת מעגל ב־G_π. (על פי למה 25.8, הצבה כזאת אינה יכולה להיווצר על־ידי סדרה של צעדי הקלה.)

25.1-6

יהי $G = (V, E)$ גרף מכוון משוקלל עם פונקציית משקל $w : E \to \mathbf{R}$ וללא מעגלים בעלי משקל שלילי. יהי $s \in V$ קדקוד המקור, ונניח כי הגרף G מאותחל על־ידי INITIALIZE-SINGLE-SOURCE(G, s). הוכח שעבור כל קדקוד $v \in V_\pi$, קיים ב־G_π מסלול מ־s אל v, ותכונה זו נשמרת כאינווריאנטה במהלך סדרה כלשהי של צעדי הקלה.

25.1-7

יהי $G = (V, E)$ גרף מכוון משוקלל שאינו מכיל מעגלים בעלי משקל שלילי. יהי $s \in V$ קדקוד המקור, ונניח כי G מאותחל על־ידי INITIALIZE-SINGLE-SOURCE(G, s). הוכח שקיימת סדרה של $|V| - 1$ צעדי הקלה המביאה ל־$d[v] = \delta(s, v)$ עבור כל $v \in V$.

25.1-8

יהי G גרף מכוון משוקלל כלשהו המכיל מעגל בעל משקל שלילי שניתן להגיע אליו מקדקוד המקור s. הראה שתמיד ניתן לבנות סדרה אינסופית של הקלות על הקשתות של G כך שכל הקלה בה גורמת לשינוי של אומדן של מסלול קצר ביותר.

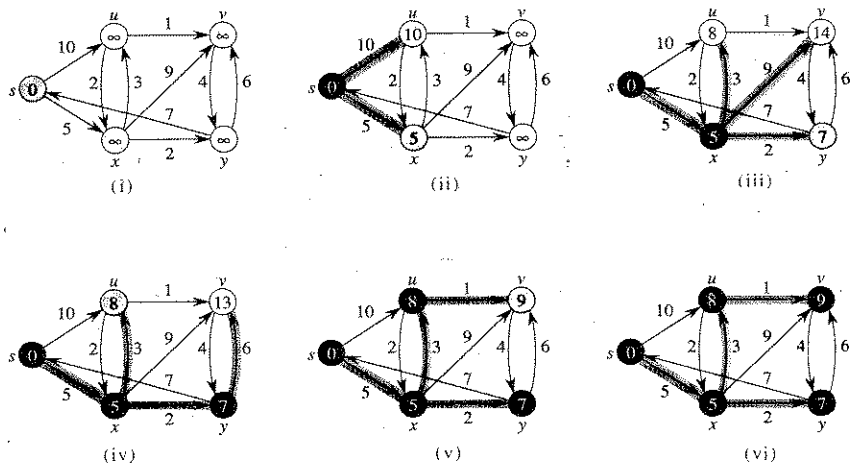
25.2 האלגוריתם של דייקסטרה

האלגוריתם של דייקסטרה (Dijkstra) פותר את הבעיה של מסלולים קצרים ביותר ממקור יחיד עבור גרף מכוון משוקלל $G = (V, E)$ במקרה שבו כל משקלות הקשתות הם אי־שליליים. בסעיף זה אנו מניחים אפוא כי $w(u, v) \geq 0$ עבור כל קשת $(u, v) \in E$.

האלגוריתם של דייקסטרה מנהל קבוצה S של קדקודים אשר עבורם כבר נקבעו סופית משקלי המסלולים הקצרים ביותר מן המקור s. כלומר, עבור כל הקדקודים $v \in S$, מתקיים $d[v] = \delta(s, v)$. האלגוריתם בוחר בכל איטרציה את הקדקוד $u \in V - S$ בעל האומדן המינימלי של מסלול קצר ביותר, מכניס את u ל־S, ומבצע הקלה על כל הקשתות היוצאות מ־u. במימוש שלהלן אנו מנהלים תור קדימויות Q המכיל את כל הקדקודים השייכים ל־$V - S$, כאשר ערכי d שלהם משמשים כמפתחות. המימוש מניח שהגרף G מיוצג על־ידי רשימות סמיכות.

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← φ
3  Q ← V[G]
4  while Q ≠ φ
5      do u ← EXTRACT-MIN(Q)
6         S ← S ∪ {u}
7         for each vertex v ∈ Adj[u]
8             do RELAX(u, v, w)
```

האלגוריתם של דייקסטרה מבצע הקלת קשתות כמתואר באיור 25.5. שורה 1 מבצעת את האתחול הרגיל של ערכי d ו־π, ושורה 2 מאתחלת את S לקבוצה הריקה. לאחר מכן, שורה 3 מאתחלת את תור הקדימויות Q כך שיכיל את כל הקדקודים השייכים ל־$V - S = V - \phi = V$. בכל פעם שמבוצעת לולאת ה־**while** שבשורות 4-8, קדקוד u נשלף מ־$Q = V - S$ ומוכנס לקבוצה S. (באיטרציה הראשונה של לולאה זו, $u = s$.) מבין הקדקודים ב־$V - S$, הקדקוד u הוא אפוא בעל האומדן הקטן ביותר של משקל המסלול הקצר ביותר. לאחר מכן, שורות 7-8 מבצעות הקלה על כל קשת (u, v) היוצאת מ־u, ועל־ידי כך מעדכנות את האומדן $d[v]$ ואת הקודם $\pi[v]$ אם המסלול הקצר ביותר אל v ניתן לשיפור על־ידי מעבר דרך u. נשים לב כי קדקודים לעולם אינם מוכנסים ל־Q לאחר שורה 3, וכי כל קדקוד נשלף מ־Q ומוכנס ל־S בדיוק פעם אחת, כך שלולאת ה־**while** שבשורות 4-8 מבוצעת בדיוק $|V|$ פעמים.



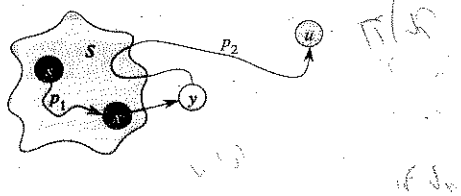
איור 25.5 ביצוע האלגוריתם של דייקסטרה. המקור הוא הקדקוד השמאלי ביותר. אומדני המסלולים הקצרים ביותר רשומים בתוך הקדקודים, וקשתות מוצללות מציינות את ערכי הקודמים: אם הקשת (u, v) מוצללת, אזי $\pi[v] = u$. קדקודים שחורים שייכים לקבוצה S, וקדקודים לבנים נמצאים בתור הקדימויות $Q = V - S$. **(i)** המצב ממש לפני האיטרציה הראשונה של לולאת ה־**while** שבשורות 4-8. הקדקוד המוצלל הוא בעל ערך d המינימלי והוא נבחר כקדקוד u בשורה 5. **(ii)-(vi)** המצב לאחר כל אחת מן האיטרציות העוקבות של לולאת ה־**while**. בכל חלק באיור, הקדקוד המוצלל הוא הקדקוד שנבחר כקדקוד u בשורה 5 של האיטרציה הבאה. ערכי d ו־π המופיעים בחלק **(vi)** הם הערכים הסופיים.

מכיוון שהאלגוריתם של דייקסטרה תמיד בוחר בקדקוד "הקל ביותר" או "הקרוב ביותר" מבין קדקודי $V - S$ כדי להכניסו לקבוצה S, אנו אומרים שהאלגוריתם מפעיל אסטרטגיה חמדנית. אסטרטגיות חמדניות מוצגות בפירוט בפרק 17, אולם קריאתו של פרק 17 אינה חיונית להבנת האלגוריתם של דייקסטרה. ככלל, אסטרטגיות חמדניות לא תמיד מניבות תוצאות אופטימליות, אולם, כפי שמוכיחים המשפט שלהלן והמסקנה הנובעת ממנו, האלגוריתם של דייקסטרה אכן מחשב מסלולים קצרים ביותר. כדי להוכיח טענה זו יש להראות שבכל פעם שקדקוד u מוכנס לקבוצה S, מתקיים $d[u] = \delta(s, u)$.

משפט 25.10 (נכונותו של האלגוריתם של דייקסטרה)
אם מריצים את האלגוריתם של דייקסטרה על גרף מכוון משוקלל $G = (V, E)$ עם פונקציית משקל אי־שלילית w ומקור s, אזי עם עצירת האלגוריתם, $d[u] = \delta(s, u)$ עבור כל הקדקודים $u \in V$.

הוכחה אנו נראה שעבור כל קדקוד $u \in V$, בעת הכנסתו של u לקבוצה S מתקיים $d[u] = \delta(s, u)$, ושוויון זה ממשיך להתקיים מרגע זה ואילך.

כדי להגיע לסתירה, נניח כי u הוא הקדקוד הראשון אשר בעת הכנסתו ל־S מתקיים $d[u] \neq \delta(s, u)$. נמקד את תשומת לבנו במצב המתקיים בתחילת אותה איטרציה של לולאת ה־**while** אשר במהלכה u מוכנס ל־S, ונראה, בסתירה להנחה, כי באותו זמן מתקיים $d[u] = \delta(s, u)$, על־ידי כך שנבחן מסלול קצר ביותר מ־s אל u. בהכרח מתקיים $u \neq s$, שכן s הוא הקדקוד הראשון המוכנס לקבוצה S ובעת הכנסתו $d[s] = \delta(s, s) = 0$. מאחר ש־$u \neq s$, הרי שממש לפני הכנסתו של u ל־S, מתקיים $S \neq \phi$. קיים בהכרח מסלול מ־s אל u, שכן אחרת $d[u] = \delta(s, u) = \infty$ (על פי מסקנה 25.6), בסתירה להנחתנו כי $d[u] \neq \delta(s, u)$. מכיוון שקיים לפחות מסלול אחד כזה, קיים מסלול p קצר ביותר מ־s אל u. המסלול p מחבר קדקוד מ־S, הקדקוד s, לקדקוד השייך ל־$V - S$, הקדקוד u. נתבונן בקדקוד הראשון y לאורך p המקיים $y \in V - S$, ויהי $x \in V$ הקודם של y. לפיכך, המסלול p ניתן לפירוק בצורה $s \overset{p_1}{\rightsquigarrow} x \rightarrow y \overset{p_2}{\rightsquigarrow} u$, כמתואר באיור 25.6.



איור 25.6 הוכחת משפט 25.10. ממש לפני הכנסתו של הקדקוד u לקבוצה S, S אינה ריקה. מסלול קצר ביותר p מן המקור s אל הקדקוד u ניתן לפירוק בצורה $s \overset{p_1}{\rightsquigarrow} x \rightarrow y \overset{p_2}{\rightsquigarrow} u$, כאשר y הוא הקדקוד הראשון לאורך המסלול אשר אינו שייך ל־S, ו־$x \in S$ הוא הקודם של y. הקדקודים x ו־y שונים זה מזה, אולם ייתכן כי $x = s$ או $u = y$. המסלול p_2 עשוי להיכנס ועשוי שלא להיכנס בחזרה לקבוצה S.

אנו טוענים כי בעת הכנסתו של u ל־S, $d[y] = \delta(s, y)$. כדי להוכיח טענה זו, נשים לב כי $x \in S$. כעת, מאחר ש־u נבחר כקדקוד הראשון שעבורו $d[u] \neq \delta(s, u)$ כאשר הוא מוכנס ל־S, הרי שבעת הכנסתו של x ל־S התקיים $d[x] = \delta(s, x)$. באותו זמן התבצעה הקלה על הקשת (x, y), ולכן הטענה נובעת מלמה 25.7.

עתה נוכל לקבל סתירה ולהוכיח בכך את המשפט. מכיוון ש־y מופיע לפני u על מסלול קצר ביותר מ־s אל u, ומכיוון שמשקלות כל הקשתות הם אי־שליליים (בפרט, משקלות הקשתות השייכות למסלול p_2), הרי ש־$\delta(s, y) \leq \delta(s, u)$, ולכן:

$$
\begin{aligned}
d[y] &= \delta(s, y) \\
&\le \delta(s, u) \qquad (25.2) \\
&\le d[u] \qquad \text{(על פי למה 25.5)}
\end{aligned}
$$

אולם מאחר שהקדקודים u ו־y השתייכו שניהם ל־$V - S$ כאשר u נבחר בשורה 5, הרי ש־$d[y] \le d[u]$. לכן, שני האי־שוויונים ב־(25.2) הם למעשה שוויונים, ואנו מקבלים:

$$d[y] = \delta(s, y) = \delta(s, u) = d[u]$$

מכאן נובע כי $d[u] = \delta(s, u)$, בסתירה לבחירתנו את u. נסיק כי עבור כל קדקוד $u \in V$, בעת הכנסתו של u לקבוצה S מתקיים $d[u] = \delta(s, u)$, ועל פי למה 25.5, שוויון זה מתקיים מאותו רגע ואילך. ■

מסקנה 25.11

אם מריצים את האלגוריתם של דייקסטרה על גרף מכוון משוקלל $G = (V, E)$ עם פונקציית משקל אי־שלילית w ומקור s, אזי, עם סיום ההרצה, תת־גרף הקודמים G_π הוא עץ מסלולים קצרים ביותר המושרש ב־s.

הוכחה ההוכחה נובעת ישירות ממשפט 25.10 ומלמה 25.9. ■

ניתוח

עד כמה מהיר הוא האלגוריתם של דייקסטרה? נתבונן תחילה במקרה שבו תור הקדימויות $Q = V - S$ מתוחזק כמערך לינארי. במימוש כזה, כל פעולת EXTRACT-MIN מתבצעת בזמן $O(V)$, ומכיוון שמתבצעות $|V|$ פעולות כאלה, הזמן הכולל שהן צורכות הוא $O(V^2)$. כל קדקוד $v \in V$ מוכנס לקבוצה S בדיוק פעם אחת, כך שכל קשת ברשימת הסמיכות $Adj[v]$ נבחנת בלולאת ה־**for** שבשורות 7-8 בדיוק פעם אחת במהלך ביצועו של האלגוריתם. מכיוון שמספר הקשתות הכולל בכל רשימות הסמיכות הוא $|E|$, מתבצעות בסך הכל $|E|$ איטרציות של לולאת **for** זו, כאשר כל איטרציה מתבצעת בזמן $O(1)$. זמן הריצה של האלגוריתם כולו הוא אפוא $O(V^2 + E) = O(V^2)$.

אבל, אם הגרף דליל, כדאי לממש את תור הקדימויות Q באמצעות ערימה בינרית. האלגוריתם המתקבל מכונה לעתים **אלגוריתם דייקסטרה המותאם** (modified Dijkstra algorithm). במקרה זה, כל פעולת EXTRACT-MIN מתבצעת בזמן $O(\lg V)$. כמו קודם, מתבצעות $|V|$ פעולות כאלה. הזמן לבניית הערימה הבינרית הוא $O(V)$. ההשמה $d[v] \leftarrow d[u] + w(u, v)$ בשגרה RELAX נעשית באמצעות הקריאה DECREASE-KEY$(Q, v, d[u] + w(u, v))$ המתבצעת בזמן $O(\lg V)$ (ראה תרגיל 7.5-4), ועדיין מתבצעות לכל היותר $|E|$ פעולות כאלה. זמן הריצה הכולל הוא אפוא $O((V + E) \lg V)$, שהוא $O(E \lg V)$ אם כל הקדקודים ניתנים להגעה מן המקור.

למעשה ניתן להשיג זמן ריצה של $O(V \lg V + E)$ על־ידי מימוש תור הקדימויות Q באמצעות ערימת פיבונצ'י (ראה פרק 21). העלות לשיעורין של כל אחת מ־$|V|$ הפעולות EXTRACT-MIN היא $O(\lg V)$, וכל אחת מ־$|E|$ הקריאות ל־DECREASE-KEY מתבצעת בזמן לשיעורין של $O(1)$ בלבד. מבחינה היסטורית, המוטיבציה לפיתוחן של ערימות פיבונצ'י היתה ההבחנה שבאלגוריתם דייקסטרה המשופר ישנן באופן פוטנציאלי הרבה יותר קריאות ל־DECREASE-KEY מאשר קריאות ל־EXTRACT-MIN, כך שכל שיטה המקטינה את הזמן לשיעורין של כל פעולת DECREASE-KEY ל־$o(\lg V)$ בלי להגדיל את הזמן לשיעורין של EXTRACT-MIN תניב מימוש מהיר יותר אסימפטוטית.

האלגוריתם של דייקסטרה דומה במובנים מסוימים לחיפוש לרוחב (ראה סעיף 23.2) וגם לאלגוריתם של פרים (Prim) ליצירת עצים פורשים מינימליים (ראה סעיף 24.2). הוא דומה לחיפוש לרוחב בכך שהקבוצה S מקבילה לקבוצת הקדקודים השחורים בחיפוש לרוחב; בדיוק כשם שהקדקודים ב־S כבר קיבלו את הערכים הסופיים של משקלי המסלולים הקצרים ביותר שלהם, כך הקדקודים השחורים בחיפוש לרוחב כבר קיבלו את הערכים הנכונים של מרחקי החיפוש־לרוחב שלהם. האלגוריתם של דייקסטרה דומה לאלגוריתם של פרים בכך ששני האלגוריתמים משתמשים בתור קדימויות כדי למצוא את הקדקוד "הקל ביותר" מבין הקדקודים שאינם שייכים לקבוצה נתונה (הקבוצה S באלגוריתם של דייקסטרה והעץ ההולך ונבנה באלגוריתם של פרים), מכניסים את הקדקוד הזה לקבוצה, ומעדכנים בהתאם את משקליהם של הקדקודים הנותרים מחוץ לקבוצה.

תרגילים

25.2-1

הרץ את האלגוריתם של דייקסטרה על הגרף המכוון שבאיור 25.2, תחילה עם הקדקוד s כמקור ולאחר מכן עם הקדקוד y כמקור. על פי דוגמת איור 25.5, הראה את ערכי d ו־π ואת הקדקודים בקבוצה S לאחר כל איטרציה של לולאת ה־**while**.

25.2-2

הבא דוגמה פשוטה לגרף מכוון עם קשתות בעלות משקלות שליליים, שעבורו מפיק האלגוריתם של דייקסטרה תשובות שגויות. מדוע ההוכחה של משפט 25.10 אינה פועלת כאשר מרשים קשתות בעלות משקלות שליליים?

25.2-3

נניח שאנו מחליפים את שורה 4 באלגוריתם של דייקסטרה בשורה הבאה:

4 **while** $|Q| > 1$

שינוי זה גורם לכך שלולאת ה־**while** תתבצע $|V| - 1$ פעמים במקום $|V|$ פעמים. האם האלגוריתם המתקבל נכון?

25.2-4

נתון גרף מכוון $G = (V, E)$ שבו לכל קשת $(u, v) \in E$ מצורף ערך $r(u, v)$, שהוא מספר ממשי בתחום $0 \le r(u, v) \le 1$, אשר מייצג את אמינותו של ערוץ תקשורת מן הקדקוד u אל הקדקוד v. אנו מפרשים את $r(u, v)$ כהסתברות שהערוץ מ־u אל v יהיה תקין, ומניחים שהסתברויות אלה הן בלתי תלויות. כתוב אלגוריתם יעיל למציאת המסלול האמין ביותר בין שני קדקודים נתונים.

25.2-5

יהי $G = (V, E)$ גרף מכוון משוקלל עם פונקציית משקל $w : E \to \{0, 1, \ldots, W - 1\}$, כאשר W הוא שלם אי־שלילי כלשהו. שנה את האלגוריתם של דייקסטרה כך שיחשב את המסלולים הקצרים ביותר מקדקוד מקור נתון s בזמן $O(WV + E)$.

25.2-6

שנה את האלגוריתם שכתבת בתרגיל 25.2-5 כך שירוץ בזמן $O((V + E) \lg W)$. (**רמז:** כמה אומדנים שונים של מסלולים קצרים ביותר יכולים להיות ב־$V - S$ ברגע כלשהו?)

25.3 אלגוריתם בלמן־פורד

אלגוריתם בלמן־פורד (Bellman-Ford algorithm) פותר את בעיית המסלולים הקצרים ביותר ממקור יחיד במקרה הכללי יותר, שבו משקלות של קשתות עשויים להיות שליליים. בהינתן גרף מכוון משוקלל $G = (V, E)$ עם מקור s ופונקציית משקל $w : E \to \mathbf{R}$, אלגוריתם בלמן־פורד מחזיר ערך בוליאני המציין אם קיים או לא קיים בגרף מעגל בעל משקל שלילי שניתן להגיע אליו מן המקור. אם קיים מעגל כזה, האלגוריתם מודיע שלא קיים פתרון לבעיה. אם לא קיים מעגל כזה, האלגוריתם יוצר את המסלולים הקצרים ביותר ואת משקליהם.

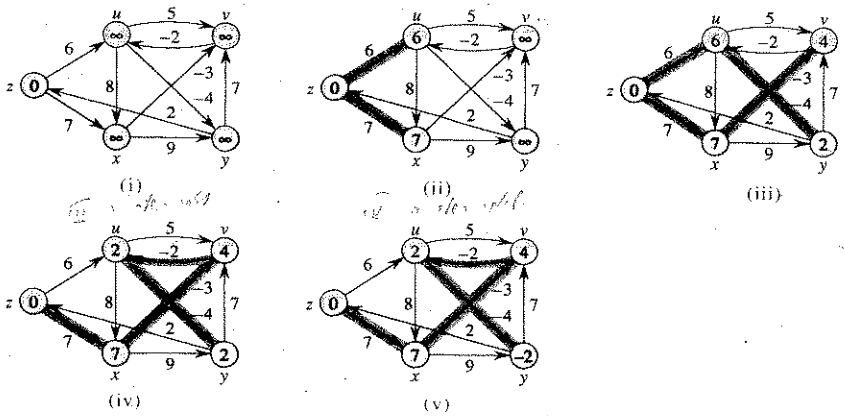
כמו האלגוריתם של דייקסטרה, גם אלגוריתם בלמן־פורד משתמש בטכניקת ההקלה, והוא הולך ומקטין אומדן $d[v]$ של המשקל של מסלול קצר ביותר מן המקור s אל כל קדקוד $v \in V$, עד שמתקבל המשקל האמיתי $\delta(s, v)$ של מסלול קצר ביותר מ־s אל v. האלגוריתם מחזיר TRUE אם ורק אם הגרף אינו מכיל מעגלים בעלי משקל שלילי שניתן להגיע אליהם מן המקור.

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i ← 1 to |V[G]| − 1
3       do for each edge (u, v) ∈ E[G]
4            do RELAX(u, v, w)
5  for each edge (u, v) ∈ E[G]
6       do if d[v] > d[u] + w(u, v)
7            then return FALSE
8  return TRUE
```

באיור 25.7 מודגמת פעולתו של אלגוריתם בלמן־פורד על גרף המכיל 5 קדקודים. לאחר ביצוע האתחול הרגיל, האלגוריתם מבצע $|V| - 1$ מעברים על קשתות הגרף. כל מעבר הוא איטרציה אחת של לולאת ה־**for** שבשורות 2-4 ומורכב מביצוע הקלה אחת על כל קשת בגרף. אירוים 25.7(ii)-(v) מראים את מצב האלגוריתם לאחר כל אחד מארבעת המעברים על הקשתות. לאחר השלמת $|V| - 1$ מעברים, שורות 5-8 בודקות אם קיים מעגל בעל משקל שלילי ומחזירות את הערך הבוליאני המתאים. (בהמשך נראה מדוע בדיקה זו פועלת היטב.)

זמן ריצתו של אלגוריתם בלמן־פורד הוא $O(VE)$, שכן האתחול בשורה 1 מתבצע בזמן $\Theta(V)$, כל אחד מ־$|V| - 1$ המעברים על הקשתות בשורות 2-4 מתבצע בזמן $O(E)$, ולולאת ה־**for** בשורות 5-7 מתבצעת בזמן $O(E)$.

כדי להוכיח את נכונותו של אלגוריתם בלמן־פורד, נתחיל בכך שנראה כי אם לא קיימים מעגלים בעלי משקל שלילי, אזי האלגוריתם מחשב משקלים נכונים של מסלולים קצרים ביותר אל כל הקדקודים שניתן להגיע אליהם מן המקור. בהוכחת למה זו טמון המפתח להבנת האינטואיציה שמאחורי האלגוריתם.



איור 25.7 ביצועו של אלגוריתם בלמן־פורד. המקור הוא הקדקוד z. ערכי d רשומים בתוך הקדקודים, וקשתות מוצללות מציינות ערכי π. בדוגמה מסוימת זו, כל מעבר מבצע הקלת קשתות בסדר לקסיקוגרפי (מימין לשמאל): (t, x), (t, y), (t, z), (u, v), (u, x), (u, y), (v, u), (x, v), (x, y), (y, v), (y, z), (z, u), ו־(z, x). **(i)** המצב ממש לפני המעבר הראשון על הקשתות. **(ii)-(v)** המצב לאחר כל אחד מן המעברים העוקבים על הקשתות. ערכי d ו־π המופיעים בחלק **(v)** הם הערכים הסופיים. בדוגמה זו מחזיר אלגוריתם בלמן־פורד את הערך TRUE.

למה 25.12

יהי $G = (V, E)$ גרף מכוון משוקלל עם מקור s ופונקציית משקל $w : E \to \mathbf{R}$, ונניח כי G אינו מכיל מעגלים בעלי משקל שלילי שניתן להגיע אליהם מ־s. אזי, עם עצירת האלגוריתם BELLMAN-FORD, $d[v] = \delta(s, v)$ עבור כל הקדקודים v שניתן להגיע אליהם מ־s.

הוכחה יהי v קדקוד שניתן להגיע אליו מ־s, ויהי $p = \langle v_0, v_1, \ldots, v_k \rangle$ מסלול קצר ביותר מ־s אל v, כאשר $v_0 = s$ ו־$v_k = v$. המסלול p הוא פשוט, ולכן $k \leq |V| - 1$. ברצוננו להוכיח באינדוקציה שעבור $i = 1, 2, \ldots, k$, מתקיים $d[v_i] = \delta(s, v_i)$ לאחר המעבר ה־i על הקשתות של G, ושוויון זה נשמר משלב זה ואילך. מכיוון שהאלגוריתם מבצע $|V| - 1$ מעברים, די בהוכחת טענה זו כדי להוכיח את הלמה.

עבור מקרה הבסיס, לאחר האתחול מתקיים $d[v_0] = \delta(s, v_0) = 0$, ועל פי למה 25.5, שוויון זה נשמר משלב זה ואילך.

עבור צעד האינדוקציה, נניח כי $d[v_{i-1}] = \delta(s, v_{i-1})$ לאחר המעבר ה־$i - 1$. במהלך המעבר ה־i מתבצעת הקלה של הקשת (v_{i-1}, v_i), ולכן על פי למה 25.7, נסיק כי לאחר המעבר ה־i ובכל זמן נתון לאחר מכן, $d[v_i] = \delta(s, v_i)$. בכך נשלמה הוכחת הלמה. ■

מסקנה 25.13

יהי $G = (V, E)$ גרף מכוון משוקלל עם קדקוד מקור s ופונקציית משקל $w : E \to \mathbf{R}$. אזי, עבור כל קדקוד $v \in V$, קיים מסלול מ־s אל v אם ורק אם אלגוריתם BELLMAN-FORD עוצר עם $d[v] < \infty$ כאשר הוא מורץ על G.

הוכחה ההוכחה דומה להוכחת למה 25.12 ואנו מותירים אותה כתרגיל לקורא (תרגיל 25.3-2). ■

משפט 25.14 (נכונותו של אלגוריתם בלמן־פורד)
נניח שמריצים את BELLMAN-FORD על גרף מכוון משוקלל $G = (V, E)$ עם מקור s ופונקציית משקל $w : E \to \mathbf{R}$. אם G אינו מכיל מעגלים בעלי משקל שלילי שניתן להגיע אליהם מ־s, אזי האלגוריתם מחזיר TRUE, מתקיים $d[v] = \delta(s, v)$ עבור כל הקדקודים $v \in V$, ותת־גרף הקודמים G_π הוא עץ מסלולים קצרים ביותר המושרש ב־s. אם G מכיל מעגלים בעלי משקל שלילי שניתן להגיע אליהם מ־s, אזי האלגוריתם מחזיר FALSE.

הוכחה נניח שהגרף G אינו מכיל מעגלים בעלי משקל שלילי שניתן להגיע אליהם מן המקור s. תחילה נוכיח את הטענה שעם עצירת האלגוריתם, $d[v] = \delta(s, v)$ עבור כל הקדקודים $v \in V$. אם v הוא קדקוד שניתן להגיע אליו מ־s, אזי למה 25.12 מוכיחה טענה זו. אם לא ניתן להגיע אל v מ־s, אזי הטענה נובעת ממסקנה 25.6. בכך הוכחה הטענה. מטענה זו ומלמה 25.9 נובע כי G הוא עץ מסלולים קצרים ביותר. עתה נשתמש בטענה זו כדי להראות כי BELLMAN-FORD מחזיר TRUE. בעת עצירת האלגוריתם, עבור כל הקשתות $(u, v) \in E$ מתקיים:

$$\begin{aligned} d[v] &= \delta(s, v) \\ &\le \delta(s, u) + w(u, v) \qquad \text{(על פי למה 25.3)} \\ &= d[u] + w(u, v) \end{aligned}$$

ולפיכך אף אחת מן הבדיקות שבשורה 6 אינה גורמת ל־BELLMAN-FORD להחזיר FALSE. האלגוריתם מחזיר אפוא TRUE.

עתה, בכיוון ההפוך, נניח שהגרף G מכיל מעגל בעל משקל שלילי $c = \langle v_0, v_1, \ldots, v_k \rangle$, כאשר $v_0 = v_k$, אשר ניתן להגיע אליו מן המקור s. אזי:

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0 \qquad (25.3)$$

נניח בשלילה שאלגוריתם בלמן־פורד מחזיר TRUE. לפיכך, $d[v_i] \le d[v_{i-1}] + w(v_{i-1}, v_i)$ עבור $i = 1, 2, \ldots, k$. נסכם את האי־שוויונים סביב המעגל c ונקבל:

$$\sum_{i=1}^{k} d[v_i] \le \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

כמו בהוכחת למה 25.8, כל קדקוד ב־c מופיע בדיוק פעם אחת בכל אחד משני הסכומים הראשונים. לכן:

$$\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$$

יתר על כן, על פי מסקנה 25.13, $d[v_i]$ הוא בעל ערך סופי עבור $i = 1, 2, \ldots, k$. לכן:

$$0 \le \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

בסתירה לאי־שוויון (25.3). נסיק כי אלגוריתם בלמן־פורד מחזיר TRUE אם הגרף G אינו מכיל מעגלים בעלי משקל שלילי שניתן להגיע אליהם מן המקור, ו־FALSE אחרת. ■

תרגילים

25.3-1
הרץ את אלגוריתם בלמן־פורד על הגרף המכוון מאיור 25.7, עם הקדקוד y כמקור. בצע הקלת קשתות בסדר לקסיקוגרפי בכל מעבר, והצג את ערכי d ו־π לאחר כל מעבר. עתה, שנה את משקל הקשת (y, v) ל־4, והרץ שוב את האלגוריתם, עם z כמקור.

25.3-2
הוכח את מסקנה 25.13.

25.3-3
בהינתן גרף מכוון משוקלל $G = (V, E)$ שאינו מכיל מעגלים בעלי משקל שלילי, יהי m המקסימום, על כל זוגות הקדקודים $u, v \in V$, של המספר המינימלי של קשתות במסלול קצר ביותר מ־u אל v. (כאן, מסלול קצר ביותר נקבע על פי משקלות ולא על פי מספר הקשתות.) הצע שינוי פשוט באלגוריתם בלמן־פורד שיאפשר לו לעצור לאחר $m + 1$ מעברים.

25.3-4
שנה את אלגוריתם בלמן־פורד כך שיציב $-\infty$ ב־$d[v]$ בכל קדקוד v שעבורו קיים מעגל בעל משקל שלילי על מסלול כלשהו מן המקור אל v.

25.3-5
יהי $G = (V, E)$ גרף מכוון משוקלל עם פונקציית משקל $w : E \to \mathbf{R}$. כתוב אלגוריתם שזמן ריצתו $O(VE)$, אשר מוצא, עבור כל קדקוד $v \in V$, את הערך $\delta^*(v) = \min_{u \in V}\{\delta(u, v)\}$.

*** 25.3-6**
נניח שגרף מכוון משוקלל $G = (V, E)$ מכיל מעגל בעל משקל שלילי. כתוב אלגוריתם יעיל המחזיר את רשימת הקדקודים על מעגל אחד כזה. הוכח את נכונותו של האלגוריתם שלך.

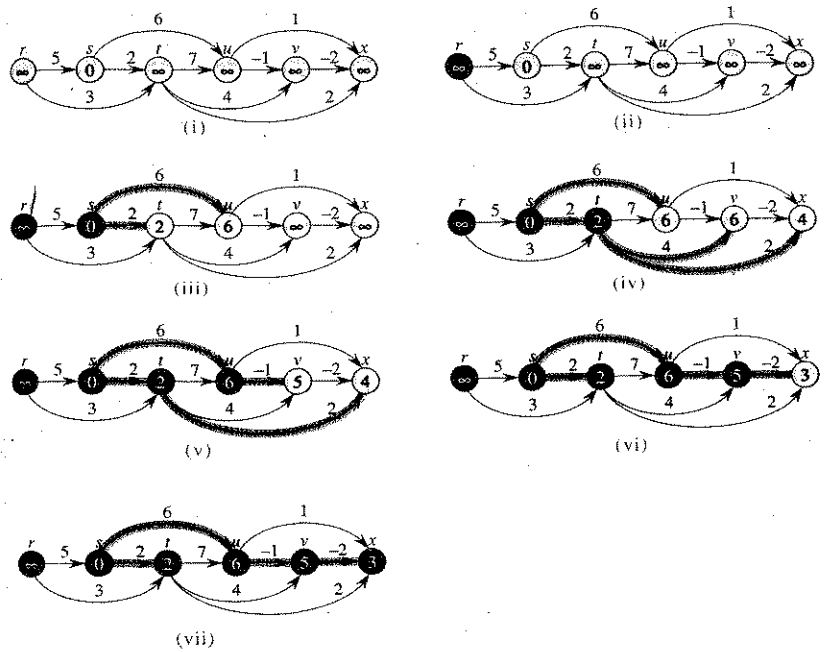
25.4 מסלולים קצרים ביותר ממקור יחיד בגרף מכוון ללא מעגלים

על־ידי הקלת הקשתות של גמ"ל (גרף מכוון ללא מעגלים) משוקלל $G = (V, E)$ על־פי מיון טופולוגי של קדקודיו, ניתן לחשב מסלולים קצרים ביותר ממקור יחיד בזמן $\Theta(V + E)$. מסלולים קצרים ביותר תמיד מוגדרים היטב בגמ"ל, שכן גם אם קיימות בו קשתות בעלות משקל שלילי, לא ייתכן שיהיו בו מעגלים בעלי משקל שלילי.

האלגוריתם מתחיל במיון טופולוגי של הגמ"ל (ראה סעיף 23.4), כדי לכפות על הקדקודים סדר ליניארי. אם קיים מסלול מקדקוד u אל קדקוד v, אזי u יבוא לפני v בסדר הטופולוגי. אנו מבצעים מעבר אחד בלבד על הקדקודים, בסדר שנקבע על פי המיון הטופולוגי. במהלך עיבודו של כל קדקוד מתבצעת הקלה על כל הקשתות היוצאות מקדקוד זה.

```
DAG-SHORTEST-PATHS(G, w, s)
1  topologically sort the vertices of G
2  INITIALIZE-SINGLE-SOURCE(G, s)
3  for each vertex u taken in topologically sorted order
4      do for each edge v ∈ Adj[u]
5          do RELAX(u, v, w)
```

באיור 25.8 מודגמת פעולתו של אלגוריתם זה.



איור 25.8 ביצועו של האלגוריתם למציאת מסלולים קצרים ביותר בגרף מכוון ללא מעגלים. הקדקודים ממוינים מיון טופולוגי משמאל לימין. קדקוד המקור הוא s. בתוך הקדקודים רשומים ערכי d, וקשתות מוצללות מציינות ערכי π. **(i)** המצב לפני האיטרציה הראשונה של לולאת ה־**for** שבשורות 3-5. **(ii)-(vii)** המצב לאחר כל איטרציה נוספת של לולאת ה־**for** שבשורות 3-5. בכל חלק באיור, הקדקוד המושחר החדש הוא זה ששימש כ־u באותה איטרציה. הערכים המופיעים בחלק (vii) הם הערכים הסופיים.

זמן הריצה של אלגוריתם זה נקבע על־ידי שורה 1 ועל־ידי לולאת ה־**for** שבשורות 3-5. כפי שהראינו בסעיף 23.4, את המיון הטופולוגי ניתן לבצע בזמן $\Theta(V+E)$. בלולאת ה־**for** שבשורות 3-5, כמו באלגוריתם של דייקסטרה, מתבצעת איטרציה אחת לכל קדקוד. עבור כל קדקוד, כל אחת מן הקשתות היוצאות ממנו נבחנת בדיוק פעם אחת. אבל בניגוד לאלגוריתם של דייקסטרה, באלגוריתם זה איננו משתמשים בתור קדימויות, ולכן הטיפול בכל קשת צורך זמן של $O(1)$ בלבד. זמן הריצה הוא אפוא $\Theta(V+E)$, דהיינו, לינארי בגודל הייצוג של הגרף על־ידי רשימות סמיכות.

המשפט הבא מראה שהשגרה DAG-SHORTEST-PATHS מחשבת נכונה את המסלולים הקצרים ביותר.

משפט 25.15

אם גרף מכוון משוקלל $G=(V,E)$ מכיל קדקוד מקור s ואינו מכיל מעגלים, אזי עם עצירת השגרה DAG-SHORTEST-PATHS, $d[v]=\delta(s,v)$ עבור כל הקדקודים $v\in V$, ותת־גרף הקודמים G_π הוא עץ מסלולים קצרים ביותר.

הוכחה תחילה נראה שבעת העצירה מתקיים $d[v]=\delta(s,v)$ עבור כל הקדקודים $v\in V$. אם v אינו ניתן להגעה מ־s, אזי על פי מסקנה 25.6, $d[v]=\delta(s,v)=\infty$. עתה, נניח ש־v ניתן להגעה מ־s, כך שקיים מסלול קצר ביותר $p=\langle v_0,v_1,\ldots,v_k\rangle$, כאשר $v_0=s$ ו־$v_k=v$. מכיוון שאנו מעבדים את הקדקודים בסדר שנקבע על־ידי מיון טופולוגי, ההקלה של הקשתות על p מתבצעת בסדר הבא (משמאל לימין): $(v_0,v_1),(v_1,v_2),\ldots,(v_{k-1},v_k)$. אינדוקציה פשוטה, תוך שימוש בלמה 25.7 (כמו בהוכחת למה 25.12), מראה שבעת העצירה, $d[v_i]=\delta(s,v_i)$ עבור $i=0,1,\ldots,k$. לבסוף, על פי למה 25.9, G_π הוא עץ מסלולים קצרים ביותר. ■

יישום מעניין של אלגוריתם זה מופיע בקביעת מסלולים קריטיים בניתוח **דיאגרמות PERT**.[2] קשתות מייצגות משימות שיש לבצע, ומשקלות הקשתות מייצגים את הזמנים הדרושים לביצוע המשימות. אם קשת (u,v) נכנסת לקדקוד v וקשת (v,x) יוצאת מ־v, אזי יש לבצע את המשימה (u,v) לפני המשימה (v,x). מסלול בגמ"ל זה מייצג סדרת משימות שחייבות להתבצע בסדר מסוים. **מסלול קריטי** (critical path) הוא מסלול *ארוך ביותר* בגמ"ל והוא מייצג את פרק הזמן הארוך ביותר הנדרש לביצוע סדרה סדורה של משימות. משקלו של מסלול קריטי מהווה חסם תחתון על הזמן הכולל הנדרש לביצוע כל המשימות. ניתן למצוא מסלול קריטי באחת משתי דרכים:

- היפוך הסימנים של משקלות הקשתות והרצת DAG-SHORTEST-PATHS, או
- הרצת האלגוריתם DAG-SHORTEST-PATHS, תוך החלפת "∞" ב־"$-\infty$" בשורה 2 של השגרה INITIALIZE- SINGLE-SOURCE והחלפת "$<$" ב־"$>$" בשגרה RELAX.

תרגילים

25.4-1

הרץ את DAG-SHORTEST-PATHS על הגרף המכוון מאיור 25.8 עם הקדקוד r כמקור.

25.4-2

נניח שנשנה את שורה 3 של DAG-SHORTEST-PATHS באופן הבא:

3 **for** the first $|V|-1$ vertices, taken in topologically sorted order

הראה שהשגרה המתקבלת עדיין נכונה.

25.4-3

התיאור של דיאגרמות PERT שהובא לעיל הוא בלתי טבעי במידה מסוימת. טבעי יותר שהקדקודים ייצגו משימות, וקשתות ייצגו אילוצים על סדר ביצוע המשימות; כלומר, קשת (u,v) תציין שהמשימה u חייבת להתבצע לפני המשימה v. בייצוג כזה מגדירים משקלות לקדקודים ולא לקשתות. שנה את השגרה DAG-SHORTEST-PATHS כך שתמצא בזמן לינארי מסלול ארוך ביותר בגרף מכוון ללא מעגלים עם משקלות על הקדקודים.

25.4-4

כתוב אלגוריתם יעיל למניית המספר הכולל של מסלולים בגרף מכוון ללא מעגלים. נתח את האלגוריתם שלך ודון במעשיותו.

---

2. "PERT" הן ראשי התיבות של "program evaluation and review technique" — שיטה לסקירת תכנונים ולהערכתם.

25.5 אילוצי הפרשים ומסלולים קצרים ביותר

במקרה הכללי של בעיית תכנון לינארי, אנו מעוניינים למצוא ערך אופטימלי של פונקציה לינארית, בכפוף לקבוצת אי־שוויונים לינאריים. בסעיף זה נחקור מקרה פרטי של תכנון לינארי, אשר ניתן לעשות לו רדוקציה למציאת מסלולים קצרים ביותר ממקור יחיד. את בעיית המסלולים הקצרים ביותר ממקור יחיד המתקבלת ניתן לפתור באמצעות אלגוריתם בלמן־פורד, ובכך לפתור גם את בעיית התכנון הלינארי המקורית.

תכנון לינארי

במקרה הכללי של **בעיית תכנון לינארי** (linear-programming problem), נתונים לנו מטריצה A שממדיה $n \times m$, וקטור b באורך m, וקטור c באורך n. ברצוננו למצוא וקטור x בן n איברים, שעבורו **פונקציית המטרה** $\sum_{i=1}^{n} c_i x_i$ מקבלת ערך מקסימלי, בכפוף ל־m האילוצים הנתונים על־ידי $Ax \leq b$.

קיימות בעיות רבות שניתן לנסח אותן כבעיות תכנון לינאריות, ובשל כך הושקעה עבודה רבה בפיתוח אלגוריתמים עבור תכנון לינארי. **אלגוריתם הסימפלקס**[3] (simplex algorithm) פותר באופן מעשי בעיות לינאריות כלליות במהירות רבה. אולם ניתן להרכיב קלטים מתוחכמים שעבורם שיטת הסימפלקס יכולה לדרוש זמן אקספוננציאלי. בעיות לינאריות כלליות ניתן לפתור בזמן פולינומיאלי באמצעות **אלגוריתם האליפסואיד** (ellipsoid algorithm), אשר ביישומים מעשיים מתבצע באיטיות, או באמצעות **האלגוריתם של קרמארקר** (Karmarkar's algorithm), אשר ביצועיו ביישומים מעשיים מתחרים לעתים קרובות באלה של שיטת הסימפלקס.

בשל ההשקעה המתמטית הנדרשת להבנת אלגוריתמים אלה ולניתוחם, בחרנו שלא לעסוק כאן בפירוט באלגוריתמים הפותרים את בעיית התכנון הלינארי הכללית. עם זאת, חשוב להבין את המבנה של בעיות תכנון לינארי, וזאת בשל סיבות מספר. ראשית, אם אנו יודעים שבעיה נתונה ניתנת לניסוח כבעיית תכנון לינארי בגודל פולינומיאלי, הרי שמכך נובע מיד שקיים עבור בעיה זו אלגוריתם שזמן ריצתו פולינומיאלי. שנית, קיימים מקרים פרטיים רבים של תכנון לינארי אשר עבורם קיימים אלגוריתמים מהירים יותר. לדוגמה, כפי שנראה בסעיף זה, בעיית המסלולים הקצרים ביותר ממקור יחיד היא מקרה פרטי של תכנון לינארי. בעיות אחרות שניתן לנסחן כבעיות תכנון לינארי כוללות את הבעיה של מציאת מסלול קצר ביותר בין זוג קדקודים (תרגיל 25.5-4) ואת בעיית הזרימה המקסימלית (תרגיל 27.1-8).

לעתים, ערכה של פונקציית המטרה אינו מעניין אותנו כלל; ברצוננו למצוא רק **פתרון אפשרי** כלשהו, דהיינו, וקטור x המקיים $Ax \leq b$, או לקבוע שלא קיים פתרון כזה. אנו נתמקד ב**בעיית אפשרות** אחת כזאת.

3. אלגוריתם הסימפלקס מוצא פתרון אופטימלי לבעיות תכנון לינארי על־ידי כך שהוא בוחן סדרת נקודות בתחום האפשרי — התחום במרחב ה־n־ממדי המקיים $Ax \leq b$. האלגוריתם מבוסס על העובדה שפתרון אשר עבורו פונקציית המטרה מקבלת ערך מקסימלי בתחום האפשרי, מתקבל באיזושהי "נקודת קיצון" או "פינה" של התחום האפשרי. אלגוריתם הסימפלקס עובר מפינה לפינה בתחום האפשרי, עד אשר לא ניתן יותר לשפר את ערכה של פונקציית המטרה. "סימפלקס" הוא קמור (ראה סעיף 35.3) של $d+1$ נקודות במרחב ה־d־ממדי (כמו למשל, משולש במישור או טטרהדרון (פירמידה משולשת) במרחב התלת־ממדי). על פי Danzig [53], ניתן לראות את פעולת המעבר מפינה אחת לאחרת כפעולה על סימפלקס הנגזר מפירוש "דואלי" של בעיית התכנון הלינארי — ומכאן השם "שיטת הסימפלקס".

מערכות של אילוצי הפרשים

במערכת של אילוצי הפרשים (system of difference constraints), כל שורה במטריצת התכנון הלינארי A מכילה 1 אחד ו־-1 אחד, וכל שאר התאים ב־A מכילים 0. לכן, האילוצים הנתונים על ידי $Ax \leq b$ הם מערכת של m **אילוצי הפרשים** ב־n נעלמים, כאשר כל אילוץ הוא אי־שוויון לינארי פשוט מן הצורה:

$$x_j - x_i \leq b_k$$

כאשר $1 \leq i, j \leq n$ ו־$1 \leq k \leq m$.

לדוגמה, נתבונן בבעיה של מציאת וקטור באורך 5, $x = (x_i)$, המקיים:

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{pmatrix}.$$

בעיה זו שקולה לבעיה של מציאת הנעלמים x_i עבור $i = 1, 2, \ldots, 5$, אשר מקיימים את שמונת אילוצי ההפרשים הבאים:

$$\begin{aligned} x_1 - x_2 &\leq 0 \\ x_1 - x_5 &\leq -1 \\ x_2 - x_5 &\leq 1 \\ x_3 - x_1 &\leq 5 \\ x_4 - x_1 &\leq 4 \\ x_4 - x_3 &\leq -1 \\ x_5 - x_3 &\leq -3 \\ x_5 - x_4 &\leq -3 \end{aligned} \tag{25.4}$$

אחד הפתרונות לבעיה זו הוא $x = (-5, -3, 0, -1, -4)$, כפי שניתן לוודא ישירות על־ידי בדיקת כל אחד מהאי־שוויונים. למעשה, אין זה הפתרון היחיד לבעיה. פתרון אחר הוא $x' = (0, 2, 5, 4, 1)$. קיים קשר בין שני הפתרונות הללו: כל רכיב ב־x' גדול ב־5 מהרכיב המתאים ב־x. קשר זה אינו מקרי.

למה 25.16

יהי $x = \langle x_1, x_2, \ldots, x_n \rangle$ פתרון למערכת $Ax \leq b$ של אילוצי הפרשים, ויהי d קבוע כלשהו. אזי, $x + d = (x_1 + d, x_2 + d, \ldots, x_n + d)$ גם הוא פתרון ל־$Ax \leq b$.

הוכחה עבור כל x_i ו־x_j מתקיים $(x_j + d) - (x_i + d) = x_j - x_i$. לכן, אם x מקיים את $Ax \leq b$, כך גם $x + d$. ■

מערכות של אילוצי הפרשים מופיעות ביישומים רבים ושונים. לדוגמה, הנעלמים x_i עשויים לייצג זמנים שבהם אמורים להתרחש מאורעות. ניתן לראות כל אילוץ כדרישה שמאורע מסוים לא יתרחש זמן רב מדי לאחר מאורע אחר. המאורעות עשויים להיות, למשל, עבודות שיש לבצע במהלך בניית בית. אם חפירת היסודות מתחילה בזמן x_1 ואורכת שלושה ימים, ויציקת הבטון לתוך היסודות מתחילה בזמן x_2, רצוי שיתקיים $x_2 \geq x_1 + 3$, או, באופן שקול, $x_1 - x_2 \leq -3$. כלומר, את האילוץ הנוגע לתזמון היחסי ניתן לבטא כאילוץ של הפרש.

גרפי אילוצים

יש תועלת בפירוש מערכות של אילוצי הפרשים מנקודת מבט של תורת הגרפים. הרעיון הוא שבמערכת $Ax \leq b$ של אילוצי הפרשים, ניתן לראות את מטריצת התכנון הלינארי A, שממדיה $n \times m$, כמטריצת הקשירויות (ראה תרגיל 23.1-7) של גרף בעל n קדקודים ו־m קשתות. כל קדקוד v_i בגרף, עבור $i = 1, 2, \ldots, n$, מתאים לאחד מ־n הנעלמים x_i. כל קשת מכוונת בגרף מתאימה לאחד מ־m האי־שוויונים, אשר בכל אחד מהם מופיעים שני נעלמים.

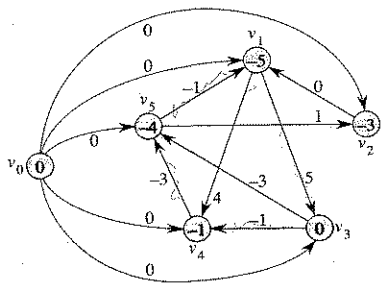
בניסוח פורמלי יותר, בהינתן מערכת $Ax \leq b$ של אילוצי הפרשים, **גרף האילוצים** (constraint graph) המתאים הוא גרף מכוון משוקלל $G = (V, E)$, אשר בו:

$$V = \{v_0, v_1, \ldots, v_n\}$$

וכן:

$$E = \{(v_i, v_j) : \text{הוא אילוץ } x_j - x_i \leq b_k\} \cup \{(v_0, v_1), (v_0, v_2), (v_0, v_3), \ldots, (v_0, v_n)\}$$

כפי שנראה מיד, הקדקוד v_0 מצורף לגרף כדי להבטיח שכל קדקוד אחר בגרף ניתן להגעה מקדקוד זה. אם כן, קבוצת הקדקודים V מורכבת מקדקוד v_i עבור כל נעלם x_i, ועוד קדקוד נוסף v_0. קבוצת הקשתות E מכילה קשת אחת עבור כל אחד מאילוצי ההפרשים, ועוד קשת (v_0, v_1) עבור כל נעלם x_i. אם $x_j - x_i \leq b_k$ הוא אחד מאילוצי ההפרשים, אזי משקלה של הקשת (v_i, v_j) הוא $w(v_i, v_j) = b_k$. משקלה של כל קשת היוצאת מ־v_0 הוא 0. באיור 25.9 מוצג גרף האילוצים עבור מערכת אילוצי ההפרשים (25.4).



איור 25.9 גרף האילוצים המתאים למערכת אילוצי ההפרשים (25.4). בתוך כל קדקוד v_i רשום הערך $\delta(v_0, v_i)$. $x = (-5, -3, 0, -1, -4)$ הוא פתרון אפשרי למערכת.

המשפט הבא מראה שניתן לפתור מערכת אילוצי הפרשים על־ידי מציאת משקליהם של מסלולים קצרים ביותר בגרף האילוצים המתאים.

משפט 25.17

בהינתן מערכת $Ax \leq b$ של אילוצי הפרשים, יהי $G = (V, E)$ גרף האילוצים המתאים. אם G אינו מכיל מעגלים בעלי משקל שלילי, אזי:

$$x = (\delta(v_0, v_1), \delta(v_0, v_2), \delta(v_0, v_3), \ldots, \delta(v_0, v_n)) \tag{25.5}$$

הוא פתרון אפשרי של המערכת. אם G מכיל מעגל בעל משקל שלילי, אזי לא קיים עבור המערכת פתרון אפשרי.

הוכחה תחילה נראה שאם גרף האילוצים אינו מכיל מעגלים בעלי משקל שלילי, אזי משוואה (25.5) נותנת פתרון אפשרי. נתבונן בקשת כלשהי $(v_i, v_j) \in E$. על פי למה 25.3, $\delta(v_0, v_j) \leq \delta(v_0, v_i) + w(v_i, v_j)$, או, באופן שקול, $\delta(v_0, v_j) - \delta(v_0, v_i) \leq w(v_i, v_j)$. לכן, אם נציב $x_i = \delta(v_0, v_i)$ ו־$x_j = \delta(v_0, v_j)$ מקיימים את אילוץ ההפרשים $x_j - x_i \leq w(v_i, v_j)$ המתאים לקשת (v_i, v_j).

עתה נראה שאם גרף האילוצים מכיל מעגל בעל משקל שלילי, אזי לא קיים פתרון אפשרי למערכת אילוצי ההפרשים. נניח, בלי הגבלת הכלליות, שהמעגל שמשקלו שלילי הוא $c = \langle v_1, v_2, \ldots, v_k \rangle$, כאשר $v_1 = v_k$. (הקדקוד v_0 אינו יכול להשתייך למעגל c מכיוון שלא קיימות קשתות הנכנסות אליו.) המעגל c מתאים למערכת אילוצי ההפרשים הבאה:

$$\begin{aligned}
x_2 - x_1 &\leq w(v_1, v_2) \\
x_3 - x_2 &\leq w(v_2, v_3) \\
&\vdots \\
x_k - x_{k-1} &\leq w(v_{k-1}, v_k) \\
x_1 - x_k &\leq w(v_k, v_1)
\end{aligned}$$

מכיוון שכל פתרון עבור x חייב לקיים את k האי־שוויונים הללו, הרי שכל פתרון חייב לקיים גם את האי־שוויון המתקבל כאשר מסכמים את כולם. כאשר מסכמים את כל האגפים השמאליים של האי־שוויונים, מחברים כל נעלם x_i פעם אחת ומחסרים אותו פעם אחת, כך שסכום האגפים השמאליים הוא 0. סכום האגפים הימניים הוא $w(c)$, ולכן מקבלים $0 \leq w(c)$. אולם, מכיוון ש־c הוא מעגל בעל משקל שלילי, $w(c) < 0$, ומכאן שכל פתרון עבור x חייב לקיים $0 \leq w(c) < 0$, וזה בלתי אפשרי. ■

פתרון מערכות של אילוצי הפרשים

משפט 25.17 מראה לנו שניתן להשתמש באלגוריתם בלמן־פורד כדי לפתור מערכת של אילוצי הפרשים. מכיוון שקיימות קשתות מקדקוד המקור v_0 אל כל הקדקודים האחרים בגרף האילוצים, הרי שכל מעגל בעל משקל שלילי בגרף ניתן להגעה מ־v_0. אם אלגוריתם בלמן־פורד מחזיר TRUE, אזי משקלי המסלולים הקצרים ביותר שהוא מחשב מהווים פתרון אפשרי למערכת. באיור 25.9, לדוגמה, משקלי המסלולים הקצרים ביותר נותנים את הפתרון האפשרי $x = (d - 5, d - 3, d, d - 1, d - 4)$, ועל פי למה 25.16, $x = (-5, -3, 0, -1, -4)$ גם הוא פתרון אפשרי, עבור קבוע d כלשהו. אם אלגוריתם בלמן־פורד מחזיר FALSE, אזי לא קיים פתרון אפשרי למערכת אילוצי ההפרשים.

מערכת של אילוצי הפרשים המכילה m אילוצים ב־n נעלמים יוצרת גרף המכיל $n + 1$ קדקודים ו־$m + n$ קשתות. לכן, אם משתמשים באלגוריתם בלמן־פורד, ניתן לפתור את המערכת בזמן $O((n + 1)(n + m)) = O(n^2 + nm)$. בתרגיל 25.5-5 תתבקש להראות שזמן ריצתו של האלגוריתם הוא למעשה $O(nm)$, גם כאשר m קטן בהרבה מ־n.

תרגילים

25.5-1

מצא פתרון אפשרי, או קבע שפתרון כזה אינו קיים, עבור מערכת אילוצי ההפרשים הבאה:

$$\begin{aligned}
x_1 - x_2 &\le 1\\
x_1 - x_4 &\le -4\\
x_2 - x_3 &\le 2\\
x_2 - x_5 &\le 7\\
x_2 - x_6 &\le 5\\
x_3 - x_6 &\le 10\\
x_4 - x_2 &\le 2\\
x_5 - x_1 &\le -1\\
x_5 - x_4 &\le 3\\
x_6 - x_3 &\le -8
\end{aligned}$$

25.5-2

מצא פתרון אפשרי, או קבע שפתרון כזה אינו קיים, עבור מערכת אילוצי ההפרשים הבאה:

$$\begin{aligned}
x_1 - x_2 &\le 4\\
x_1 - x_5 &\le 5\\
x_2 - x_4 &\le -6\\
x_3 - x_2 &\le 1\\
x_4 - x_1 &\le 3\\
x_4 - x_3 &\le 5\\
x_4 - x_5 &\le 10\\
x_5 - x_3 &\le -4\\
x_5 - x_4 &\le -8
\end{aligned}$$

25.5-3

האם משקלו של מסלול קצר ביותר כלשהו מהקדקוד החדש v_0 בגרף אילוצים יכול להיות חיובי? הסבר.

25.5-4

בטא את בעיית המסלול הקצר ביותר בין זוג קדקודים אחד כבעיה בתכנון לינארי.

25.5-5

הראה כיצד לשנות מעט את אלגוריתם בלמן־פורד כך שכאשר הוא משמש לפתרון מערכת אילוצי הפרשים בת m אי־שוויונים ב־n נעלמים, זמן הריצה הוא $O(nm)$.

25.5-6

הראה כיצד ניתן לפתור מערכת אילוצי הפרשים באמצעות אלגוריתם דמוי אלגוריתם בלמן־פורד אשר פועל על גרף אילוצים שאינו כולל את הקדקוד הנוסף v_0.

25.5-7 *

תהי $Ax \le b$ מערכת של m אילוצי הפרשים ב־n נעלמים. הראה שכאשר מריצים את אלגוריתם בלמן־פורד על גרף האילוצים המתאים, הוא מחזיר פתרון שעבורו הסכום $\sum_{i=1}^{n} x_i$ הוא בעל ערך מקסימלי בכפוף לאילוצים $Ax \le b$ ו־$x_i \le 0$ עבור כל x_i.

25.5-8 *

הראה שכאשר מריצים את אלגוריתם בלמן־פורד על גרף האילוצים של מערכת אילוצי הפרשים $Ax \le b$, האלגוריתם ממזער את הגודל $\max\{x_i\} - \min\{x_i\}$, בכפוף ל־$Ax \le b$. הסבר כיצד ניתן לנצל עובדה זו כאשר משתמשים באלגוריתם זה לקביעת לוח זמנים לעבודות בנייה.

25.5-9

נניח שבבעיית תכנון לינארי $Ax \le b$ כל שורה במטריצה A מייצגת אילוץ הפרשים, אילוץ במשתנה אחד מן הצורה $x_i \le b_k$, או אילוץ במשתנה אחד מן הצורה $-x_i \le b_k$. הראה כיצד ניתן להתאים את אלגוריתם בלמן־פורד לפתרון גרסה זו של מערכת אילוצים.

25.5-10

נניח כי בנוסף למערכת אילוצי הפרשים, ברצוננו לטפל גם באילוצי שוויון מן הצורה $x_i = x_j + b_k$. הראה כיצד ניתן להתאים את אלגוריתם בלמן־פורד לפתרון גרסה זו של מערכת אילוצים.

25.5-11

כתוב אלגוריתם יעיל לפתרון מערכת $Ax \le b$ של אילוצי הפרשים אשר בה כל האיברים b הם מספרים ממשיים וכל הנעלמים x_i חייבים לקבל ערכים שלמים.

25.5-12 *

כתוב אלגוריתם יעיל לפתרון מערכת $Ax \le b$ של אילוצי הפרשים אשר בה כל האיברים b הם מספרים ממשיים, וחלק מן הנעלמים x_i, אולם לא בהכרח כולם, חייבים לקבל ערכים שלמים.

בעיות

25-1 השיפור של ין (Yen) לאלגוריתם בלמן־פורד

נניח שאנו מבצעים את הקלת הקשתות בכל מעבר של אלגוריתם בלמן־פורד בסדר המתואר להלן. לפני המעבר הראשון, אנו קובעים סדר לינארי שרירותי $v_1, v_2, \ldots, v_{|V|}$ על קדקודי גרף הקלט $G = (V, E)$. לאחר מכן אנו מחלקים את קבוצת הקשתות E ל־$E_f \cup E_b$, כאשר $E_f = \{(v_i, v_j) \in E : i < j\}$ ו־$E_b = \{(v_i, v_j) \in E : i > j\}$. אנו מגדירים $G_f = (V, E_f)$ ו־$G_b = (V, E_b)$.

א. הוכח כי G_f חסר מעגלים ובעל מיון טופולוגי $\langle v_1, v_2, \ldots, v_{|V|}\rangle$, וכי G_b חסר מעגלים ובעל מיון טופולוגי $\langle v_{|V|}, v_{|V|-1}, \ldots, v_1\rangle$.

נניח שאנו מממשים כל מעבר של אלגוריתם בלמן־פורד כמתואר להלן. אנו מבקרים בכל קדקוד על פי הסדר הבא (משמאל לימין): $v_1, v_2, \ldots, v_{|V|}$, ומבצעים הקלה על כל הקשתות של E_f היוצאות מן הקדקוד; לאחר מכן אנו מבקרים בכל קדקוד על פי הסדר הבא (משמאל לימין): $v_{|V|}, v_{|V|-1}, \ldots, v_1$, ומבצעים הקלה על הקשתות של E_b היוצאות מן הקדקוד.

ב. הוכח כי בשיטה זו, אם G אינו מכיל מעגלים בעלי משקל שלילי שניתן להגיע אליהם מקדקוד המקור s, אזי לאחר $\lceil |V|/2 \rceil$ מעברים בלבד על הקשתות, $d[v] = \delta(s, v)$ עבור כל הקדקודים $v \in V$.

ג. כיצד משפיעה שיטת מעברים זו על זמן הריצה של אלגוריתם בלמן־פורד?

25-2 קופסאות מקוננות

קופסה n־ממדית, שממדיה $(x_1, x_2, \ldots, x_d)$ **מקוננת** בתוך קופסה אחרת שממדיה $(y_1, y_2, \ldots, y_d)$ אם קיימת תמורה π של $\{1, 2, \ldots, d\}$ כך שמתקיים $x_{\pi(1)} < y_1, x_{\pi(2)} < y_2, \ldots, x_{\pi(d)} < y_d$.

א. הראה שיחס הקינון הוא טרנזיטיבי.

ב. תאר שיטה יעילה שבאמצעותה ניתן לקבוע אם קופסה d־ממדית אחת מקוננת באחרת או לא.

ג. נניח שנתונה קבוצה בת n קופסאות d־ממדיות $\{B_1, B_2, \ldots, B_n\}$. תאר אלגוריתם יעיל אשר מוצא את הסדרה הארוכה ביותר של קופסאות $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k} \rangle$ כך ש־B_{i_j} מקוננת ב־$B_{i_{j+1}}$ עבור $j = 1, 2, \ldots, k-1$. בטא את זמן הריצה של האלגוריתם שלך במונחי n ו־d.

25-3 ארביטרז'

ארביטרז' הוא השימוש בפערים בין שערי החליפין של מטבעות כדי להמיר יחידה אחת של מטבע מסוים ביותר מיחידה אחת של אותו מטבע. לדוגמה, נניח שהשער של דולר אחד (של ארה"ב) הוא 0.7 לישי"ט, השער של 1 לישי"ט הוא 9.5 פרנקים שוויצריים, ושערו של פרנק שוויצרי אחד הוא 0.16 דולרים. במקרה זה יכול סוחר, על ידי סדרה של המרות מטבע, לקנות באמצעות דולר אחד, $0.7 \times 9.5 \times 0.16 = 1.064$ דולרים, וכך ליצור רווח של 6.4 אחוזים.

נניח שנתונים לנו n מטבעות $c_1, c_2, \ldots, c_n$ וטבלת שערי חליפין R, שממדיה $n \times n$, כך שהשער של המטבע c_i הוא $R[i, j]$ יחידות של המטבע c_j.

א. תאר אלגוריתם יעיל אשר קובע אם יש או אין סדרת מטבעות $\langle c_{i_1}, c_{i_2}, \ldots, c_{i_k} \rangle$ המקיימת:

$$R[i_1, i_2] \cdot R[i_2, i_3] \cdot \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1] > 1$$

נתח את זמן הריצה של האלגוריתם שלך.

ב. תאר אלגוריתם יעיל אשר מדפיס סדרה כזאת, אם היא קיימת. נתח את זמן הריצה של האלגוריתם שלך.

25-4 האלגוריתם המדורג של גאבו (Gabow) למציאת מסלולים קצרים ביותר ממקור יחיד

אלגוריתם מדורג (scaling algorithm) פותר בעיה על־ידי כך שהוא בוחן בתחילה רק את הסיבית המשמעותית ביותר בכל ערך קלט רלוונטי (משקלה של קשת, למשל). לאחר מכן הוא מעדן את הפתרון ההתחלתי על ידי בחינת הסיביות השניות בחשיבותן. עם התקדמותו בוחן האלגוריתם עוד ועוד סיביות על פי סדר משמעות יורד, מעדן שוב ושוב את הפתרון, עד שהוא בוחן את כל הסיביות ומגיע לפתרון הנכון.

בבעיה זו, נבחן אלגוריתם אשר מחשב מסלולים קצרים ביותר ממקור יחיד על־ידי בחינה מדורגת כזאת של משקלות הקשתות. נתון לנו גרף מכוון $G = (V, E)$ עם משקלות קשתות w שהם שלמים אי־שליליים. יהי $W = \max_{(u, v) \in E} \{w(u, v)\}$. מטרתנו היא לפתח אלגוריתם שזמן ריצתו $O(E \lg W)$. אנו מניחים שכל הקדקודים ניתנים להגעה מן המקור.

האלגוריתם בוחן את סיביות הייצוג הבינרי של משקלות הקשתות, אחת בכל פעם, מן הסיבית המשמעותית ביותר ועד לסיבית הכי פחות משמעותית. ביתר פירוט, יהי $k = \lceil \lg(W + 1) \rceil$

מספר הסיביות בייצוג הבינרי של W, ויהי $w_i(u, v) = \lfloor w(u, v)/2^{k-i} \rfloor$, עבור $i = 1, 2, \ldots, k$. כלומר, $w_i(u, v)$ היא הגרסה "המקורבת" של $w(u, v)$ הנתונה על־ידי i הסיביות המשמעותיות ביותר ב־$w(u, v)$. (לפיכך, $w_k(u, v) = w(u, v)$ עבור כל הקשתות $(u, v) \in E$.) לדוגמה, אם $k = 5$ ו־$w(u, v) = 25$, אשר ייצוגו הבינרי הוא $\langle 11001 \rangle$, אזי $w_3(u, v) = \langle 110 \rangle = 6$. דוגמה אחרת עם $k = 5$: אם $w(u, v) = \langle 00100 \rangle = 4$, אזי $w_3(u, v) = \langle 001 \rangle = 1$. נגדיר את $\delta_i(u, v)$ כמשקל המסלול הקצר ביותר מקדקוד u אל קדקוד v, על פי פונקציית המשקל w_i. לפיכך, $\delta_k(u, v) = \delta(u, v)$ עבור כל $u, v \in V$. בהינתן קדקוד מקור s, האלגוריתם המדורג מחשב תחילה את משקל המסלול הקצר ביותר $\delta_1(s, v)$ עבור כל $v \in V$, לאחר מכן הוא מחשב את $\delta_2(s, v)$ עבור כל $v \in V$, וכן הלאה, עד שהוא מחשב את $\delta_k(s, v)$ עבור כל הקדקודים $v \in V$. במשך כל התהליך הזה, אנו מניחים כי $|E| \geq |V| - 1$, ונראה שחישוב δ_i מתוך δ_{i-1} מתבצע בזמן $O(E)$, כך שזמן הריצה של האלגוריתם כולו הוא $O(kE) = O(E \lg W)$.

א. נניח כי $|E| \leq \delta(s, v)$ עבור כל הקדקודים $v \in V$. הראה שניתן לחשב את $\delta(s, v)$ עבור כל הקדקודים $v \in V$ בזמן $O(E)$.

ב. הראה שניתן לחשב את $\delta_1(s, v)$ עבור כל הקדקודים $v \in V$ בזמן $O(E)$.

הבה נתמקד עתה בחישוב δ_i מתוך δ_{i-1}.

ג. הוכח שעבור $i = 2, 3, \ldots, k$, מתקיים אחד מהשניים: $w_i(u, v) = 2w_{i-1}(u, v)$ או $w_i(u, v) = 2w_{i-1}(u, v) + 1$. לאחר מכן, הוכח שעבור כל $v \in V$, מתקיים:

$$2\delta_{i-1}(s, v) \leq \delta_i(s, v) \leq 2\delta_{i-1}(s, v) + |V| - 1$$

ד. נגדיר עבור $i = 2, 3, \ldots, k$ ועבור כל הקשתות $(u, v) \in E$:

$$\hat{w}_i(u, v) = w_i(u, v) + 2\delta_{i-1}(s, u) - 2\delta_{i-1}(s, v)$$

הוכח שעבור $i = 2, 3, \ldots, k$ ועבור כל הקדקודים $u, v \in V$, ערכו של "המשקל המחודש" $\hat{w}_i(u, v)$ של הקשת (u, v) הוא שלם אי־שלילי.

ה. עתה, נגדיר את $\hat{\delta}_i(s, v)$ כמשקל המסלול הקצר ביותר מ־s אל v על פי פונקציית המשקל $\hat{w}_i$. הוכח שעבור $i = 2, 3, \ldots, k$ ועבור כל הקדקודים $v \in V$ מתקיים:

$$\hat{\delta}_i(s, v) = \hat{\delta}_i(s, v) + 2\delta_{i-1}(s, v)$$

וכן $\hat{\delta}_i(s, v) \leq |E|$.

ו. הראה כיצד לחשב את $\delta_i(s, v)$ מתוך $\delta_{i-1}(s, v)$ עבור כל הקדקודים $v \in V$ בזמן $O(E)$, והסק שניתן לחשב את $\delta(s, v)$ עבור כל הקדקודים $v \in V$ בזמן $O(E \lg W)$.

25-5 האלגוריתם של קארפ (Karp) לחישוב המשקל הממוצע המינימלי של מעגל

יהי $G = (V, E)$ גרף מכוון עם פונקציית משקל $w : E \to \mathbf{R}$, ויהי $n = |V|$. נגדיר **משקל ממוצע** (mean weight) של מעגל $c = \langle e_1, e_2, \ldots, e_k \rangle$ של קשתות ב־E על־ידי:

$$\mu(c) = \frac{1}{k} \sum_{i=1}^{k} w(e_i)$$

יהי $\mu^* = \min_c \mu(c)$, כאשר c נע על כל המעגלים המכוונים ב־G. מעגל c שעבורו $\mu(c) = \mu^*$ נקרא **מעגל בעל משקל ממוצע מינימלי** (minimum mean-weight cycle). בעיה זו חוקרת אלגוריתם יעיל לחישוב μ^*.

נניח, בלי הגבלת הכלליות, שניתן להגיע אל כל קדקוד $v \in V$ מקדקוד מקור $s \in V$. יהי $\delta(s, v)$ משקלו של מסלול קצר ביותר מ־s אל v, ויהי $\delta_k(s, v)$ משקלו של מסלול קצר ביותר מ־s אל v המכיל *בדיוק* k קשתות. אם לא קיים מסלול בן k קשתות בדיוק מ־s אל v, אזי $\delta_k(s, v) = \infty$.

א. הראה שאם $\mu^* = 0$, אזי G אינו מכיל מעגלים בעלי משקל שלילי, וכן $\delta(s, v) = \min_{0 \le k \le n-1} \delta_k(s, v)$ עבור כל הקדקודים $v \in V$.

ב. הראה שאם $\mu^* = 0$, אזי:

$$\max_{0 \le k \le n-1} \frac{\delta_n(s, v) - \delta_k(s, v)}{n - k} \ge 0$$

עבור כל הקדקודים $v \in V$. (**רמז:** השתמש בשתי התכונות שהוכחת בחלק א.)

ג. יהי c מעגל שמשקלו 0, ויהיו u ו־v שני קדקודים כלשהם על c. נניח שמשקלו של המסלול מ־u אל v לאורך המעגל הוא x. הוכח כי $\delta(s, v) = \delta(s, u) + x$. (**רמז:** משקל המסלול מ־v אל u לאורך המעגל הוא $-x$.)

ד. הראה שאם $\mu^* = 0$, אזי קיים קדקוד v על המעגל בעל המשקל הממוצע המינימלי כך שמתקיים:

$$\max_{0 \le k \le n-1} \frac{\delta_n(s, v) - \delta_k(s, v)}{n - k} = 0$$

(**רמז:** הראה שבהינתן מסלול קצר ביותר אל קדקוד כלשהו על המעגל בעל המשקל הממוצע המינימלי, ניתן להמשיך מסלול זה לאורך המעגל, ובכך ליצור מסלול קצר ביותר אל הקדקוד הבא על המעגל.)

ה. הראה שאם $\mu^* = 0$, אזי:

$$\min_{v \in V} \max_{0 \le k \le n-1} \frac{\delta_n(s, v) - \delta_k(s, v)}{n - k} = 0$$

ו. הראה שאם מוסיפים קבוע t למשקלה של כל קשת ב־G, אזי μ^* גדל ב־t. השתמש בכך כדי להראות כי:

$$\mu^* = \min_{v \in V} \max_{0 \le k \le n-1} \frac{\delta_n(s, v) - \delta_k(s, v)}{n - k}$$

ז. כתוב אלגוריתם שזמן ריצתו $O(VE)$ לחישוב μ^*.

הערות ביבליוגרפיות לפרק 25

האלגוריתם של דייקסטרה (Dijkstra) [55] הופיע ב־1959, אולם לא הוזכר בו כלל תור קדימויות. אלגוריתם בלמן־פורד (Bellman-Ford) מבוסס על אלגוריתמים נפרדים של Bellman [22] ושל Ford [71]. בלמן מתאר את הקשר בין מסלולים קצרים ביותר לבין אילוצי הפרשים. Lawler [132] מתאר את האלגוריתם הלינארי למציאת מסלולים קצרים ביותר בגמ"ל ומתייחס אליו כאל חלק מהפולקלור.

כאשר משקלות הקשתות הם מספרים שלמים קטנים יחסית, ניתן לפתור את בעיית המסלולים הקצרים ביותר ממקור יחיד באמצעות אלגוריתמים יעילים יותר. Ahuja, Mehlhorn, Orlin & Tarjan [6] מציגים אלגוריתם שזמן ריצתו על גרפים עם משקלות קשתות אי־שליליים

הוא $O(E + V\sqrt{\lg W})$, כאשר W הוא המשקל המקסימלי מבין משקלות קשתות הגרף. הם גם מציגים אלגוריתם קל לתכנות שזמן ריצתו $O(E + V \lg W)$. עבור גרפים עם קשתות בעלות משקלות שליליים, האלגוריתם של Gabow & Tarjan [77] רץ בזמן $O(\sqrt{V}E \lg(VW))$, כאשר W הוא הערך המוחלט של המשקל בעל הערך המוחלט המקסימלי מבין משקלות הקשתות בגרף.

ב־Papadimitriou & Steiglitz [154] ניתן למצוא דיון טוב בשיטת הסימפלקס ובאלגוריתם האליפסואיד, וכן באלגוריתמים אחרים הקשורים לתכנון לינארי. אלגוריתם הסימפלקס עבור תכנון לינארי הומצא על־ידי G. Danzig ב־1947. גרסאות של הסימפלקס היו ונותרו השיטות הפופולריות ביותר לפתרון בעיות תכנון לינארי. אלגוריתם האליפסואיד הוא של L. G. Khachian, 1979, ומבוסס על עבודה קודמת של N. Z. Shor, D. B. Judin ו־A. S. Nemirovskii. Karmarkar מתאר את האלגוריתם שלו ב־[115].

26 מסלולים קצרים ביותר בין כל הזוגות

בפרק זה נבחן את הבעיה של מציאת מסלולים קצרים ביותר בין כל זוגות הקדקודים בגרף. בעיה כזאת עשויה לעלות כאשר מכינים טבלת מרחקים בין כל זוגות הערים עבור מפת דרכים. כמו בפרק 25, נתון לנו גרף מכוון משוקלל $G = (V, E)$ עם פונקציית משקל $w : E \to \mathbf{R}$ הממפה קשתות אל משקלות בעלי ערכים ממשיים. ברצוננו למצוא, עבור כל זוג קדקודים $u, v \in V$, מסלול קצר ביותר (דהיינו, בעל משקל מינימלי) מ־u אל v, כאשר משקלו של מסלול מוגדר כסכום משקלות הקשתות המרכיבות אותו. בדרך כלל נרצה שהפלט יינתן בצורת טבלה: הוא בשורה של u ובעמודה של v יכיל את משקל המסלול הקצר ביותר מ־u אל v.

ניתן לפתור בעיית מסלולים קצרים ביותר בין כל הזוגות על ידי הרצת אלגוריתם עבור מסלולים קצרים ביותר ממקור יחיד $|V|$ פעמים, פעם אחת עבור כל אחד מן הקדקודים כמקור. אם כל משקלות הקשתות הם אי־שליליים, ניתן להשתמש באלגוריתם של דייקסטרה. אם מממשים את תור הקדימויות באמצעות מערך לינארי, זמן הריצה הוא $O(V^3 + VE) = O(V^3)$. מימוש תור הקדימויות באמצעות ערימה בינרית מניב זמן ריצה של $O(VE \lg V)$, המהווה שיפור אם הגרף דליל. לחלופין, ניתן לממש את תור הקדימויות באמצעות ערימת פיבונצ'י, ולקבל זמן ריצה של $O(V^2 \lg V + VE)$.

אם מרשים משקלות קשתות שליליים, לא ניתן להשתמש באלגוריתם של דייקסטרה. תחת זאת עלינו להריץ את אלגוריתם בלמן־פורד האיטי יותר, פעם אחת מכל אחד מן הקדקודים. זמן הריצה במקרה זה הוא $O(V^2E)$, שהוא $O(V^4)$ אם הגרף צפוף. בפרק זה נראה כיצד נוכל לעשות זאת ביתר יעילות. כמו כן, נחקור כיצד הבעיה של מסלולים קצרים ביותר בין כל הזוגות מתקשרת לכפל מטריצות, ונלמד את המבנה האלגברי שלה.

שלא כמו האלגוריתמים עבור מקור יחיד, אשר מניחים כי הגרף מיוצג על־ידי רשימות סמיכות, מרבית האלגוריתמים בפרק זה משתמשים בייצוג על־ידי מטריצת סמיכויות. (האלגוריתם של ג'ונסון עבור גרפים דלילים משתמש ברשימות סמיכות.) הקלט הוא מטריצה W שממדיה $n \times n$, המייצגת את משקלות הקשתות בגרף מכוון בן n קדקודים $G = (V, E)$. כלומר, $W = (w_{ij})$, כאשר:

$$w_{ij} = \begin{cases} 0 & \text{אם } i = j \\ \text{משקל הקשת המכוונת } (i,j) & \text{אם } i \neq j \text{ וגם } (i,j) \in E \\ \infty & \text{אם } i \neq j \text{ וגם } (i,j) \notin E \end{cases} \qquad (26.1)$$

מותר שיהיו משקלות שליליים לקשתות, אולם לעת עתה אנו מניחים שגרף הקלט אינו מכיל מעגלים בעלי משקל שלילי.

טבלת הפלט של האלגוריתמים למציאת מסלולים קצרים ביותר בין כל הזוגות המוצגים בפרק זה היא מטריצה $D = (d_{ij})$ בגודל $n \times n$, כאשר התא d_{ij} מכיל את משקלו של מסלול קצר ביותר

מהקדקוד i אל הקדקוד j. כלומר, אם נסמן ב־$\delta(i, j)$ את משקלו של מסלול קצר ביותר מקדקוד i אל קדקוד j (כמו בפרק 25), אזי עם עצירת האלגוריתם, $d_{ij} = \delta(i, j)$.

כדי לפתור את הבעיה של מסלולים קצרים ביותר בין כל הזוגות על מטריצת סמיכויות הנתונה כקלט, עלינו לחשב לא רק את משקליהם של מסלולים קצרים ביותר אלא גם **מטריצת קודמים** (predecessor matrix) $\Pi = (\pi_{ij})$ אשר בה $\pi_{ij} = \text{NIL}$ אם $i = j$ או אם לא קיים מסלול מ־i אל j; אחרת, π_{ij} הוא הקודם של j על איזשהו מסלול קצר ביותר מ־i. בדיוק כשם שתת־גרף הקודמים G_π מפרק 25 הוא עץ מסלולים קצרים ביותר עבור קדקוד מקור נתון, כך תת־גרף המושרה על־ידי השורה ה־i במטריצה Π צריך להיות עץ מסלולים קצרים ביותר ששורשו i. עבור כל קדקוד $i \in V$, נגדיר את **תת־גרף הקודמים** (predecessor subgraph) של G עבור i, $G_{\pi,i} = (V_{\pi,i}, E_{\pi,i})$, על־ידי:

$$V_{\pi,i} = \{j \in V : \pi_{ij} \neq \text{NIL}\} \cup \{i\}$$

וכן:

$$E_{\pi,i} = \{(\pi_{ij}, j) : j \in V_{\pi,i} \text{ וגם } \pi_{ij} \neq \text{NIL}\}$$

אם $G_{\pi,i}$ הוא עץ מסלולים קצרים ביותר, אזי השגרה הבאה, שהיא גרסה מותאמת של השגרה PRINT-PATH מפרק 23, מדפיסה מסלול קצר ביותר מקדקוד i אל קדקוד j.

```
PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, j)
1  if i = j
2     then print i
3     else if π_ij = NIL
4             then print "no path from" i "to" j "exists"
5             else PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, π_ij)
6                  print j
```

כדי להבליט את המאפיינים המהותיים של האלגוריתמים למציאת מסלולים קצרים ביותר בין כל הזוגות שיוצגו בפרק זה, לא נטפל בבנייתן של מטריצות הקודמים ובתכונותיהן באותה הרחבה שבה טיפלנו בתת־גרף הקודמים בפרק 25. חלק מהתרגילים עוסקים בתכונות הבסיסיות של מטריצות הקודמים.

תקציר הפרק

סעיף 26.1 מציג אלגוריתם תכנון־דינמי המבוסס על כפל מטריצות לפתרון בעיית המסלולים הקצרים ביותר בין כל הזוגות. על־ידי שימוש בטכניקה של "העלאה חוזרת בריבוע" ניתן לגרום לכך שהאלגוריתם ירוץ בזמן $\Theta(V^3 \lg V)$. אלגוריתם תכנון־דינמי אחר, אלגוריתם פלויד־וורשאל (Floyd-Warshall), מובא בסעיף 26.2. אלגוריתם פלויד־וורשאל רץ בזמן $\Theta(V^3)$. סעיף 26.2 עוסק גם בבעיה של מציאת חסָגור הטרנזיטיבי של גרף מכוון, הקשור לבעיית המסלולים הקצרים ביותר בין כל הזוגות. בסעיף 26.3 מוצג האלגוריתם של ג'ונסון (Johnson). שלא כמו האלגוריתמים האחרים המוצגים בפרק זה, האלגוריתם של ג'ונסון משתמש בייצוג גרף על־ידי רשימות סמיכות. הוא פותר את בעיית המסלולים הקצרים ביותר בין כל הזוגות בזמן $O(V^2 \lg V + VE)$, ולפיכך הוא אלגוריתם טוב עבור גרפים גדולים ודלילים. לבסוף, בסעיף 26.4, נבחן מבנה אלגברי המכונה "חוג־למחצה סגור", המאפשר ליישם אלגוריתמי מסלולים־קצרים־ביותר רבים עבור מגוון רחב של בעיות אחרות של כל הזוגות, שאינן קשורות במסלולים קצרים ביותר.

לפני שנמשיך, עלינו לקבוע מוסכמות מספר באשר לייצוגים של מטריצות סמיכויות. ראשית, בדרך כלל נניח שגרף הקלט $G = (V, E)$ מכיל n קדקודים, כך ש־$n = |V|$. שנית, נשתמש במוסכמה

של סימון מטריצות באותיות לטיניות גדולות, כגון W או D, וסימון איברי המטריצות באותיות לטיניות קטנות עם אינדקסים, כגון w_{ij} או d_{ij}. בחלק מן המטריצות יופיעו אינדקסים עיליים בסוגריים, כגון $D^{(m)} = (d_{ij}^{(m)})$, לציון איטרציות. לבסוף, בהינתן מטריצה A בגודל $n \times n$, נניח שהערך של n מאוחסן במאפיין $rows[A]$.

26.1 מסלולים קצרים ביותר וכפל מטריצות

בסעיף זה נציג אלגוריתם תכנון־דינמי עבור הבעיה של מסלולים קצרים ביותר בין כל הזוגות בגרף מכוון $G = (V, E)$. כל לולאה ראשית בתכנית הדינמית תקרא לפעולה הדומה מאוד לכפל מטריצות, כך שהאלגוריתם ייראה ככפל חוזר של מטריצות. נתחיל בפיתוח אלגוריתם הפותר את הבעיה של מסלולים קצרים ביותר בין כל הזוגות בזמן $\Theta(V^4)$, ולאחר מכן נשפר את זמן ריצתו של האלגוריתם ל־$\Theta(V^3 \lg V)$.

לפני שנמשיך, הבה נחזור בקצרה על השלבים בפיתוחו של אלגוריתם תכנון־דינמי שהובאו בפרק 16:

1. אפיון מבנהו של פתרון אופטימלי.
2. הגדרה רקורסיבית של ערכו של פתרון אופטימלי.
3. חישוב ערכו של פתרון אופטימלי "מלמטה־למעלה".

(השלב הרביעי, בניית פתרון אופטימלי מתוך המידע שחושב, מובא בתרגילים.)

המבנה של מסלול קצר ביותר

נתחיל באפיון מבנהו של פתרון אופטימלי. עבור הבעיה של מסלולים קצרים ביותר בין כל הזוגות בגרף $G = (V, E)$, הוכחנו (למה 25.1) שכל התת־מסלולים של מסלול קצר ביותר הם מסלולים קצרים ביותר. נניח כי הגרף מיוצג על־ידי מטריצת סמיכויות $W = (w_{ij})$. נתבונן במסלול קצר ביותר p מקדקוד i אל קדקוד j, ונניח כי p מכיל לכל היותר m קשתות. בהנחה שלא קיימים מעגלים בעלי משקל שלילי, m סופי אם $i = j$, אזי p אינו מכיל קשתות כלל ומשקלו 0. אם הקדקודים i ו־j שונים זה מזה, אזי נפרק את המסלול p באופן הבא: $i \overset{p'}{\rightsquigarrow} k \rightarrow j$, כאשר המסלול p' מכיל עתה לכל היותר $m - 1$ קשתות. יתר על כן, על פי למה 25.1, p' הוא מסלול קצר ביותר מ־i אל k. לכן, על פי מסקנה 25.2, מקבלים כי $\delta(i, j) = \delta(i, k) + w_{kj}$.

פתרון רקורסיבי לבעיה של מסלולים קצרים ביותר בין כל הזוגות

כעת, יהי $d_{ij}^{(m)}$ המשקל המינימלי של מסלול כלשהו מקדקוד i אל קדקוד j המכיל לכל היותר m קשתות. כאשר $m = 0$, קיים מסלול קצר ביותר מ־i אל j שאינו מכיל קשתות כלל אם ורק אם $i = j$. לכן:

$$d_{ij}^{(0)} = \begin{cases} 0 & \text{אם } i = j \\ \infty & \text{אם } i \neq j \end{cases}$$

עבור $m \geq 1$, אנו מחשבים את $d_{ij}^{(m)}$ כמינימום מבין $d_{ij}^{(m-1)}$ (משקלו של המסלול הקצר ביותר מ־i אל j המכיל לכל היותר $m - 1$ קשתות) והמשקל המינימלי של מסלול כלשהו מ־i אל j המכיל לכל היותר m קשתות המתקבל על־ידי כך שמתבוננים בכל הקודמים האפשריים k של j. נגדיר אפוא באופן רקורסיבי:

$$d_{ij}^{(m)} = \min\left(d_{ij}^{(m-1)}, \min_{1 \leq k \leq n}\left\{d_{ik}^{(m-1)} + w_{kj}\right\}\right)$$

$$= \min_{1 \leq k \leq n}\left\{d_{ik}^{(m-1)} + w_{kj}\right\} \qquad (26.2)$$

השוויון האחרון נובע מכך ש־$w_{jj} = 0$ עבור כל j.

מהם המשקלים האמיתיים, $\delta(i, j)$, של המסלולים הקצרים ביותר? אם הגרף אינו מכיל מעגלים בעלי משקל שלילי, אזי כל המסלולים הקצרים ביותר הם פשוטים ולכן מכילים לכל היותר $n - 1$ קשתות. משקלו של מסלול מקדקוד i אל קדקוד j בעל יותר מ־$n - 1$ קשתות אינו יכול להיות נמוך ממשקלו של מסלול קצר ביותר מ־i אל j. המשקלים האמיתיים של המסלולים הקצרים ביותר נתונים אפוא על־ידי:

$$\delta(i, j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \cdots \qquad (26.3)$$

חישוב משקלי המסלולים הקצרים ביותר "מלמטה־למעלה"

בהינתן המטריצה $W = (w_{ij})$ כקלט, אנו מחשבים עתה סדרת מטריצות $D^{(1)}, D^{(2)}, \ldots, D^{(n-1)}$, אשר מקיימות $D^{(m)} = (d_{ij}^{(m)})$ עבור $m = 1, 2, \ldots, n - 1$. המטריצה הסופית $D^{(n-1)}$ מכילה את המשקלים האמיתיים של המסלולים הקצרים ביותר. יש לשים לב כי מאחר ש־$d_{ij}^{(1)} = w_{ij}$ עבור כל הקדקודים $i, j \in V$, נובע מכך כי $D^{(1)} = W$.

לב האלגוריתם הוא השגרה הבאה, אשר, בהינתן מטריצות $D^{(m-1)}$ ו־W, מחזירה את המטריצה $D^{(m)}$. כלומר, השגרה מאריכה בקשת אחת נוספת את המסלולים הקצרים ביותר שחושבו עד כה.

```
EXTEND-SHORTEST-PATHS(D, W)
1  n ← rows[D]
2  let D' = (d'_ij) be an n × n matrix
3  for i ← 1 to n
4      do for j ← 1 to n
5             do d'_ij ← ∞
6                for k ← 1 to n
7                    do d'_ij ← min(d'_ij, d_ik + w_kj)
8  return D'
```

השגרה מחשבת מטריצה $D' = (d'_{ij})$ המוחזרת בסיום ההרצה. השגרה עושה זאת על־ידי כך שהיא מחשבת את משוואה (26.2) עבור כל i וכל j, כאשר D משמשת כ־$D^{(m-1)}$ ו־D' משמשת כ־$D^{(m)}$. (סימוני המטריצות נכתבים ללא האינדקסים העיליים כדי שמטריצות הקלט והפלט של השגרה לא יהיו תלויות ב־m.) זמן הריצה של השגרה הוא $\Theta(n^3)$, בשל שלוש לולאות ה־**for** המקוננות.

עתה אנו יכולים לראות את הקשר לכפל מטריצות. נניח שברצוננו לחשב את מטריצת המכפלה $C = A \cdot B$ של שתי מטריצות A ו־B בגודל $n \times n$. אזי, עבור $i, j = 1, 2, \ldots, n$, אנו מחשבים:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj} \qquad (26.4)$$

נשים לב לכך שאם נציב במשוואה (26.2) את ההצבות הבאות:

$$\begin{aligned} d^{(m-1)} &\to a \\ w &\to b \\ d^{(m)} &\to c \\ \min &\to + \\ + &\to \cdot \end{aligned}$$

נקבל את משוואה (26.4). לפיכך, אם נבצע את השינויים האלה בשגרה EXTEND-SHORTEST-PATHS, וכמו כן נחליף את ∞ (איבר היחידה עבור min) ב-0 (איבר היחידה עבור +), נקבל את השגרה הפשוטה לכפל מטריצות, שזמן ריצתה $\Theta(n^3)$.

```
MATRIX-MULTIPLY(A, B)
1  n ← rows[A]
2  let C be an n × n matrix
3  for i ← 1 to n
4      do for j ← 1 to n
5             do c_ij ← 0
6                for k ← 1 to n
7                    do c_ij ← c_ij + a_ik · b_kj
8  return C
```

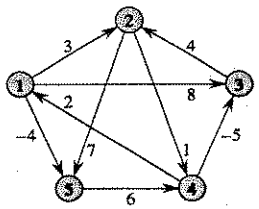
נחזור עתה לבעיית המסלולים הקצרים ביותר בין כל הזוגות. אנו מחשבים את משקליהם של המסלולים הקצרים ביותר על-ידי הארכת מסלולים קצרים ביותר קשת אחר קשת. נסמן ב-$A \cdot B$ את מטריצת "המכפלה" המוחזרת על-ידי EXTEND-SHORTEST-PATHS(A, B). אנו מחשבים את סדרת $n-1$ המטריצות הבאה:

$$\begin{aligned} D^{(1)} &= D^{(0)} \cdot W &&= W \\ D^{(2)} &= D^{(1)} \cdot W &&= W^2 \\ D^{(3)} &= D^{(2)} \cdot W &&= W^3 \\ &\vdots \\ D^{(n-1)} &= D^{(n-2)} \cdot W &&= W^{n-1} \end{aligned}$$

כפי שטענו לעיל, המטריצה $D^{(n-1)} = W^{n-1}$ מכילה את משקליהם של המסלולים הקצרים ביותר. השגרה שלהלן מחשבת את סדרת המטריצות הזאת בזמן $\Theta(n^4)$.

```
SLOW-ALL-PAIRS-SHORTEST-PATHS(W)
1  n ← rows[W]
2  D^(1) ← W
3  for m ← 2 to n − 1
4      do D^(m) ← EXTEND-SHORTEST-PATHS(D^(m−1), W)
5  return D^(n−1)
```

באיור 26.1 מופיעים גרף מכוון וסדרת המטריצות $D^{(m)}$ המחושבות על-ידי השגרה SLOW-ALL-PAIRS-SHORTEST-PATHS.



$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad D^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

איור 26.1 גרף מכוון וסדרת המטריצות $D^{(m)}$ המחושבת על-ידי SLOW-ALL-PAIRS-SHORTEST-PATHS. הקורא יכול לוודא כי $D^{(5)} = D^{(4)} \cdot W$, השווה ל-$D^{(4)}$, ולכן $D^{(m)} = D^{(4)}$ עבור כל $m \geq 4$.

שיפור זמן הריצה

אולם מטרתנו אינה לחשב את כל המטריצות $D^{(m)}$; אנו מעוניינים רק במטריצה $D^{(n-1)}$. כזכור, בהיעדרם של מעגלים בעלי משקל שלילי, משוואה (26.3) גוררת $D^{(m)} = D^{(n-1)}$ עבור כל השלמים $m \geq n-1$. ניתן לחשב את $D^{(n-1)}$ באמצעות $\lceil \lg(n-1) \rceil$ מכפלות מטריצות בלבד על-ידי חישוב הסדרה:

$$\begin{aligned} D^{(1)} &= W \\ D^{(2)} &= W^2 &&= W \cdot W \\ D^{(4)} &= W^4 &&= W^2 \cdot W^2 \\ D^{(8)} &= W^8 &&= W^4 \cdot W^4 \\ &\vdots \\ D^{(2^{\lceil \lg(n-1) \rceil})} &= W^{2^{\lceil \lg(n-1) \rceil}} &&= W^{2^{\lceil \lg(n-1) \rceil - 1}} \cdot W^{2^{\lceil \lg(n-1) \rceil - 1}} \end{aligned}$$

מאחר ש-$2^{\lceil \lg(n-1) \rceil} \geq n-1$, מטריצת המכפלה הסופית $D^{(2^{\lceil \lg(n-1) \rceil})}$ שווה ל-$D^{(n-1)}$.

השגרה הבאה מחשבת את סדרת המטריצות שלעיל באמצעות טכניקה זו של **העלאה חוזרת בריבוע**.

```
FASTER-ALL-PAIRS-SHORTEST-PATHS(W)
1  n ← rows[W]
2  D^(1) ← W
3  m ← 1
4  while n − 1 > m
5      do D^(2m) ← EXTEND-SHORTEST-PATHS(D^(m), D^(m))
6         m ← 2m
7  return D^(m)
```

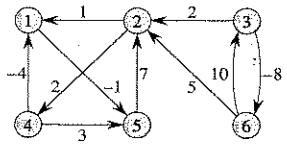
בכל איטרציה של לולאת ה-**while** שבשורות 4-6, אנו מחשבים את $D^{(2m)} = (D^{(m)})^2$, כאשר מתחילים עם $m = 1$. בסופה של כל איטרציה, אנו מכפילים את ערכו של m. האיטרציה האחרונה מחשבת את $D^{(n-1)}$ על-ידי כך שהיא מחשבת ממש את $D^{(2m)}$ עבור איזשהו $n - 1 \le 2m < 2n - 2$. על פי משוואה (26.3), $D^{(2m)} = D^{(n-1)}$. בפעם הבאה שמתבצעת הבדיקה בשורה 4, ערכו של m הוכפל, כך שעתה $n - 1 \le m$, הבדיקה נכשלת, והשגרה מחזירה את המטריצה האחרונה שחושבה.

זמן הריצה של FASTER-ALL-PAIRS-SHORTEST-PATHS הוא $\Theta(n^3 \lg n)$, שכן חישוב כל אחת מ-$\lceil \lg(n-1) \rceil$ מכפלות המטריצות מתבצע בזמן $\Theta(n^3)$. יש לשים לב לכך שהקוד הדוק ואינו מכיל מבני נתונים מתוחכמים, ולכן הקבוע החבוי בסימון Θ הוא קטן.

תרגילים

26.1-1

הרץ את SLOW-ALL-PAIRS-SHORTEST-PATHS על הגרף המכוון המשוקלל שבאיור 26.2. הצג את המטריצות המתקבלות בכל איטרציה של כל אחת מן הלולאות. אחר-כך הרץ את FASTER-ALL-PAIRS-SHORTEST-PATHS באותו אופן.



איור 26.2 גרף מכוון משוקלל עבור תרגילים 26.1-1, 26.2-1 ו-26.3-1.

26.1-2

מדוע אנו דורשים כי $w_{ii} = 0$ עבור כל $1 \le i \le n$?

26.1-3

לְמה בכפל מטריצות רגיל מתאימה המטריצה:

$$D^{(0)} = \begin{pmatrix} 0 & \infty & \infty & \cdots & \infty \\ \infty & 0 & \infty & \cdots & \infty \\ \infty & \infty & 0 & \cdots & \infty \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \infty & \infty & \infty & \cdots & 0 \end{pmatrix}$$

המשמשת באלגוריתמים לחישוב מסלולים קצרים ביותר?

26.1-4

הראה כיצד ניתן לבטא את בעיית המסלולים הקצרים ביותר ממקור יחיד ככפל של מטריצות ווקטור. תאר כיצד חישוב ערכה של מכפלה זו מקביל לאלגוריתם דמוי בלמן-פורד (ראה סעיף 25.3).

26.1-5

נניח שברצוננו גם לזהות את הקדקודים על המסלולים הקצרים ביותר המתקבלים מן האלגוריתמים המוצגים בסעיף זה. הראה כיצד ניתן לחשב בזמן $O(n^3)$ את מטריצת הקודמים Π מתוך המטריצה הסופית D של משקלי מסלולים קצרים ביותר.

26.1-6

את הקדקודים על מסלולים קצרים ביותר ניתן לחשב גם בעת חישוב משקליהם של מסלולים אלה. נסמן ב-$\pi_{ij}^{(m)}$ את הקודם לקדקוד j על מסלול כלשהו בעל משקל מינימלי מ-i אל j המכיל לכל היותר m קשתות. שנה את השגרה EXTEND-SHORTEST-PATHS ואת השגרה SLOW-ALL-PAIRS-SHORTEST-PATHS כך שיחשבו את המטריצות $\Pi^{(1)}, \Pi^{(2)}, \ldots, \Pi^{(n-1)}$ במהלך חישובן של המטריצות $D^{(1)}, D^{(2)}, \ldots, D^{(n-1)}$.

26.1-7

השגרה FASTER-ALL-PAIRS-SHORTEST-PATHS, כפי שהיא כתובה כאן, מחייבת אותנו לאחסן $\lceil \lg(n-1) \rceil$ מטריצות, אשר כל אחת מהן מכילה n^2 איברים, כך שמקום האחסון הכולל הנדרש הוא $\Theta(n^2 \lg n)$. שנה את השגרה כך שתשתמש רק בשתי מטריצות בגודל $n \times n$, ולפיכך תדרוש רק מקום אחסון של $\Theta(n^2)$.

26.1-8

שנה את FASTER-ALL-PAIRS-SHORTEST-PATHS כך שתגלה את קיומו של מעגל בעל משקל שלילי.

26.1-9

כתוב אלגוריתם יעיל למציאת האורך (מספר הקשתות) של מעגל בעל משקל שלילי ואורך מינימלי בגרף.

26.2 אלגוריתם פלויד-וורשאל

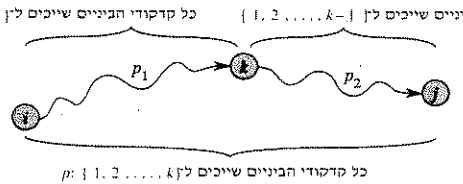
בסעיף זה נשתמש בצורה שונה של תכנון דינמי כדי לפתור את בעיית המסלולים הקצרים ביותר בין כל הזוגות בגרף מכוון $G = (V, E)$. האלגוריתם המתקבל ידוע כ**אלגוריתם פלויד-וורשאל** (Floyd-Warshall algorithm) וזמן ריצתו $\Theta(V^3)$. כמו קודם, הגרף עשוי להכיל קשתות בעלות משקל שלילי אולם הוא אינו מכיל מעגלים בעלי משקל שלילי. כפי שעשינו בסעיף 26.1, נפתח אלגוריתם זה על פי שלבי התכנון הדינמי. לאחר שנחקור את האלגוריתם המתקבל, נציג שיטה דומה למציאת הסְגוֹר הטרנזיטיבי של גרף מכוון.

מבנהו של מסלול קצר ביותר

באלגוריתם פלויד־וורשאל נאפיין את מבנהו של מסלול קצר ביותר בצורה שונה מזו ששימשה אותנו באלגוריתמים המבוססים על כפל מטריצות. האלגוריתם בוחן את קדקודי ה"ביניים" על מסלול קצר ביותר, כאשר קדקוד **ביניים** במסלול פשוט $p = \langle v_1, v_2, \ldots, v_l \rangle$ הוא כל קדקוד ב־p שאינו v_1 או v_l, כלומר, כל קדקוד השייך לקבוצה $\{v_2, v_3, \ldots, v_{l-1}\}$.

אלגוריתם פלויד־וורשאל מבוסס על ההבחנה הבאה: תהי $V = \{1, 2, \ldots, n\}$ קבוצת הקדקודים של G, ונתבונן בתת־קבוצה $\{1, 2, \ldots, k\}$ של קדקודים, עבור k כלשהו. עבור כל זוג קדקודים $i, j \in V$, נתבונן בכל המסלולים מ־i אל j אשר קדקודי הביניים שלהם שייכים ל־$\{1, 2, \ldots, k\}$, ויהי p מסלול בעל משקל מינימלי מבין המסלולים האלה. (המסלול p הוא פשוט, שכן אנו מניחים כי G אינו מכיל מעגלים בעלי משקל שלילי.) אלגוריתם פלויד־וורשאל מנצל קשר המתקיים בין המסלול p לבין מסלולים קצרים ביותר מ־i אל j אשר כל קדקודי הביניים בהם שייכים לקבוצה $\{1, 2, \ldots, k-1\}$. הקשר תלוי בשאלה אם k הינו או אינו קדקוד ביניים במסלול p.

- אם k אינו קדקוד ביניים במסלול p, אזי כל קדקודי הביניים במסלול p שייכים לקבוצה $\{1, 2, \ldots, k-1\}$. לכן, מסלול קצר ביותר מקדקוד i אל קדקוד j אשר כל קדקודי הביניים בו שייכים לקבוצה $\{1, 2, \ldots, k-1\}$ הינו גם מסלול קצר ביותר מ־i אל j אשר כל קדקודי הביניים בו שייכים לקבוצה $\{1, 2, \ldots, k\}$.
- אם k הינו קדקוד ביניים במסלול p, אזי אנו מפרקים את p באופן הבא: $i \overset{p_1}{\rightsquigarrow} k \overset{p_2}{\rightsquigarrow} j$ כמתואר באיור 26.3. על פי למה 25.1, p_1 הוא מסלול קצר ביותר מ־i אל k אשר כל קדקודי הביניים בו שייכים לקבוצה $\{1, 2, \ldots, k\}$. למעשה, הקדקוד k אינו קדקוד ביניים במסלול p_1, ולכן, p_1 הינו מסלול קצר ביותר מ־i אל k אשר כל קדקודי הביניים בו שייכים לקבוצה $\{1, 2, \ldots, k-1\}$. באותו אופן, p_2 הוא מסלול קצר ביותר מקדקוד k אל קדקוד j אשר כל קדקודי הביניים בו שייכים לקבוצה $\{1, 2, \ldots, k-1\}$.



איור 26.3 p הוא מסלול קצר ביותר מקדקוד i אל קדקוד j, ו־k הוא קדקוד הביניים בעל המספר הגבוה ביותר ב־p. במסלול p_1, החלק של המסלול p מן הקדקוד i אל הקדקוד k, כל קדקודי הביניים שייכים לקבוצה $\{1, 2, \ldots, k-1\}$. הוא הדבר גם לגבי המסלול p_2 מן הקדקוד k אל הקדקוד j.

פתרון רקורסיבי לבעיית המסלולים הקצרים ביותר בין כל הזוגות

בהתבסס על מה שראינו לעיל, נביא עתה הגדרה רקורסיבית של אומדני מסלולים קצרים ביותר השונה מזו שהובאה בסעיף 26.1. יהי $d_{ij}^{(k)}$ משקלו של מסלול קצר ביותר מקדקוד i אל קדקוד j אשר כל קדקודי הביניים בו שייכים לקבוצה $\{1, 2, \ldots, k\}$. כאשר $k = 0$, מסלול מקדקוד i אל קדקוד j אשר אינו מכיל קדקוד ביניים שמספרו גבוה מ־0 אינו מכיל כלל קדקודי ביניים. הוא מורכב אפוא מקשת אחת לכל היותר, ומכאן, $d_{ij}^{(0)} = w_{ij}$. הגדרה רקורסיבית של $d_{ij}^{(k)}$ נתונה על־ידי:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{אם } k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{אם } k \geq 1 \end{cases} \tag{26.5}$$

המטריצה $D^{(n)} = (d_{ij}^{(n)})$ מכילה את התשובה הסופית – $d_{ij}^{(n)} = \delta(i, j)$ עבור כל $i, j \in V$ – משום שכל קדקודי הביניים שייכים לקבוצה $\{1, 2, \ldots, n\}$.

חישוב משקלי מסלולים קצרים ביותר "מלמטה־למעלה"

בהתבסס על נוסחת הנסיגה (26.5), ניתן להשתמש בשגרה שלהלן, הפועלת "מלמטה־למעלה", לחישוב הערכים $d_{ij}^{(k)}$ בסדר עולה של ערכי k. הקלט של השגרה הוא מטריצה W בגודל $n \times n$ המוגדרת כמו במשוואה (26.1). השגרה מחזירה את המטריצה $D^{(n)}$ של משקלי מסלולים קצרים ביותר.

```
FLOYD-WARSHALL(W)
1  n ← rows[W]
2  D^(0) ← W
3  for k ← 1 to n
4      do for i ← 1 to n
5             do for j ← 1 to n
6                    do d_ij^(k) ← min(d_ij^(k−1), d_ik^(k−1) + d_kj^(k−1))
7  return D^(n)
```

באיור 26.4 מופיעות המטריצות $D^{(k)}$ שחושבו על ידי אלגוריתם פלויד־וורשאל עבור הגרף מאיור 26.1.

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

איור 26.4 סדרות המטריצות $D^{(k)}$ ו־$\Pi^{(k)}$ שחושבו על־ידי אלגוריתם פלויד־וורשאל עבור הגרף מאיור 26.1.

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

איור 26.4 (המשך)

זמן הריצה של אלגוריתם פלויד־וורשאל נקבע על־ידי שלוש לולאות ה־**for** המקוננות שבשורות 3-6. כל ביצוע של שורה 6 אורך זמן $O(1)$. האלגוריתם רץ אפוא בזמן $\Theta(n^3)$. כמו באלגוריתם הסופי שבסעיף 26.1, הקוד הדוק ואין כאן מבני נתונים מורכבים, ולכן הקבוע החבוי בסימון Θ קטן. לפיכך, אלגוריתם פלויד־וורשאל הוא אלגוריתם מעשי למדי, אפילו עבור גרפי קלט בגודל בינוני.

בניית מסלול קצר ביותר

קיים מגוון של שיטות שונות לבניית מסלולים קצרים ביותר באלגוריתם פלויד־וורשאל. אחת הדרכים היא לחשב את המטריצה D של משקלי מסלולים קצרים ביותר ואז לבנות את מטריצת הקודמים Π מן המטריצה D. שיטה זו ניתן לממש כך שזמן הריצה יהיה $O(n^3)$ (ראה תרגיל 26.1-5). בהינתן מטריצת הקודמים Π, ניתן להשתמש ב־PRINT-ALL-PAIRS-SHORTEST-PATH כדי להדפיס את כל הקדקודים על מסלול קצר ביותר נתון.

ניתן לחשב את מטריצת הקודמים Π באופן "מקוון" (on-line) מיד כאשר אלגוריתם פלויד־וורשאל מחשב את המטריצות $D^{(k)}$. ובפירוט, אנו מחשבים סדרת מטריצות $\Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(n)}$, כאשר $\Pi = \Pi^{(n)}$ ו־$\pi_{ij}^{(k)}$ מוגדר כקדקוד הקודם לקדקוד j על מסלול קצר ביותר מקדקוד i אשר כל קדקודי הביניים בו שייכים לקבוצה $\{1, 2, \ldots, k\}$.

ניתן להגדיר את $\pi_{ij}^{(k)}$ באופן רקורסיבי. כאשר $k = 0$, מסלול קצר ביותר מ־i אל j אינו מכיל כלל קדקודי ביניים. לכן:

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{אם } i = j \text{ או } w_{ij} = \infty \\ i & \text{אם } i \neq j \text{ וגם } w_{ij} < \infty \end{cases} \tag{26.6}$$

עבור $k \geq 1$, אם נעבור במסלול $i \rightsquigarrow k \rightsquigarrow j$, אזי הקודם של j שנבחר הוא אותו קדקוד קודם של j שבחרנו על מסלול קצר ביותר מ־k אשר כל קדקודי הביניים בו שייכים לקבוצה $\{1, 2, \ldots, k-1\}$. אחרת, נבחר אותו קדקוד קודם של j שבחרנו על מסלול קצר ביותר מ־i אשר כל קדקודי הביניים בו שייכים לקבוצה $\{1, 2, \ldots, k-1\}$. ובניסוח פורמלי, עבור $k \geq 1$:

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{אם } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{אם } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases} \tag{26.7}$$

את הכללת החישוב של המטריצות $\Pi^{(k)}$ בשגרה FLOYD-WARSHALL אנו משאירים לקורא כתרגיל 26.2-3. באיור 26.4 מופיעה סדרת המטריצות $\Pi^{(k)}$ שמחשב האלגוריתם המתקבל עבור הגרף מאיור 26.1. התרגיל כולל גם משימה קשה יותר: ההוכחה כי תת־גרף הקודמים $G_{\pi,i}$ הוא עץ מסלולים קצרים ביותר ששורשו i. דרך שונה נוספת לבניית מסלולים קצרים ביותר נתונה כתרגיל 26.2-6.

סְגוֹר טרנזיטיבי של גרף מכוון

בהינתן גרף מכוון $G = (V, E)$ שקבוצת קדקודיו היא $V = \{1, 2, \ldots, n\}$, ייתכן שנרצה לדעת אם קיים ב־G מסלול מ־i אל j עבור כל זוגות הקדקודים $i, j \in V$. **הסְגוֹר הטרנזיטיבי** (transitive closure) של G מוגדר כגרף $G^* = (V, E^*)$, כאשר:

$$E^* = \{(i, j) : \text{קיים ב־} G \text{ מסלול מן הקדקוד } i \text{ אל הקדקוד } j\}$$

אחת הדרכים לחישוב הסגור הטרנזיטיבי של גרף בזמן $\Theta(n^3)$ היא לייחס לכל קשת ב־E משקל של 1, ולהריץ את אלגוריתם פלויד־וורשאל. אם קיים מסלול מקדקוד i אל קדקוד j, נקבל $d_{ij} < n$. אחרת, נקבל $d_{ij} = \infty$.

ישנה דרך אחרת, דומה, לחישוב הסגור הטרנזיטיבי של G בזמן $\Theta(n^3)$, אשר ביישומים מעשיים יכולה לחסוך זמן ומקום. דרך זו כרוכה בהצבת הפעולות הלוגיות $\vee$ (OR לוגי) ו־$\wedge$ (AND לוגי) במקום הפעולות האריתמטיות min ו־+ באלגוריתם פלויד־וורשאל. עבור $i, j, k = 1, 2, \ldots, n$, נגדיר משתנה $t_{ij}^{(k)}$ המקבל את הערך 1 אם קיים בגרף G מסלול מקדקוד i אל קדקוד j אשר כל קדקודי הביניים בו שייכים לקבוצה $\{1, 2, \ldots, k\}$, ואת הערך 0, אחרת. אנו בונים את הסגור הטרנזיטיבי $G^* = (V, E^*)$ על ידי כך שאנו מוסיפים קשת (i, j) ל־E^* אם ורק אם $t_{ij}^{(n)} = 1$. הגדרה רקורסיבית של $t_{ij}^{(k)}$, אנלוגית לנוסחת הנסיגה (26.5), היא:

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{אם } i \neq j \text{ וגם } (i, j) \notin E \\ 1 & \text{אם } i = j \text{ או } (i, j) \in E \end{cases}$$

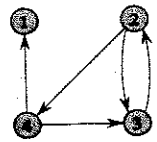
ועבור $k \geq 1$:

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right) \tag{26.8}$$

כמו באלגוריתם פלויד־וורשאל, אנו מחשבים את המטריצות $T^{(k)} = (t_{ij}^{(k)})$ בסדר עולה של k.

```
TRANSITIVE-CLOSURE(G)
1  n ← |V[G]|
2  for i ← 1 to n
3      do for j ← 1 to n
4             do if i = j or (i, j) ∈ E[G]
5                   then t_ij^(0) ← 1
6                   else t_ij^(0) ← 0
7  for k ← 1 to n
8      do for i ← 1 to n
9             do for j ← 1 to n
10                do t_ij^(k) ← t_ij^(k-1) ∨ (t_ik^(k-1) ∧ t_kj^(k-1))
11 return T^(n)
```

איור 26.5 מראה את המטריצות $T^{(k)}$ שחושבו על־ידי השגרה TRANSITIVE-CLOSURE עבור גרף לדוגמה. כמו אלגוריתם פלויד־וורשאל, זמן הריצה של השגרה TRANSITIVE-CLOSURE הוא $\Theta(n^3)$. אלא שישנם מחשבים שבהם פעולות לוגיות על סיביות בודדות מתבצעות מהר יותר מאשר פעולות אריתמטיות על מילות נתונים שהן מספרים שלמים. יתר על כן, מאחר שהאלגוריתם הישיר לחישוב הסגור הטרנזיטיבי משתמש רק בערכים בוליאניים ולא בערכים שלמים, דרישות המקום שלו קטנות מאלה של אלגוריתם פלויד־וורשאל פי גורם הקשור ישירות לגודלה של מילה בזיכרון המחשב.



$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

איור 26.5 גרף מכוון והמטריצות $T^{(k)}$ שחושבו על ידי אלגוריתם הסגור הטרנזיטיבי.

בסעיף 26.4 נראה שההקבלה בין FLOYD-WARSHALL לבין TRANSITIVE-CLOSURE אינה צירוף מקרים בלבד. שני האלגוריתמים מבוססים על טיפוס מסוים של מבנה אלגברי המכונה "חוג־למחצה סגור" (closed semiring).

תרגילים

26.2-1

הרץ את אלגוריתם פלויד־וורשאל על הגרף המכוון המשוקלל מאיור 26.2. הצג את המטריצה $D^{(k)}$ המתקבלת בכל איטרציה של הלולאה החיצונית.

26.2-2

כפי שראינו לעיל, אלגוריתם פלויד־וורשאל דורש מקום של $\Theta(n^3)$, שכן הוא מחשב את $d_{ij}^{(k)}$ עבור $i, j, k = 1, 2, \ldots, n$. הראה שהשגרה שלהלן, אשר בה פשוט הושמטו כל האינדקסים העיליים, נכונה, ולכן דרוש מקום של $\Theta(n^2)$ בלבד.


```
FLOYD-WARSHALL'(W)
1  n ← rows[W]
2  D ← W
3  for k ← 1 to n
4      do for i ← 1 to n
5          do for j ← 1 to n
6              d_ij ← min(d_ij, d_ik + d_kj)
7  return D
```

26.2-3

שנה את השגרה FLOYD-WARSHALL כך שתכלול את חישוב המטריצות $\Pi^{(k)}$ על פי משוואות (26.6) ו־(26.7). הוכח באופן מדוקדק שעבור כל $i \in V$, תת־גרף הקודמים $G_{\pi,i}$ הוא עץ מסלולים קצרים ביותר ששורשו i. (**רמז:** כדי להראות כי $G_{\pi,i}$ אינו מכיל מעגלים, הראה תחילה כי $\pi_{ij}^{(k)} = l$ גורר $d_{ij}^{(k)} \geq d_{il}^{(k-1)} + w_{lj}$. לאחר מכן, התאם את הוכחת למה 25.8.)

26.2-4

נניח שאנו משנים את דרך הטיפול בשוויון במשוואה (26.7) באופן הבא:

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{אם } d_{ij}^{(k-1)} < d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{אם } d_{ij}^{(k-1)} \geq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

האם הגדרה חלופית זו של מטריצת הקודמים Π נכונה?

26.2-5

כיצד ניתן להשתמש בפלט של אלגוריתם פלויד־וורשאל לזיהוי קיומו של מעגל בעל משקל שלילי?

26.2-6

דרך אחרת לשחזור מסלולים קצרים ביותר באלגוריתם פלויד־וורשאל היא להשתמש בערכים $\phi_{ij}^{(k)}$, עבור $i, j, k = 1, 2, \ldots, n$, כאשר $\phi_{ij}^{(k)}$ הוא קדקוד הביניים בעל המספר הסידורי הגבוה ביותר על מסלול קצר ביותר מ־i אל j. כתוב הגדרה רקורסיבית של $\phi_{ij}^{(k)}$, שנה את השגרה FLOYD-WARSHALL כך שתחשב את הערכים $\phi_{ij}^{(k)}$, וכתוב מחדש את השגרה PRINT-ALL-PAIRS-SHORTEST-PATH כך שתקבל כקלט את המטריצה $\Phi = (\phi_{ij}^{(n)})$. במה דומה המטריצה Φ לטבלה s מבעיית הכפל של סדרת מטריצות שהובאה בסעיף 16.1?

26.2-7

כתוב אלגוריתם המחשב בזמן $O(VE)$ את הסגור הטרנזיטיבי של גרף מכוון $G = (V, E)$.

26.2-8

נניח שניתן לחשב את הסגור הטרנזיטיבי של גרף מכוון ללא מעגלים בזמן $f(|V|, |E|)$, כאשר f היא פונקציה מונוטונית עולה של $|V|$ ו־$|E|$. הראה שהזמן לחישוב הסגור הטרנזיטיבי של גרף מכוון כללי $G = (V, E)$ הוא $f(|V| + |E|) + O(V + E^*)$, כאשר $G^* = (V, E^*)$.

26.3 האלגוריתם של ג'ונסון עבור גרפים דלילים

האלגוריתם של ג'ונסון (Johnson) מוצא מסלולים קצרים ביותר בין כל הזוגות בזמן $O(V^2 \lg V + VE)$; עבור גרפים דלילים זהו אפוא אלגוריתם מהיר אסימפטוטית הן מהעלאה חוזרת בריבוע של מטריצות והן מאלגוריתם פלויד־וורשאל. האלגוריתם מחזיר מטריצה של משקלי מסלולים קצרים ביותר בין כל זוגות הקדקודים או מדווח שגרף הקלט מכיל מעגל בעל משקל שלילי. האלגוריתם של ג'ונסון משתמש כתת־שגרות גם באלגוריתם של דייקסטרה וגם באלגוריתם בלמן־פורד, המתוארים בפרק 25.

האלגוריתם של ג'ונסון משתמש בטכניקה של **שקלול־מחדש** (reweighting), הפועלת באופן הבא: אם כל משקלות הקשתות w בגרף $G = (V, E)$ הם אי־שליליים, נוכל למצוא מסלולים קצרים ביותר בין כל הזוגות באמצעות הרצת האלגוריתם של דייקסטרה פעם אחת עבור כל קדקוד; כאשר תור הקדימויות ממומש על ידי ערימת פיבונצ'י, זמן הריצה של אלגוריתם כל־הזוגות הזה הוא $O(V^2 \lg V + VE)$. אם G מכיל קשתות בעלות משקלות שליליים, אנו פשוט מחשבים קבוצה חדשה של משקלות קשתות אי־שליליים המאפשרת לנו להשתמש באותה שיטה. הקבוצה החדשה של משקלות קשתות $\widehat{w}$ חייבת לקיים שתי תכונות חשובות:

1. עבור כל זוגות הקדקודים $u, v \in V$, מסלול קצר ביותר מ־u אל v כאשר פונקציית המשקל היא w הוא גם מסלול קצר ביותר מ־u אל v כאשר פונקציית המשקל היא $\widehat{w}$.
2. עבור כל הקשתות (u, v), המשקל החדש $\widehat{w}(u, v)$ הוא אי־שלילי.

כפי שנראה מיד, העיבוד המקדים של G לחישוב פונקציית המשקל החדשה $\widehat{w}$ ניתן לביצוע בזמן $O(VE)$.

שימור מסלולים קצרים ביותר על־ידי שקלול־מחדש

כפי שמראה הלמה שלהלן, קל למצוא שקלול־מחדש של קשתות המקיים את התכונה הראשונה שלעיל. נסמן ב־δ את משקליהם של מסלולים קצרים ביותר המתקבלים מפונקציית המשקל w, וב־$\widehat{\delta}$ את משקליהם של מסלולים קצרים ביותר המתקבלים מפונקציית המשקל $\widehat{w}$.

למה 26.1 (שקלול־מחדש אינו משנה מסלולים קצרים ביותר)
בהינתן גרף מכוון משוקלל $G = (V, E)$ עם פונקציית משקל $w : E \to \mathbf{R}$, תהי $h : V \to \mathbf{R}$ פונקציה כלשהי הממפה קדקודים אל מספרים ממשיים. עבור כל קשת $(u, v) \in E$, נגדיר:

$$\widehat{w}(u, v) = w(u, v) + h(u) - h(v) \tag{26.9}$$

יהי $p = \langle v_0, v_1, \ldots, v_k \rangle$ מסלול מקדקוד v_0 אל קדקוד v_k. אזי, $w(p) = \delta(v_0, v_k)$ אם ורק אם $\widehat{w}(p) = \widehat{\delta}(v_0, v_k)$. כמו כן, G מכיל מעגל בעל משקל שלילי כאשר משתמשים בפונקציית המשקל w אם ורק אם G מכיל מעגל בעל משקל שלילי כאשר משתמשים בפונקציית המשקל $\widehat{w}$.

הוכחה נתחיל בכך שנראה כי:

$$\widehat{w}(p) = w(p) + h(v_0) - h(v_k) \tag{26.10}$$

$\widehat{w}(p)$ מקיים:

$$\begin{aligned}
\widehat{w}(p) &= \sum_{i=1}^{k} \widehat{w}(v_{i-1}, v_i) \\
&= \sum_{i=1}^{k} (w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)) \\
&= \sum_{i=1}^{k} w(v_{i-1}, v_i) + h(v_0) - h(v_k) \\
&= w(p) + h(v_0) - h(v_k)
\end{aligned}$$

השוויון השלישי נובע מכך שהסכום בשורה השנייה מצטמצם טלסקופית.

עתה נראה בדרך השלילה כי $w(p) = \delta(v_0, v_k)$ גורר $\widehat{w}(p) = \widehat{\delta}(v_0, v_k)$. נניח כי כאשר משתמשים בפונקציית המשקל $\widehat{w}$, קיים מסלול קצר יותר p' מ־v_0 אל v_k. אזי, $\widehat{w}(p') < \widehat{w}(p)$. על פי משוואה (26.10):

$$\begin{aligned}
w(p') + h(v_0) - h(v_k) &= \widehat{w}(p') \\
&< \widehat{w}(p) \\
&= w(p) + h(v_0) - h(v_k)
\end{aligned}$$

ומכאן נובע כי $\widehat{w}(p') < w(p)$, בסתירה להנחתנו כי p הוא מסלול קצר ביותר מ־v_0 אל v_k כאשר משתמשים בפונקציית המשקל w. הוכחת הכיוון ההפוך — דומה.

לבסוף, נראה כי G מכיל מעגל בעל משקל שלילי כאשר משתמשים בפונקציית המשקל w אם ורק אם G מכיל מעגל שלילי כאשר משתמשים בפונקציית המשקל $\widehat{w}$. נתבונן במעגל כלשהו $c = \langle v_0, v_1, \ldots, v_k \rangle$, כאשר $v_0 = v_k$. על פי משוואה (26.10):

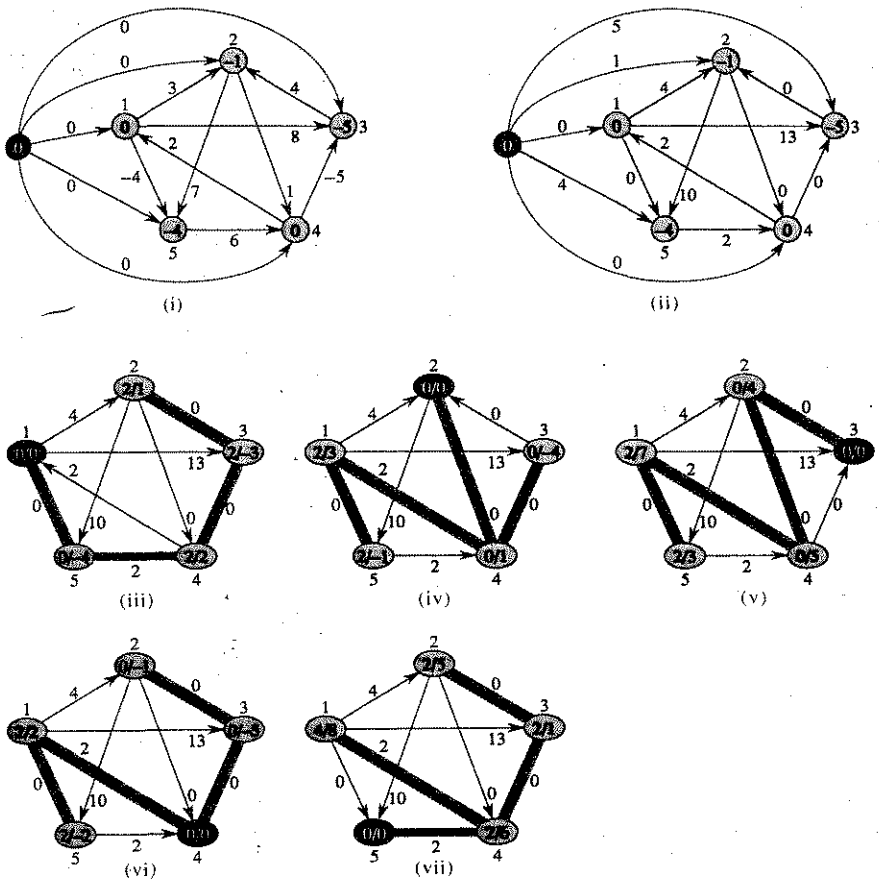
$$\begin{aligned}
\widehat{w}(c) &= w(c) + h(v_0) - h(v_k) \\
&= w(c)
\end{aligned}$$

ולכן c הוא בעל משקל שלילי כאשר משתמשים בפונקציית המשקל w אם ורק אם משקלו שלילי כאשר משתמשים בפונקציית המשקל $\widehat{w}$. ■

יצירת משקלות אי־שליליים על־ידי שקלול־מחדש

מטרתנו הבאה היא להבטיח את קיומה של התכונה השנייה: אנו רוצים כי $\widehat{w}(u, v)$ יהיה אי־שלילי עבור כל הקשתות $(u, v) \in E$. בהינתן גרף מכוון משוקלל $G = (V, E)$ עם פונקציית משקל $w : E \to \mathbf{R}$, אנו יוצרים גרף חדש $G' = (V', E')$, כאשר $V' = V \cup \{s\}$ עבור קדקוד חדש כלשהו $s \notin V$, ו־$E' = E \cup \{(s, v) : v \in V\}$. אנו מרחיבים את פונקציית המשקל w כך ש־$w(s, v) = 0$ עבור כל $v \in V$. נשים לב כי מכיוון שאין קשתות הנכנסות אל s, לא קיימים ב־G' מסלולים קצרים ביותר המכילים את s, לבד מאלה המכילים את s כמקור. יתר על כן, G' אינו מכיל מעגלים בעלי משקל שלילי אם ורק אם G אינו מכיל מעגלים בעלי משקל שלילי. איור 26.6(i) מראה את הגרף G' המתאים לגרף G מאיור 26.1.

עתה נניח כי G ו־G' אינם מכילים מעגלים בעלי משקל שלילי. הבה נגדיר $h(v) = \delta(s, v)$ עבור כל $v \in V'$. על פי למה 25.3, מתקיים $h(v) \le h(u) + w(u, v)$ עבור כל הקשתות $(u, v) \in E'$. לפיכך, אם נגדיר את המשקלות החדשים $\widehat{w}$ בהתאם למשוואה (26.9), נקבל $\widehat{w}(u, v) = w(u, v) + h(u) - h(v) \ge 0$, והתכונה השנייה תתקיים. איור 26.6(ii) מראה את הגרף G' מאיור 26.6(i) עם המשקלות החדשים של הקשתות.



איור 26.6 הרצת האלגוריתם של ג'ונסון למציאת מסלולים קצרים ביותר בין כל הזוגות על הגרף מאיור 26.1. **(i)** הגרף G' עם פונקציית המשקל המקורית w. הקדקוד השחור הוא הקדקוד s החדש. בתוך כל קדקוד v רשום $h(v) = \delta(s, v)$. **(ii)** המשקל החדש הנקבע לכל קשת (u, v) על־ידי פונקציית המשקל $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$. **(vii)-(iii)** התוצאה של הרצת האלגוריתם של דייקסטרה על כל אחד מקדקודי G, תוך שימוש בפונקציית המשקל $\hat{w}$. בכל חלק באיור, הקדקוד השחור הוא קדקוד המקור u. בתוך כל קדקוד v רשומים הערכים $\hat{\delta}(u, v)$ (משמאל) ו־$\delta(u, v)$ (מימין), מופרדים על־ידי לוכסן. הערך $d_{uv} = \delta(u, v)$ שווה ל־$\hat{\delta}(u, v) + h(v) - h(u)$.

חישוב מסלולים קצרים ביותר בין כל הזוגות

האלגוריתם של ג'ונסון למציאת מסלולים קצרים ביותר בין כל הזוגות משתמש באלגוריתם בלמן־פורד (סעיף 25.3) ובאלגוריתם של דייקסטרה (סעיף 25.2) כתת־שגרות. האלגוריתם מניח שהקשתות מאוחסנות ברשימות סמיכות. האלגוריתם מחזיר את המטריצה הרגילה $D = d_{ij}$, שממדיה $|V| \times |V|$, כאשר $d_{ij} = \delta(i, j)$, או מדווח כי גרף הקלט מכיל מעגל בעל משקל שלילי. (כדי שמשמעותם של האינדקסים המציינים תאים במטריצה D תהיה ברורה, אנו מניחים שקדקודי הגרף ממוספרים מ־1 עד $|V|$.)


```
JOHNSON(G)
 1  compute G', where V[G'] = V[G] ∪ {s} and
        E[G'] = E[G] ∪ {(s, v) : v ∈ V[G]}
 2  if BELLMAN-FORD(G', w, s) = FALSE
 3     then print "the input graph contains a negative-weight cycle"
 4     else for each vertex v ∈ V[G']
 5              do set h(v) to the value of δ(s, v)
                       computed by the Bellman-Ford algorithm
 6          for each edge (u, v) ∈ E[G']
 7              do ŵ(u, v) ← w(u, v) + h(u) − h(v)
 8          for each vertex u ∈ V[G]
 9              do run DIJKSTRA(G, ŵ, u) to compute
                       δ̂(u, v) for all v ∈ V[G]
10                 for each vertex v ∈ V[G]
11                     do d_uv ← δ̂(u, v) + h(v) − h(u)
12  return D
```

קוד זה פשוט מבצע את הפעולות שפירטנו לעיל. שורה 1 יוצרת את הגרף G'. שורה 2 מריצה את אלגוריתם בלמן־פורד על G' עם פונקציית המשקל w. אם G', ולכן גם G, מכיל מעגל בעל משקל שלילי, שורה 3 מדווחת על הבעיה. שורות 4-11 מניחות כי G' אינו מכיל מעגלים בעלי משקל שלילי. שורות 4-5 מציבות ב־$h(v)$, עבור כל $v \in V'$, את משקל המסלול הקצר ביותר $\delta(s, v)$ שחושב על ידי אלגוריתם בלמן־פורד. שורות 6-7 מחשבות את המשקלות החדשים $\hat{w}$. לולאת ה־**for** בשורות 8-11 מחשבת עבור כל זוג קדקודים $u, v \in V$ את משקל המסלול הקצר ביותר $\hat{\delta}(u, v)$ באמצעות קריאה לאלגוריתם של דייקסטרה פעם אחת מכל קדקוד ב־V. שורה 11 מאחסנת בתא d_{uv} במטריצה D את המשקל הנכון של מסלול קצר ביותר $\delta(u, v)$, המחושב באמצעות משוואה (26.10). לבסוף, שורה 12 מחזירה את המטריצה הסופית D. באיור 26.6 מודגמת פעולתו של האלגוריתם של ג'ונסון.

קל לראות שזמן הריצה של האלגוריתם של ג'ונסון הוא $O(V^2 \lg V + VE)$ אם תור הקדימויות באלגוריתם של דייקסטרה ממומש על־ידי ערימת פיבונצ'י. המימוש הפשוט יותר על־ידי ערימה בינרית מניב זמן ריצה של $O(VE \lg V)$; דהיינו, אם הגרף דליל, האלגוריתם עדיין מהיר יותר אסימפטוטית מאלגוריתם פלויד־וורשאל.

תרגילים

26.3-1

השתמש באלגוריתם של ג'ונסון כדי למצוא את המסלולים הקצרים ביותר בין כל זוגות הקדקודים בגרף שבאיור 26.2. הצג את הערכים h ו־$\hat{w}$ שמחשב האלגוריתם.

26.3-2

לשם מה מוסיפים ל־V את הקדקוד החדש s ויוצרים בכך את V'?

26.3-3

נניח כי $w(u, v) \geq 0$ עבור כל הקשתות $(u, v) \in E$. מהו הקשר בין פונקציות המשקל w ו־$\hat{w}$?

* 26.4 מסגרת כללית לפתרון בעיות מסלולים בגרפים מכוונים

בסעיף זה נבחן "חוגים־למחצה סגורים" — מבנה אלגברי המספק מסגרת כללית לפתרון בעיות מסלולים בגרפים מכוונים. נתחיל בהגדרת חוגים־למחצה סגורים ונראה את הקשר שלהם לתחשיב של מסלולים מכוונים. לאחר מכן נביא כמה דוגמאות לחוגים־למחצה סגורים וכן אלגוריתם "כללי" לחישוב מידע על מסלולים בין כל הזוגות. הן אלגוריתם פלויד־וורשאל והן אלגוריתם הסגור הטרנזיטיבי שהוצגו בסעיף 26.2 הם מקרים פרטיים של אלגוריתם כללי זה.

הגדרה של חוג־למחצה סגור

חוג־למחצה סגור (closed semiring) הוא מערכת $(S, \oplus, \odot, \bar{0}, \bar{1})$, כאשר S היא קבוצת איברים, $\oplus$ (**אופרטור הסיכום**, summary operator) ו־$\odot$ (**אופרטור ההרחבה**, extension operator) הם אופרטורים בינריים על S, ו־$\bar{0}$ ו־$\bar{1}$ הם איברים של S, המערכת מקיימת את שמונה התכונות הבאות:

1. $(S, \oplus, \bar{0})$ הוא **מונואיד** (monoid):
   - S **סגורה** תחת $\oplus$: $a \oplus b \in S$ עבור כל $a, b \in S$.
   - $\oplus$ **אסוציאטיבי**: $a \oplus (b \oplus c) = (a \oplus b) \oplus c$ עבור כל $a, b, c \in S$.
   - $\bar{0}$ הוא **איבר היחידה** עבור $\oplus$: $a \oplus \bar{0} = \bar{0} \oplus a = a$ עבור כל $a \in S$.

   באופן דומה, $(S, \odot, 1)$ הוא מונואיד.
2. $\bar{0}$ הוא **מאפס**: $a \odot \bar{0} = \bar{0} \odot a = \bar{0}$ עבור כל $a \in S$.
3. $\oplus$ **קומוטטיבי**: $a \oplus b = b \oplus a$ עבור כל $a, b \in S$.
4. $\oplus$ **אידמפוטנטי**: $a \oplus a = a$ עבור כל $a \in S$.
5. $\odot$ **מתפלג** מעל $\oplus$: $a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c)$ וכן
   $(b \oplus c) \odot a = (b \odot a) \oplus (c \odot a)$ עבור כל $a, b, c \in S$.
6. אם $a_1, a_2, a_3, \ldots$ היא סדרה בת־מנייה של איברים מ־S, אזי $a_1 \oplus a_2 \oplus a_3 \oplus \cdots$ מוגדר היטב ונמצא ב־S.
7. אסוציאטיביות, קומוטטיביות ואידמפוטנטיות מתקיימות גם עבור סיכומים אינסופיים. (לכן, כל סיכום אינסופי ניתן לכתיבה מחדש כסיכום אינסופי שבו כל איבר בסיכום מופיע בדיוק פעם אחת וסדר הסיכום הוא שרירותי.)
8. $\odot$ מתפלג מעל סיכומים אינסופיים:

$$a \odot (b_1 \oplus b_2 \oplus b_3 \oplus \cdots) = (a \odot b_1) \oplus (a \odot b_2) \oplus (a \odot b_3) \oplus \cdots$$

וכן:

$$(a_1 \oplus a_2 \oplus a_3 \oplus \cdots) \odot b = (a_1 \odot b) \oplus (a_2 \odot b) \oplus (a_3 \odot b) \oplus \cdots$$

תחשיב של מסלולים בגרפים מכוונים

אף כי התכונות של חוג־למחצה סגור עשויות להיראות מופשטות, ניתן למצוא קשר בינן לבין תחשיב של מסלולים בגרפים מכוונים. נניח שנתון לנו גרף מכוון $G = (V, E)$ ו**פונקציית שיוך תוויות** $\lambda : V \times V \to S$ הממפה את כל הזוגות הסדורים של קדקודים לתוך קו־תחום כלשהו S. את **תווית הקשת** $(u, v) \in E$ נסמן ב־$\lambda(u, v)$. מאחר ש־λ מוגדרת מעל התחום $V \times V$, התווית $\lambda(u, v)$ נקבעת בדרך כלל כ־$\bar{0}$ אם (u, v) איננה קשת של G (מיד נראה מדוע).

אנו משתמשים באופרטור ההרחבה האסוציאטיבי $\odot$ להרחבת מושג התווית גם למסלולים. **תווית של מסלול** $p = \langle v_1, v_2, \ldots, v_k \rangle$ היא:

$$\lambda(p) = \lambda(v_1, v_2) \odot \lambda(v_2, v_3) \odot \cdots \odot \lambda(v_{k-1}, v_k)$$

איבר היחידה $\bar{1}$ עבור $\odot$ משמש כתווית של המסלול הריק.

בדוגמה שנביא כאן ובהמשך ליישום של חוגים־למחצה סגורים, נשתמש במסלולים קצרים ביותר עם משקלות אי־שליליים של קשתות. הקו־תחום S הוא $\mathbf{R}^{\geq 0} \cup \{\infty\}$, כאשר $\mathbf{R}^{\geq 0}$ היא קבוצת הממשיים האי־שליליים, ו־$\lambda(i, j) = w_{ij}$ עבור כל $i, j \in V$. אופרטור ההרחבה $\odot$ מתאים לאופרטור האריתמטי $+$, ולכן התווית של מסלול $p = \langle v_1, v_2, \ldots, v_k \rangle$ היא:

$$\begin{aligned}\lambda(p) &= \lambda(v_1, v_2) \odot \lambda(v_2, v_3) \odot \cdots \odot \lambda(v_{k-1}, v_k) \\ &= w_{v_1, v_2} + w_{v_2, v_3} + \cdots + w_{v_{k-1}, v_k} \\ &= w(p)\end{aligned}$$

שלא במפתיע, את תפקידו של $\bar{1}$, איבר היחידה עבור $\odot$, ממלא 0, איבר היחידה עבור $+$. את המסלול הריק אנו מסמנים ב־ε, ותוויתו היא $\lambda(\varepsilon) = w(\varepsilon) = 0 = \bar{1}$.

מאחר שאופרטור ההרחבה $\odot$ הוא אסוציאטיבי, נוכל להגדיר את התווית של שרשור של שני מסלולים באופן טבעי. בהינתן המסלולים $p_1 = \langle v_1, v_2, \ldots, v_k \rangle$ ו־$p_2 = \langle v_k, v_{k+1}, \ldots, v_l \rangle$, **השרשור** (concatenation) שלהם הוא:

$$p_1 \circ p_2 = \langle v_1, v_2, \ldots, v_k, v_{k+1}, \ldots, v_l \rangle$$

והתווית של השרשור שלהם היא:

$$\begin{aligned}\lambda(p_1 \circ p_2) &= \lambda(v_1, v_2) \odot \lambda(v_2, v_3) \odot \cdots \odot \lambda(v_{k-1}, v_k) \odot \\ &\qquad \lambda(v_k, v_{k+1}) \odot \lambda(v_{k+1}, v_{k+2}) \odot \cdots \odot \lambda(v_{l-1}, v_l) \\ &= (\lambda(v_1, v_2) \odot \lambda(v_2, v_3) \odot \cdots \odot \lambda(v_{k-1}, v_k)) \odot \\ &\qquad (\lambda(v_k, v_{k+1}) \odot \lambda(v_{k+1}, v_{k+2}) \odot \cdots \odot \lambda(v_{l-1}, v_l)) \\ &= \lambda(p_1) \odot \lambda(p_1)\end{aligned}$$

אופרטור הסיכום $\oplus$, שהוא קומוטטיבי וגם אסוציאטיבי, משמש **לסיכום** תוויות מסלולים. כלומר, הערך $\lambda(p_1) \oplus \lambda(p_2)$ מציין סיכום, אשר המשמעות הסמנטית שלו נקבעת על פי היישום, של תוויות המסלולים p_1 ו־p_2.

מטרתנו היא לחשב, עבור כל זוגות הקדקודים $i, j \in V$, את הסיכום של כל תוויות המסלולים מ־i אל j:

$$l_{ij} = \bigoplus_{i \overset{p}{\rightsquigarrow} j} \lambda(p) \tag{26.11}$$

אנו דורשים מ־$\oplus$ הן קומוטטיביות והן אסוציאטיביות מכיוון שלא צריכה להיות חשיבות לסדר שבו אנו מסכמים את המסלולים. מאחר שאנו משתמשים במאפס $\bar{0}$ כתוויותו של זוג סדור (u, v) שאינו קשת בגרף, כל מסלול המנסה לעבור דרך קשת חסרה הוא בעל התווית $\bar{0}$.

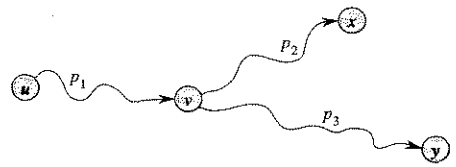
עבור מסלולים קצרים ביותר, אנו משתמשים ב־$\min$ כאופרטור הסיכום $\oplus$. איבר היחידה עבור $\min$ הוא ∞, ואכן ∞ הוא מאפס עבור $+$: $a + \infty = \infty + a = \infty$ עבור כל $a \in \mathbf{R}^{\geq 0} \cup \{\infty\}$. קשתות חסרות הן בעלות משקל ∞, ואם מסלול מכיל קשת שמשקלה ∞, זהו גם משקלו של המסלול כולו.

אנו רוצים שאופרטור הסיכום $\oplus$ יהיה אידמפוטנטי, מכיוון שעל פי משוואה (26.11) אנו רואים כי $\oplus$ צריך לסכם את התוויות של קבוצת מסלולים. אם p הוא מסלול, אזי $\{p\} = \{p\} \cup \{p\}$; אם אנו מסכמים את p עם עצמו, התווית המתקבלת צריכה להיות התווית של p: $\lambda(p) \oplus \lambda(p) = \lambda(p)$.

מאחר שהמסלולים שאנו בוחנים עשויים להיות מסלולים לא פשוטים, ייתכן שמספר המסלולים בגרף יהיה אינסופי בן־מנייה. (כל מסלול, פשוט או לא פשוט, מכיל מספר סופי של קשתות.) משום כך דרוש שניתן יהיה להפעיל את האופרטור $\oplus$ גם עבור מספר אינסופי בן־מנייה של תוויות מסלולים. כלומר, אם $a_1, a_2, a_3, \ldots$ היא סדרה בת־מנייה של איברים בקו־תחום S, אזי התווית $a_1 \oplus a_2 \oplus a_3 \oplus \cdots$ צריכה להיות מוגדרת היטב ושייכת ל־S. לא צריכה להיות חשיבות לסדר שבו מסכמים תוויות מסלולים, ולכן אסוציאטיביות וקומוטטיביות צריכות להתקיים עבור סיכומים אינסופיים. יתר על כן, אם אנו מסכמים את אותה תווית מסלול a עם עצמה מספר אינסופי בן מנייה של פעמים, אנו צריכים לקבל את התוצאה a, ולפיכך האידמפוטנטיות צריכה להתקיים עבור סיכומים אינסופיים.

נחזור עתה לדוגמת המסלולים הקצרים ביותר, ונשאל אם ניתן להפעיל את $\min$ על סדרה אינסופית של ערכים ב־$\mathbf{R}^{\geq 0} \cup \{\infty\}$. לדוגמה, האם ערכו של $\min_{k=1}^{\infty} \{1/k\}$ מוגדר היטב? הוא אכן מוגדר היטב אם אנו חושבים על $\min$ כעל אופרטור אשר מחזיר למעשה את החסם התחתון הגדול ביותר (infimum) של הארגומנטים שלו, ובמקרה כזה אנו מקבלים $\min_{k=1}^{\infty} \{1/k\} = 0$.

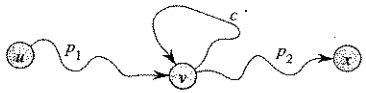
כדי לחשב תוויות של מסלולים מתבדרים, אנו זקוקים להתפלגות של אופרטור ההרחבה $\odot$ מעל אופרטור הסיכום $\oplus$. כפי שניתן לראות באיור 26.7, נניח שנתונים לנו המסלולים $u \overset{p_1}{\rightsquigarrow} v$, $v \overset{p_2}{\rightsquigarrow} x$ ו־$v \overset{p_3}{\rightsquigarrow} y$. עקב ההתפלגות של $\odot$ מעל $\oplus$ אנו יכולים לסכם את תוויות המסלולים $p_1 \circ p_2$ ו־$p_1 \circ p_3$ בשתי דרכים: $(\lambda(p_1) \odot \lambda(p_2)) \oplus (\lambda(p_1) \odot \lambda(p_3))$ או $\lambda(p_1) \odot (\lambda(p_2) \oplus \lambda(p_3))$.



איור 26.7 שימוש בהתפלגות של $\odot$ מעל $\oplus$. את תוויות המסלולים $p_1 \circ p_2$ ו־$p_1 \circ p_3$ ניתן לסכם בשתי דרכים: $(\lambda(p_1) \odot \lambda(p_2)) \oplus (\lambda(p_1) \odot \lambda(p_3))$ או $\lambda(p_1) \odot (\lambda(p_2) \oplus \lambda(p_3))$.

מכיוון שמספר המסלולים בגרף עשוי להיות אינסופי בן־מנייה, $\odot$ צריך להתפלג מעל סיכומים אינסופיים כמו גם מעל סיכומים סופיים. איור 26.8, לדוגמה, מכיל מסלולים $u \overset{p_1}{\rightsquigarrow} v$ ו־$x \overset{p_2}{\rightsquigarrow} x$, יחד עם המעגל $v \overset{c}{\rightsquigarrow} v$. עלינו להיות מסוגלים לסכם את המסלולים $p_1 \circ p_2$, $p_1 \circ c \circ p_2$, $p_1 \circ c \circ c \circ p_2$, $\ldots$. מהתפלגותו של $\odot$ מעל סיכומים אינסופיים בני־מנייה אנו מקבלים:

$$\begin{aligned}
&(\lambda(p_1) \odot \lambda(p_2)) \oplus (\lambda(p_1) \odot \lambda(c) \odot \lambda(p_2)) \\
&\quad \oplus (\lambda(p_1) \odot \lambda(c) \odot \lambda(c) \odot \lambda(p_2)) \oplus \cdots \\
&= \lambda(p_1) \odot (\lambda(p_2) \oplus (\lambda(c) \odot \lambda(p_2)) \\
&\quad \oplus (\lambda(c) \odot \lambda(c) \odot \lambda(p_2)) \oplus \cdots) \\
&= \lambda(p_1) \odot (\bar{1} \oplus \lambda(c) \oplus (\lambda(c) \odot \lambda(c)) \\
&\quad \oplus (\lambda(c) \odot \lambda(c) \odot \lambda(c)) \oplus \cdots) \odot \lambda(p_2)
\end{aligned}$$



איור 26.8 התפלגות $\odot$ מעל $\oplus$ עבור סיכומים אינסופיים בני מנייה. בשל המעגל c, קיים מספר אינסופי בן־מנייה של מסלולים מקדקוד v אל קדקוד x. עלינו להיות מסוגלים לסכם את המסלולים $p_1 \circ p_2$, $p_1 \circ c \circ p_2$, $p_1 \circ c \circ c \circ p_2$, $\ldots$.

אנו משתמשים בסימון מיוחד כדי לסמן תווית של מעגל שניתן לעבור לאורכו מספר פעמים רב כרצוננו. נניח שנתון לנו מעגל c שתוויתו $\lambda(c) = a$. אנו יכולים לעבור לאורך המעגל אפס פעמים, ולקבל את התווית $\lambda(\varepsilon) = \bar{1}$; לעבור לאורכו פעם אחת, ולקבל את התווית $\lambda(c) = a$, לעבור פעמיים ולקבל את התווית $\lambda(c) \odot \lambda(c) = a \odot a$ וכן הלאה. התווית המתקבלת על־ידי סיכום מספר אינסופי זה של מעברים לאורך המעגל c היא **הסְגוֹר** (closure) של a, המוגדר על־ידי:

$$a^* = \bar{1} \oplus a \oplus (a \odot a) \oplus (a \odot a \odot a) \oplus (a \odot a \odot a \odot a) \oplus \cdots$$

כך אפוא, בדוגמה שבאיור 26.8, ברצוננו לחשב את $\lambda(p_1) \odot (\lambda(c))^* \odot \lambda(p_2)$.

בדוגמת המסלולים הקצרים ביותר, עבור כל מספר ממשי אי־שלילי $a \in \mathbf{R}^{\geq 0} \cup \{\infty\}$ מתקיים:

$$\begin{aligned}
a^* &= \min_{k=0}^{\infty} \{ka\} \\
&= 0
\end{aligned}$$

פירושה של תכונה זו הוא, שמכיוון שכל המעגלים הם בעלי משקל אי־שלילי, מסלול קצר ביותר לעולם אינו עובר לאורך מעגל שלם.

חוגים־למחצה סגורים: דוגמאות

כבר ראינו דוגמה אחת של חוג־למחצה סגור, והיא המערכת $S_1 = (\mathbf{R}^{\geq 0} \cup \{\infty\}, \min, +, \infty, 0)$, שבה השתמשנו עבור מסלולים קצרים ביותר עם משקלות קשתות אי־שליליים. (כפי שציינו לעיל, האופרטור $\min$ מחזיר למעשה את החסם התחתון הגדול ביותר של הארגומנטים שלו.) הראינו גם שמתקיים $a^* = 0$ לכל $a \in \mathbf{R}^{\geq 0} \cup \{\infty\}$.

אולם טענו שגם אם קיימות קשתות בעלות משקל שלילי, אלגוריתם פלויד־וורשאל מחשב את משקליהם של מסלולים קצרים ביותר כל עוד לא קיימים בגרף מעגלים בעלי משקל שלילי. על־ידי הוספת אופרטור הסגור המתאים והרחבת הקרן־תחום של התוויות ל־$\mathbf{R} \cup \{-\infty, +\infty\}$, נוכל למצוא חוג־למחצה סגור לטיפול במעגלים בעלי משקל שלילי. תוך שימוש ב־min עבור $\oplus$ וב־+ עבור $\odot$, יכול הקורא לוודא כי הסגור של $a \in \mathbf{R} \cup \{-\infty, +\infty\}$ הוא:

$$a^* = \begin{cases} 0 & \text{אם } a \geq 0 \\ -\infty & \text{אם } a < 0 \end{cases}$$

המקרה השני ($a < 0$) מייצג את המצב שבו ניתן לעבור לאורך מעגל בעל משקל שלילי מספר אינסופי של פעמים כדי לקבל את המשקל $-\infty$ עבור כל מסלול המכיל את המעגל. אם כך, החוג־למחצה הסגור שיש להשתמש בו עבור אלגוריתם פלויד־וורשאל על גרף המכיל קשתות בעלות משקל שלילי הוא $S_2 = (\mathbf{R} \cup \{-\infty, +\infty\}, \min, +, +\infty, 0)$. (ראה תרגיל 26.4-3.)

עבור סגור טרנזיטיבי, אנו משתמשים בחוג־למחצה הסגור $S_3 = (\{0,1\}, \vee, \wedge, 0, 1)$, כאשר $\lambda(i,j) = 1$ אם $(i,j) \in E$, ו־$\lambda(i,j) = 0$ אחרת. כאן מתקיים $1^* = 1 = 0^*$.

אלגוריתם תכנון־דינמי עבור תוויות של מסלולים מכוונים

נניח שנתון לנו גרף מכוון $G = (V, E)$ עם פונקציית שיוך תוויות $\lambda : V \times V \to S$. קדקודי הגרף ממוספרים מ־1 עד n. עבור כל זוג קדקודים $i, j \in V$, ברצוננו לחשב את משוואה (26.11):

$$l_{ij} = \bigoplus_{i \xrightarrow{p} j} \lambda(p)$$

שהיא תוצאת סיכום כל המסלולים מ־i אל j באמצעות אופרטור הסיכום $\oplus$. עבור מסלולים קצרים ביותר, למשל, ברצוננו לחשב את:

$$l_{ij} = \delta(i,j) = \min_{i \xrightarrow{p} j} \{w(p)\}$$

קיים אלגוריתם תכנון־דינמי הפותר בעיה זו, והוא דומה מאוד במבנהו לאלגוריתם פלויד־וורשאל ולאלגוריתם הסגור הטרנזיטיבי. תהי $Q_{ij}^{(k)}$ קבוצת המסלולים מקדקוד i אל קדקוד j, כך שכל קדקודי הביניים שייכים לקבוצה $\{1, 2, \ldots, k\}$. נגדיר:

$$l_{ij}^{(k)} = \bigoplus_{p \in Q_{ij}^{(k)}} \lambda(p)$$

נשים לב להקבלה בין הגדרה זו להגדרת $d_{ij}^{(k)}$ באלגוריתם פלויד־וורשאל ולהגדרת $t_{ij}^{(k)}$ באלגוריתם הסגור הטרנזיטיבי. ניתן להגדיר את $l_{ij}^{(k)}$ באופן רקורסיבי על־ידי:

$$l_{ij}^{(k)} = l_{ij}^{(k-1)} \oplus \left(l_{ik}^{(k-1)} \odot (l_{kk}^{(k-1)})^* \odot l_{kj}^{(k-1)}\right) \tag{26.12}$$

נוסחת הנסיגה (26.12) מזכירה את נוסחאות הנסיגה (26.5) ו־(26.8), אולם נכלל בה גורם נוסף: $(l_{kk}^{(k-1)})^*$. גורם זה מייצג את סיכום כל המעגלים אשר עוברים דרך הקדקוד k ואשר כל קדקודיהם האחרים שייכים לקבוצה $\{1, 2, \ldots, k-1\}$. (כאשר מניחים, באלגוריתם פלויד־וורשאל, שלא קיימים בגרף מעגלים בעלי משקל שלילי, אזי $(l_{kk}^{(k-1)})^*$ הוא 0, המתאים ל־$\bar{1}$, משקלו של מעגל ריק. באלגוריתם הסגור הטרנזיטיבי, המסלול הריק מ־k אל k נותן $(l_{kk}^{(k-1)})^* = 1 = \bar{1}$). לכן, בשני האלגוריתמים האלה נוכל להתעלם מן הגורם $(l_{kk}^{(k-1)})^*$, שאינו אלא איבר היחידה עבור $\odot$.) הבסיס של ההגדרה הרקורסיבית הוא:

$$l_{ij}^{(0)} = \begin{cases} \lambda(i,j) & \text{אם } i \neq j \\ \bar{1} \oplus \lambda(i,j) & \text{אם } i = j \end{cases}$$

שניתן לראותו באופן הבא: תוויתו של המסלול בן הקשת האחת $\langle i, j \rangle$ היא פשוט $\lambda(i,j)$ (השווה ל־$\bar{0}$ אם (i,j) אינה קשת ב־E). אם, נוסף על כך, $i = j$, אזי $\bar{1}$ היא תוויתו של המסלול הריק מ־i אל i.

אלגוריתם התכנון־הדינמי מחשב את הערכים $l_{ij}^{(k)}$ בסדר עולה של k. הוא מחזיר את המטריצה $L^{(n)} = (l_{ij}^{(n)})$.

```
COMPUTE-SUMMARIES(λ, V)
 1  n ← |V|
 2  for i ← 1 to n
 3      do for j ← 1 to n
 4          do if i = j
 5              then l_ij^(0) ← 1̄ ⊕ λ(i,j)
 6              else l_ij^(0) ← λ(i,j)
 7  for k ← 1 to n
 8      do for i ← 1 to n
 9          do for j ← 1 to n
10              do l_ij^(k) ← l_ij^(k-1) ⊕ (l_ik^(k-1) ⊙ (l_kk^(k-1))* ⊙ l_kj^(k-1))
11  return L^(n)
```

זמן הריצה של אלגוריתם זה תלוי בזמני הביצוע של הפעולות $\odot$, $\oplus$ ו־*. אם נסמן זמנים אלה ב־$T_\odot$, $T_\oplus$ ו־T_*, בהתאמה, אזי זמן הריצה של COMPUTE-SUMMARIES הוא $\Theta(n^3(T_\odot + T_\oplus + T_*))$, שהוא $\Theta(n^3)$ אם כל אחת משלוש הפעולות הללו מתבצעת בזמן $O(1)$.

תרגילים

26.4-1

ודא כי $S_1 = (\mathbf{R}^{\geq 0} \cup \{\infty\}, \min, +, \infty, 0)$ ו־$S_3 = (\{0,1\}, \vee, \wedge, 0, 1)$ הם חוגים־למחצה סגורים.

26.4-2

ודא כי $S_2 = (\mathbf{R} \cup \{-\infty, +\infty\}, \min, +, +\infty, 0)$ הוא חוג־למחצה סגור. מהו ערך הביטוי $a + (-\infty)$ עבור $a \in \mathbf{R}$? מהו ערך הביטוי $(+\infty) + (-\infty)$?

26.4-3

כתוב מחדש את השגרה COMPUTE-SUMMARIES כך שתשתמש בחוג־למחצה סגור S_2 ותממש את אלגוריתם פלויד־וורשאל. מה צריך להיות הערך של $\infty + \infty$?

26.4-4

האם המערכת $S_4 = (\mathbf{R}, +, \cdot, 0, 1)$ היא חוג־למחצה סגור?

26.4-5

האם ניתן להשתמש בחוג־למחצה סגור כלשהו עבור האלגוריתם של דייקסטרה? האם ניתן לעשות זאת עבור אלגוריתם בלמן־פורד? מה באשר לשגרה FASTER-ALL-PAIRS-SHORTEST-PATHS?

26.4-6

חברת משאיות מעוניינת לשלוח משאית מקאסטרוביל לבוסטון כשהיא עמוסה במטען גדול ככל האפשר של ארטישוקים. אולם בארה"ב, בכל כביש קיים גבול של משקל מקסימלי למשאיות המשתמשות בכביש זה. יצג בעיה זו באמצעות גרף מכוון $G = (V, E)$ וחוג־למחצה סגור מתאים, וכתוב אלגוריתם יעיל לפתרון הבעיה.

בעיות

26-1 סגור טרנזיטיבי של גרף דינמי

נניח שברצוננו לתחזק את הסגור הטרנזיטיבי של גרף מכוון $G = (V, E)$ במהלך הכנסתן של קשתות ל־E. כלומר, לאחר הכנסתה של כל קשת, ברצוננו לעדכן את הסגור הטרנזיטיבי של הקשתות שהוכנסו עד כה. נניח שבמצבו ההתחלתי, הגרף G אינו מכיל קשתות כלל וכי יש לייצג את הסגור הטרנזיטיבי על ידי מטריצה בוליאנית.

א. הראה כיצד ניתן לעדכן בזמן $O(V^2)$ את הסגור הטרנזיטיבי $G^* = (V, E^*)$ של גרף $G = (V, E)$ בעקבות הוספתה של קשת חדשה ל־G.

ב. הבא דוגמה לגרף G וקשת e שעבורם דרוש זמן של $\Omega(V^2)$ לעדכון הסגור הטרנזיטיבי לאחר הכנסת e ל־G.

ג. תאר אלגוריתם יעיל לעדכון הסגור הטרנזיטיבי במהלך הכנסתן של קשתות לגרף. עבור סדרה כלשהי של n הכנסות, האלגוריתם שלך צריך לרוץ בזמן כולל של $\sum_{i=1}^{n} t_i = O(V^3)$, כאשר t_i הוא הזמן לעדכון הסגור הטרנזיטיבי בעת שמכניסים לגרף את הקשת ה־i. הוכח שהאלגוריתם שלך משיג את חסם הזמן הזה.

26-2 מסלולים קצרים ביותר בגרפים ϵ־צפופים

גרף $G = (V, E)$ נקרא **ϵ־צפוף** (ϵ-dense) אם $|E| = \Theta(V^{1+\epsilon})$ עבור איזשהו קבוע ϵ בתחום $0 < \epsilon \le 1$. על־ידי שימוש בערימות d־יות (ראה בעיה 2-7) באלגוריתמים למציאת מסלולים קצרים ביותר על גרפים ϵ־צפופים, ניתן להשיג זמן ריצה המשתווה לזה של אלגוריתמים המבוססים על ערימות פיבונצ'י, בלי להשתמש במבני נתונים מסובכים כל כך.

א. מהם זמני הריצה האסימפטוטיים של הפעולות INSERT, EXTRACT-MIN ו־DECREASE-KEY כפונקציה של d ושל n, מספר האיברים בערימה d־ית? מהם זמני ריצה אלה אם בוחרים $d = \Theta(n^\alpha)$ עבור איזשהו קבוע $0 < \alpha \le 1$? השווה זמני ריצה אלה לעלויות לשיעורין של פעולות אלה על ערימת פיבונצ'י.

ב. הראה כיצד ניתן לחשב בזמן $O(E)$ מסלולים קצרים ביותר ממקור יחיד בגרף מכוון ϵ־צפוף $G = (V, E)$ ללא קשתות בעלות משקל שלילי. (**רמז:** בחר d שהוא פונקציה של ϵ.)

ג. הראה כיצד ניתן לפתור בזמן $O(VE)$ את בעיית המסלולים הקצרים ביותר בין כל הזוגות בגרף מכוון ϵ־צפוף $G = (V, E)$ שאין בו קשתות בעלות משקל שלילי.

ד. הראה כיצד ניתן לפתור בזמן $O(VE)$ את בעיית המסלולים הקצרים ביותר בין כל הזוגות בגרף מכוון ϵ־צפוף שעשוי להכיל קשתות בעלות משקל שלילי אולם אינו מכיל מעגלים בעלי משקל שלילי.

26-3 עץ פורש מינימלי כחוג־למחצה סגור

יהי $G = (V, E)$ גרף בלתי־מכוון קשיר עם פונקציית משקל $w : E \to \mathbf{R}$. תהי קבוצת הקדקודים $V = \{1, 2, \ldots, n\}$, כאשר $n = |V|$, ונניח שכל משקלות הקשתות $w(i, j)$ שונים זה מזה. יהי T העץ הפורש המינימלי היחיד (ראה תרגיל 24.1-6) של G. בבעיה זו, נבנה את T תוך שימוש בחוג־למחצה סגור, כפי שהציעו B. M. Maggs ו־S. A. Plotkin. תחילה נקבע, עבור כל זוג קדקודים $i, j \in V$, את המשקל **המינימקסימלי** (minimax):

$$m_{ij} = \min_{i \xrightarrow{p} j} \max_{\text{edges } e \text{ on } p} w(e)$$

א. הצדק בקצרה את הקביעה שהמערכת $S = (\mathbf{R} \cup \{-\infty, +\infty\}, \min, \max, \infty, -\infty)$ היא חוג־למחצה סגור.

מאחר ש־S הוא חוג־למחצה סגור, ניתן להשתמש בשגרה COMPUTE-SUMMARIES כדי לקבוע את המשקלים המינימקסימליים m_{ij} בגרף G. יהי $m_{ij}^{(k)}$ המשקל המינימקסימלי על כל המסלולים מקדקוד i אל קדקוד j אשר כל קדקודי הביניים בהם שייכים לקבוצה $\{1, 2, \ldots, k\}$.

ב. כתוב נוסחת נסיגה עבור $m_{ij}^{(k)}$, כאשר $k \ge 0$.

ג. יהי $T_m = \{(i, j) \in E : w(i, j) = m_{ij}\}$. הוכח שהקשתות ב־$T_m$ יוצרות עץ פורש של G.

ד. הראה כי $T_m = T$. (**רמז:** בחן את ההשפעה של הוספת קשת (i, j) ל־T והוצאתה של קשת הנמצאת על מסלול אחר מ־i אל j. כמו כן, בחן את ההשפעה של הוצאת קשת (i, j) מ־T והחלפתה בקשת אחרת.)

הערות ביבליוגרפיות לפרק 26

ב־Lawler [132] ניתן למצוא דיון טוב בבעיית המסלולים הקצרים ביותר בין כל הזוגות, אף כי הוא אינו מנתח את הפתרונות עבור גרפים דלילים. הוא מייחס את אלגוריתם כפל המטריצות לפולקלור. אלגוריתם פלויד־וורשאל פותח בידי Floyd [68], אשר ביסס את האלגוריתם על משפט של Warshall [198]. המתאר כיצד לחשב את הסגור הטרנזיטיבי של מטריצות בוליאניות. המבנה האלגברי חוג־למחצה סגור מופיע אצל Aho, Hopcroft & Ullman [4]. האלגוריתם של Johnson לקוח מ־[114].

27 זרימה מקסימלית

כשם שניתן לייצג מפת דרכים באמצעות גרף מכוון כדי למצוא את המסלול הקצר ביותר מנקודה אחת לאחרת, ניתן גם לפרש גרף מכוון כ"רשת זרימה" ולהשתמש בו כדי לענות על שאלות העוסקות בזרימת חומר. נתאר לעצמנו חומר העובר דרך מערכת ממקור, שם מיוצר החומר, לבור, שם הוא נצרך. המקור מייצר את החומר בקצב קבוע כלשהו והבור צורך את החומר באותו קצב. אינטואיטיבית, "זרימת" החומר בכל נקודה במערכת היא הקצב שבו החומר נע. ניתן להשתמש ברשתות זרימה כדי לייצג זרימת נוזל דרך צינורות, תנועת רכיבים לאורך קו ייצור, מעבר זרם דרך רשתות חשמל, מעבר מידע דרך רשתות תקשורת, וכן הלאה.

על כל קשת מכוונת ברשת זרימה ניתן לחשוב כעל מוליך של חומר. כל מוליך הוא בעל קיבול ידוע, הנתון על־ידי הקצב המקסימלי שבו יכול חומר לזרום דרך המוליך, כמו למשל, 200 ליטרים לשעה של נוזל דרך צינור, או 20 אמפר של זרם חשמלי דרך תיל מוליך. קדקודים הם נקודות הצטלבות של מוליכים, וחוץ מאשר במקור ובבור, חומר זורם דרכם אך אינו נאסף בהם. במילים אחרות, הקצב שבו חומר נכנס לקדקוד חייב להיות שווה לקצב שבו החומר יוצא מן הקדקוד. תכונה זו נקראת "שימור הזרימה". כאשר החומר הוא זרם חשמלי, זהו בדיוק חוק הזרמים של קירכהוף (Kirchhoff's Current Law).

בעיית הזרימה המקסימלית היא הבעיה הפשוטה ביותר הנוגעת לרשתות זרימה. היא שואלת, "מהו הקצב הגבוה ביותר שבו ניתן להעביר חומר מן המקור אל הבור בלי להפר שום אילוצי קיבולו" כפי שנראה בפרק זה, קיימים אלגוריתמים יעילים לפתרון הבעיה. יתירה מזו, את הטכניקות הבסיסיות שאלגוריתמים אלה משתמשים בהן ניתן להתאים לפתרון בעיות אחרות של רשתות זרימה.

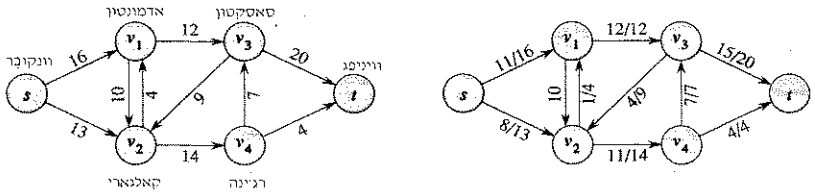
בפרק זה נציג שתי שיטות כלליות לפתרון בעיית הזרימה המקסימלית. סעיף 27.1 מביא ניסוח פורמלי של המושגים רשתות זרימה וזרימות, ומגדיר באופן פורמלי את בעיית הזרימה המקסימלית. סעיף 27.2 מתאר את השיטה הקלסית של פורד ופולקרסון (Ford & Fulkerson) למציאת זרימה מקסימלית. סעיף 27.3 מציג יישום של שיטה זו: מציאת זיווג מקסימלי בגרף דו־צדדי בלתי־מכוון. סעיף 27.4 מציג את שיטת "דחיפת הקדם־זרימה" העומדת בבסיסם של רבים מן האלגוריתמים המהירים ביותר עבור בעיות של רשתות זרימה. סעיף 27.5 דן באלגוריתם "הרם־וקדֵם", המהווה מימוש מסוים של שיטת דחיפת הקדם־זרימה שרץ בזמן $O(V^3)$. אלגוריתם זה, אף שאינו האלגוריתם המהיר ביותר המוכר כיום, מדגים כמה מן השיטות שמשתמשים בהן האלגוריתמים המהירים ביותר אסימפטוטית, והוא גם יעיל למדי ביישומים מעשיים.

27.1 רשתות זרימה

בסעיף זה נגדיר רשתות זרימה במונחי תורת הגרפים, נדון בתכונותיהן, ונגדיר במדויק את בעיית הזרימה המקסימלית. כמו כן, נציג כמה סימונים שימושיים.

רשתות זרימה וזרימות

רשת זרימה (flow network) $G = (V, E)$ היא גרף מכוון שבו כל קשת $(u, v) \in E$ היא בעלת **קיבול** (capacity) אי־שלילי $c(u, v) \neq 0$. אם $(u, v) \notin E$, אנו מניחים כי $c(u, v) = 0$. ברשת זרימה קיימים שני קדקודים מיוחדים: **מקור** (source) s ו**בור** (sink) t. מטעמי נוחות, אנו מניחים שכל קדקוד שוכן על מסלול כלשהו מן המקור אל הבור. כלומר, עבור כל קדקוד $v \in V$ קיים מסלול $s \rightsquigarrow v \rightsquigarrow t$. הגרף קשיר אפוא, ו־$|E| \geq |V| - 1$. באיור 27.1 מובאת דוגמה לרשת זרימה.



איור 27.1 **(i)** רשת זרימה $G = (V, E)$ עבור בעיית ההובלה של חברת "דיסקית המזלי". בית החרושת בוונקובר הוא המקור s, והמחסן בווינייפג הוא הבור t. משלוחי הדיסקיות עוברים דרך ערי ביניים, אולם מהעיר u לעיר v ניתן לשלוח רק $c(u, v)$ תיבות ליום. ליד כל קשת רשום קיבולה. **(ii)** זרימה f ב־G שערכה $|f| = 19$. באיור מוצגות רק זרימות נטו חיוביות. אם $f(u, v) > 0$, מופיע ליד הקשת (u, v) היחס $f(u, v)/c(u, v)$. (הלוכסן מציין הפרדה בין הזרימה לבין הקיבול; הוא אינו מציין פעולת חילוק.) אם $f(u, v) \leq 0$, אזי ליד הקשת (u, v) רשום הקיבול בלבד.

עתה אנו ערוכים להגדרת זרימות בצורה פורמלית יותר. תהי $G = (V, E)$ רשת זרימה (עם פונקציית קיבול משתמעת c). יהי s מקור הרשת ויהי t הבור. **זרימה** (flow) ב־G היא פונקציה ממשית $f : V \times V \to \mathbf{R}$ המקיימת את שלוש התכונות הבאות:

אילוצי קיבול (capacity constraints): לכל $u, v \in V$, $f(u, v) \leq c(u, v)$.

סימטריה נגדית (skew symmetry): לכל $u, v \in V$, $f(u, v) = -f(v, u)$.

שימור הזרימה (flow conservation): לכל $u \in V - \{s, t\}$, אנו דורשים כי:

$$\sum_{v \in V} f(u, v) = 0$$

הגודל $f(u, v)$, שיכול להיות חיובי או שלילי, נקרא **הזרימה נטו** (net flow) מקדקוד u אל קדקוד v. **הערך** (value) של זרימה f מוגדר על־ידי:

$$|f| = \sum_{v \in V} f(s, v) \tag{27.1}$$

דהיינו, הזרימה נטו הכוללת היוצאת מן המקור. (כאן, הסימון $|\cdot|$ מציין ערך של זרימה ולא ערך מוחלט או עוצמה.) **בבעיית הזרימה המקסימלית** (maximum-flow problem) נתונה לנו רשת זרימה G עם מקור s ובור t, וברצוננו למצוא זרימה בעלת ערך מקסימלי מ־s אל t.

לפני שנראה דוגמה של בעיית הזרימה המקסימלית, הבה נחקור בקצרה את שלוש תכונות הזרימה. אילוצי הקיבול פירושם פשוט שהזרימה נטו מקדקוד אחד לאחר אסור לה שתעלה על הקיבול הנתון. סימטריה נגדית פירושה שהזרימה נטו מקדקוד u אל קדקוד v שווה בערכה והפוכה בסימנה לזרימה נטו בכיוון ההפוך. כך, הזרימה נטו מקדקוד לעצמו היא 0, שכן לכל $u \in V$ מתקיים $f(u, u) = -f(u, u)$, ומכאן נובע כי $f(u, u) = 0$. תכונת שימור הזרימה פירושה שהזרימה נטו הכוללת היוצאת מקדקוד שאינו המקור או הבור היא 0. על סמך הסימטריה הנגדית ניתן לנסח את תכונת שימור הזרימה גם באופן הבא:

$$\sum_{u \in V} f(u, v) = 0$$

עבור כל $v \in V - \{s, t\}$. דהיינו, הזרימה נטו הכוללת הנכנסת אל קדקוד היא 0.

עוד נשים לב כי לא תיתכן זרימה נטו בין u ל־v אם לא קיימת ביניהם קשת. אם $(u, v) \notin E$ וגם $(v, u) \notin E$, אזי $c(u, v) = c(v, u) = 0$. מכאן, על פי אילוץ הקיבול, $f(u, v) \leq 0$ וגם $f(v, u) \leq 0$. אולם מכיוון שעל פי הסימטריה הנגדית, $f(u, v) = -f(v, u)$, אזי $f(u, v) = f(v, u) = 0$. לכן, אם הזרימה נטו מקדקוד u אל קדקוד v שונה מאפס, הרי שמכך נובע כי $(u, v) \in E$ או $(v, u) \in E$ (או שניהם).

ההערה האחרונה שלנו הנוגעת לתכונותיהן של זרימות עוסקת בזרימות נטו חיוביות. **הזרימה נטו החיובית** (positive net flow) הנכנסת לקדקוד v מוגדרת על־ידי:

$$\sum_{\substack{u \in V \\ f(u, v) > 0}} f(u, v) \tag{27.2}$$

הזרימה נטו החיובית היוצאת מקדקוד מוגדרת באופן סימטרי. אחד הפירושים של תכונת שימור הזרימה הוא, שהזרימה נטו החיובית הנכנסת לקדקוד שאינו המקור או הבור חייבת להיות שווה לזרימה נטו החיובית היוצאת מקדקוד זה.

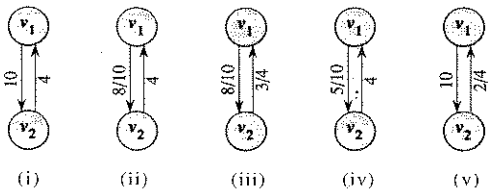
דוגמה לרשת זרימה

באמצעות רשת זרימה ניתן לבנות מודל לבעיית ההובלה המוצגת באיור 27.1. חברת "דיסקית המזל" היא בעלת מפעל בוונקובר (המקור s), המייצר דיסקיות הוקי, ומחסן בוויניפג (הבור t), שבו מאוחסנות הדיסקיות. חברת "דיסקית המזל" שוכרת מחברה אחרת מקום במשאיות, כדי להוביל את הדיסקיות מן המפעל אל המחסן. מאחר שהמשאיות נוסעות בנתיבים ידועים נוראש בין ערים, ומאחר שהן בעלות קיבול מוגבל, חברת "דיסקית המזל" יכולה להוביל לכל היותר $c(u, v)$ תיבות ליום בין כל זוג ערים, u ו־v, ברשת המוצגת באיור 27.1(i). לחברה אין שליטה על נתיבי הנסיעה ועל הקיבולים ולכן אינה יכולה לשנות את רשת הזרימה המוצגת באיור 27.1(i). מטרתה היא למצוא את המספר הגדול ביותר p של תיבות ליום שניתן להוביל, ואז לייצר דיסקיות בכמות זו, שכן אין טעם לייצר דיסקיות בכמות גדולה מזו שניתן להוביל למחסן.

הקצב שבו מובלות דיסקיות לאורך נתיב כלשהו הוא זרימה. הדיסקיות נפלטות מהמפעל בקצב של p תיבות ליום, ו־p תיבות חייבות להגיע למחסן בכל יום. החברה אינה מתעניינת בשאלה כמה זמן יארך מסעה של דיסקית נתונה מן המפעל אל המחסן; דרישתה היחידה היא ש־p תיבות ליום ייצאו מן המפעל ו־p תיבות ליום יגיעו למחסן. אילוצי הקיבול נתונים על־ידי ההגבלה שהזרימה $f(u, v)$ מעיר u לעיר v לא תעלה על $c(u, v)$ תיבות ליום. במצב יציב, הקצב שבו נכנסות הדיסקיות לעיר ביניים ברשת ההובלה חייב להיות שווה לקצב שבו הן יוצאות ממנה; אחרת, הדיסקיות יצטברו בעיר זו. הרשת מצייתת אפוא לחוק שימור הזרימה. לכן, זרימה מקסימלית ברשת קובעת את המספר המקסימלי p של תיבות ליום שניתן להוביל מן המפעל למחסן.

איור 27.1(ii) מראה זרימה אפשרית ברשת; הזרימה מיוצגת בדרך המתאימה באופן טבעי לייצוג משלוחים. עבור כל שני קדקודים u ו־v ברשת, הזרימה נטו $f(u, v)$ מייצגת משלוח של $f(u, v)$ תיבות ליום מ־u אל v. אם $f(u, v)$ שווה ל־0 או שלילי, אזי אין משלוח מ־u אל v. לפיכך, באיור 27.1(ii) מוצגות רק קשתות בעלות זרימה נטו חיובית; לידּ כל קשת כזאת רשומה הזרימה, לימינה לוכסן, ולימינו קיבול הקשת.

נוכל להיטיב להבין את הקשר בין זרימות נטו לבין משלוחים אם נתמקד בהובלות בין שני קדקודים. באיור 27.2(i) מוצג הותו־גרף המושרה על־ידי הקדקודים v_1 ו־v_2 מרשת הזרימה שבאיור 27.1. אם חברת דיסקיות המזל מובילה 8 תיבות ליום מ־v_1 ל־v_2, התוצאה מוצגת באיור 27.2(ii): הזרימה נטו מ־v_1 ל־v_2 היא 8 תיבות ליום. על פי הסימטריה הנגדית, אנו אומרים גם שהזרימה נטו בכיוון ההפוך, מ־v_2 ל־v_1, היא 8– תיבות ליום, למרות שאין כלל הובלת דיסקיות מ־v_2 ל־v_1. באופן כללי, הזרימה נטו מ־v_1 אל v_2 היא מספר התיבות ליום הנשלחות מ־v_1 אל v_2 פחות מספר התיבות ליום הנשלחות מ־v_2 אל v_1. המוסכמה שלנו באשר לייצוג זרימות נטו היא להציג זרימות נטו חיוביות בלבד, שכן אלה מצביעות על משלוחים בפועל. לכן, באיור מופיע רק 8, ללא ה־8– המקביל.



איור 27.2 קיזוזים. **(i)** הקדקודים v_1 ו־v_2 שעבורם $c(v_1, v_2) = 10$ ו־$c(v_2, v_1) = 4$. **(ii)** הדרך שבה אנו מציינים את הזרימה נטו כאשר 8 תיבות ליום נשלחות מ־v_1 אל v_2. **(iii)** משלוח נוסף של 3 תיבות ליום מתבצע מ־v_2 ל־v_1. **(iv)** על ידי קיזוז הזרימות בכיוונים המנוגדים, ניתן לייצג את המצב ב־(iii) באמצעות זרימה נטו חיובית בכיוון אחד בלבד. **(v)** 7 תיבות נוספות ליום נשלחות מ־v_2 אל v_1.

הבה נוסיף עתה עוד משלוח, והפעם משלוח של 3 תיבות ליום מ־v_2 אל v_1. ייצוג טבעי אחד של התוצאה מופיע באיור 27.2(iii). עתה המצב הוא שבין v_1 ו־v_2 מתבצעים משלוחים בשני הכיוונים. 8 תיבות ליום נשלחות מ־v_1 אל v_2, ו־3 תיבות ליום נשלחות מ־v_2 אל v_1. מהן הזרימות נטו בין שני הקדקודים? הזרימה נטו מ־v_1 אל v_2 היא $8 - 3 = 5$ תיבות ליום, והזרימה נטו מ־v_2 אל v_1 היא $3 - 8 = -5$ תיבות ליום.

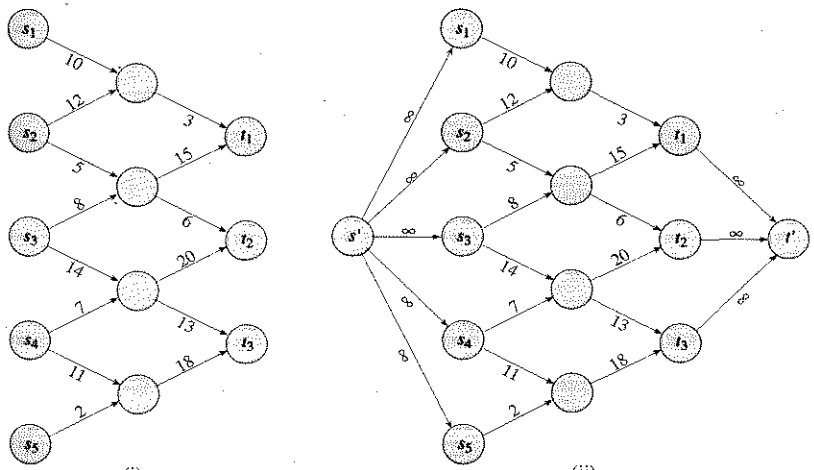
התוצאה של מצב זה שקולה למצב המתואר באיור 27.2(iv), שבו נשלחות 5 תיבות ליום מ־v_1 אל v_2 ואין שום משלוחים מ־v_2 אל v_1. למעשה, 3 התיבות ליום הנשלחות מ־v_2 אל v_1 **מקוזזות** על־ידי 3 מתוך 8 התיבות ליום הנשלחות מ־v_1 אל v_2. בשני המצבים, הזרימה נטו מ־v_1 אל v_2 היא 5 תיבות ליום, אבל ב־(iv), משלוחים בפועל מתבצעים בכיוון אחד בלבד.

באופן כללי, קיזוז מאפשר לנו לייצג משלוחים בין שתי ערים באמצעות זרימה נטו חיובית לאורך קשת אחת לכל היותר מבין שתי הקשתות בין הקדקודים המתאימים. אם הזרימה נטו מקדקוד אחד לאחר היא אפס או שלילית, אין צורך לבצע שום משלוח בכיוון זה. כלומר, כל מצב שבו דיסקיות נשלחות בין שתי ערים בשני הכיוונים ניתן להמיר באמצעות קיזוז למצב שקול שבו דיסקיות נשלחות בכיוון אחד בלבד: כיוונה של הזרימה נטו החיובית. המרה כזו אינה מפרה אילוצי קיבול, שכן אנו מצמצמים את המשלוחים בשני הכיוונים, ואילוצי שימור אינם מופרים, שכן הזרימה נטו בין שני הקדקודים אינה משתנה.

נמשיך בדוגמה שלנו ונראה מה קורה אם שולחים עוד 7 תיבות ליום מ־v_2 אל v_1. באיור 27.2(v) מוצגת התוצאה תוך שימוש במוסכמה של ייצוג זרימות נטו חיוביות בלבד. הזרימה נטו מ־v_1 אל v_2 הופכת להיות $2 - 7 = -5$, והזרימה נטו מ־v_2 אל v_1 הופכת להיות $2 - (-5) = 7$. מאחר שהזרימה נטו מ־v_2 אל v_1 היא חיובית, היא מייצגת משלוח של 2 תיבות ליום מ־v_2 אל v_1. הזרימה נטו מ־v_1 אל v_2 היא -2 תיבות ליום, ומאחר שזרימה נטו זו אינה חיובית, דיסקיות אינן מובלות כלל בכיוון זה. לחלופין, 5 מבין 7 התיבות הנוספות ליום מ־v_2 אל v_1 ניתן לראות כמקוזזות את משלוח 5 התיבות ליום מ־v_1 אל v_2, כך שהמשלוח בפועל מ־v_2 אל v_1 הוא 2 תיבות ליום.

רשתות עם מקורות ובורות מרובים

בעיית הזרימה המקסימלית עשויה לכלול רשת בעלת מספר מקורות ובורות ולא רק מקור אחד ובור אחד. לדוגמה, ייתכן שלחברת "דיסקית המזל" יש קבוצה של m מפעלים $\{s_1, s_2, \ldots, s_m\}$ וקבוצה של n מחסנים $\{t_1, t_2, \ldots, t_n\}$, כמתואר באיור 27.3(i). למרבה המזל, בעיה זו אינה קשה יותר מאשר הגרסה הפשוטה של בעיית הזרימה המקסימלית.



איור 27.3 המרת בעיה של זרימה מקסימלית ברשת מרובת מקורות ובורות לבעיה עם מקור יחיד ובור יחיד. **(i)** רשת זרימה עם חמישה מקורות $S = \{s_1, s_2, s_3, s_4, s_5\}$ ושלושה בורות $T = \{t_1, t_2, t_3\}$. **(ii)** רשת זרימה שקולה עם מקור יחיד ובור יחיד. אנו מוסיפים מקור־על s' וקשת בעלת קיבול אינסופי מ־s' לכל אחד מהמקורות המרובים. אנו מוסיפים גם בור־על t' וקשת בעלת קיבול אינסופי מכל אחד מן הבורות המרובים אל t'.

ניתן לעשות רדוקציה של בעיית הזרימה המקסימלית ברשת מרובת מקורות ובורות לבעיית הזרימה המקסימלית ברשת בעלת מקור יחיד ובור יחיד. איור 27.3(ii) מדגים כיצד ניתן להפוך את הרשת מ־(i) לרשת זרימה רגילה בעלת מקור יחיד ובור יחיד. אנו יוצרים **מקור־על** (supersource) s ומוסיפים קשת מכוונת (s, s_i) בעלת קיבול $c(s, s_i) = \infty$, עבור כל $i = 1, 2, \ldots, m$. אנו יוצרים גם **בור־על** (supersink) חדש t, ומוסיפים קשת מכוונת (t_j, t) בעלת קיבול $c(t_j, t) = \infty$, עבור כל $j = 1, 2, \ldots, n$. אינטואיטיבית, כל זרימה ברשת שב־(i) מתאימה לזרימה ברשת שב־(ii), ולהיפך. המקור היחיד s פשוט מספק זרימה גדולה ככל שנדרש למקורות המרובים s_i, ובאותו אופן, הבור היחיד t קולט זרימה גדולה ככל שנדרש עבור הבורות המרובים t_j. את ההוכחה הפורמלית כי שתי הבעיות הן שקולות השארנו כתרגיל לקורא (תרגיל 27.1-3).

עבודה עם זרימות

בהמשך נעסוק בפונקציות אחדות (כגון f) המקבלות כארגומנטים שני קדקודים ברשת זרימה. בפרק זה נשתמש ב**סימון הסיכום המובלע** (implicit summation notation), שבו כל אחד מהארגומנטים, או שניהם, יכולים להיות *קבוצה* של קדקודים, ופירושו שהערך המסומן הוא הסכום של כל הדרכים האפשריות שבהן ניתן להחליף את הארגומנטים באיברים שלהם. לדוגמה, אם X ו־Y הן קבוצות של קדקודים, אזי:

$$f(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x, y)$$

כדוגמה נוספת, את האילוץ של שימור הזרימה ניתן לבטא גם על־ידי התנאי כי $f(u, V) = 0$ עבור כל $u \in V - \{s, t\}$. מטעמי נוחות נשמיט בדרך כלל בסימון הסכום המובלע את הצומדיים לסימון קבוצה שהיו צריכים להופיע בו. לדוגמה, במשוואה $f(s, V) = f(s, V - s)$, הביטוי $V - s$ פירושו הקבוצה $V - \{s\}$.

סימון הקבוצות במובלע מפשט לעתים קרובות את המשוואות העוסקות בזרימות. הלמה שלהלן, שאת ההוכחה שלה השארנו כתרגיל 27.1-4, מציגה כמה מן ההזהויות השכיחות ביותר שמופיעים בהן זרימות וסימוני קבוצות מובלעים.

למה 27.1

תהי $G = (V, E)$ רשת זרימה, ותהי f זרימה ב־G. אזי, עבור $X \subseteq V$ מתקיים:

$$f(X, X) = 0$$

עבור $X, Y \subseteq V$ מתקיים:

$$f(X, Y) = -f(Y, X)$$

עבור $X, Y, Z \subseteq V$, כאשר $X \cap Y = \phi$, מתקיים:

$$f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$$

וכן,

$$f(Z, X \cup Y) = f(Z, X) + f(Z, Y)$$ ■

כדוגמה לעבודה עם סימון הסכום המובלע נוכיח שערכה של זרימה שווה לזרימה נטו הכוללת אל הבור; דהיינו:

$$|f| = f(V, t) \qquad (27.3)$$

אינטואיטיבית הדבר נכון, שכן על פי שימור הזרימה, כל הקדקודים חוץ מן המקור והבור הם בעלי זרימה נטו 0, ולכן הבור הוא הקדקוד היחיד שיכולה להיות לו זרימת נטו שונה מ־0 שתאזן את הזרימה נטו השונה מ־0 של המקור. ההוכחה הפורמלית שלנו היא כדלקמן:

$$\begin{aligned}
|f| &= f(s, V) && \text{(על פי ההגדרה)}\\
&= f(V, V) - f(V - s, V) && \text{(על פי למה 27.1)}\\
&= f(V, V - s) && \text{(על פי למה 27.1)}\\
&= f(V, t) + f(V, V - s - t) && \text{(על פי למה 27.1)}\\
&= f(V, t) && \text{(על פי שימור הזרימה)}
\end{aligned}$$

בהמשכו של הפרק נכליל תוצאה זו (למה 27.5).

תרגילים

27.1-1

בהינתן קדקודים u ו־v ברשת זרימה, כאשר $c(u, v) = 5$ ו־$c(v, u) = 8$, נניח כי 3 יחידות זרימה נשלחות מ־u אל v ו־4 יחידות זרימה נשלחות מ־v אל u. מהי הזרימה נטו מ־u אל v? צייר את המצב על פי דוגמת איור 27.2.

27.1-2

ודא שהזרימה f המוצגת באיור 27.1(ii) מקיימת כל אחת משלוש תכונות הזרימה.

27.1-3

הרחב את הגדרת הזרימה ותכונותיה כך שיחולו גם על רשת מרובת מקורות ובורות. הראה שלכל זרימה ברשת זרימה מרובת מקורות ובורות מתאימה זרימה בעלת ערך זהה ברשת עם מקור יחיד ובור יחיד, המתקבלת על־ידי הוספת מקור־על ובור־על. הוכח גם את הכיוון ההפוך.

27.1-4

הוכח את למה 27.1.

27.1-5

עבור רשת הזרימה $G = (V, E)$ והזרימה f המוצגות באיור 27.1(ii), מצא זוג תת־קבוצות $X, Y \subseteq V$ שעבורן $f(X, Y) = -f(V - X, Y)$. לאחר מכן, מצא זוג תת־קבוצות $X, Y \subseteq V$ שעבורן $f(X, Y) \neq -f(V - X, Y)$

27.1-6

בהינתן רשת זרימה $G = (V, E)$, יהיו f_1 ו־f_2 פונקציות מ־$V \times V$ ל־$\mathbf{R}$. **סכום הזרימות** $f_1 + f_2$ הוא הפונקציה מ־$V \times V$ ל־$\mathbf{R}$ המוגדרת על־ידי:

$$(f_1 + f_2)(u, v) = f_1(u, v) + f_2(u, v) \tag{27.4}$$

עבור כל $u, v \in V$. אם f_1 ו־f_2 הן זרימות ב־G, אילו משלוש תכונות הזרימה חייב סכום הזרימות $f_1 + f_2$ לקיים, ואילו הוא עלול להפר?

27.1-7

תהי f זרימה ברשת ויהי α מספר ממשי. **המכפלה הסקלרית של הזרימה** αf היא הפונקציה מ־$V \times V$ ל־$\mathbf{R}$ המוגדרת על־ידי:

$$(\alpha f)(u, v) = \alpha \cdot f(u, v)$$

הוכח שהזרימות ברשת זרימה יוצרות קבוצה קמורה, על־ידי כך שתראה כי אם f_1 ו־f_2 הן זרימות, אזי גם $\alpha f_1 + (1 - \alpha) f_2$ היא זרימה, עבור כל α בתחום $0 \leq \alpha \leq 1$.

27.1-8

נסח את בעיית הזרימה המקסימלית כבעיית תכנון־לינארי.

27.1-9

מודל רשת הזרימה שהוצג בסעיף זה תומך בזרימה של מוצר אחד; **רשת זרימה מרובת מוצרים** (multi-commodity flow network) תומכת בזרימת p מוצרים בין קבוצה של p **קדקודי מקור** $S = \{s_1, s_2, \ldots, s_p\}$ לבין קבוצה של p **קדקודי בור** $T = \{t_1, t_2, \ldots, t_p\}$. הזרימה נטו של המוצר ה־$i$ מ־u אל v מסומנת על־ידי $f_i(u, v)$. עבור המוצר ה־i, s_i הוא המקור היחיד ו־t_i הוא הבור היחיד. שימור הזרימה מתקיים בנפרד עבור כל מוצר: הזרימה נטו של כל מוצר מכל קדקוד היא אפס, אלא אם כן הקדקוד הוא המקור או הבור של מוצר זה. סכום הזרימות נטו של כל המוצרים מ־u אל v אסור שיעלה על $c(u, v)$, ובדרך זו מתקיימת האינטראקציה בין זרימות המוצרים השונים. **ערך** הזרימה של מוצר מסוים הוא הזרימה נטו היוצאת מן המקור של מוצר זה. **ערך הזרימה הכוללת** (total flow value) הוא סכום הערכים של כל p זרימות המוצרים. הוכח כי קיים אלגוריתם זמן־פולינומיאלי הפותר את הבעיה של מציאת הערך המקסימלי של הזרימה הכוללת ברשת זרימה מרובת מוצרים, על־ידי ניסוח הבעיה כבעיית תכנון־לינארי.

27.2 שיטת פורד־פולקרסון

בסעיף זה נציג את שיטת פורד־פולקרסון (Ford-Fulkerson) לפתרון בעיית הזרימה המקסימלית. אנו קוראים לה "שיטה" ולא "אלגוריתם" משום שהיא מקיפה מימושים אחדים בעלי זמני ריצה שונים. שיטת פורד־פולקרסון מבוססת על שלושה רעיונות חשובים, שחשיבותם חורגת מעבר לשיטה עצמה ויש להם נגיעה לבעיות ולאלגוריתמים רבים העוסקים בזרימות — רשתות שיוריות, מסלולי שיפור וחתכים. רעיונות אלה ממלאים תפקיד מכריע במשפט החשוב, משפט זרימה־מקסימלית־חתך־מינימלי (משפט 27.7), אשר מאפיין את ערכה של זרימה מקסימלית במונחי חתכים של רשת הזרימה. נסיים סעיף זה בהצגת מימוש מסוים אחד של שיטת פורד־פולקרסון וניתוח זמן ריצתו.

שיטת פורד־פולקרסון היא שיטה איטרטיבית. מתחילים עם $f(u, v) = 0$ עבור כל $u, v \in V$, כלומר, עם זרימה התחלתית שערכה 0. בכל איטרציה, מגדילים את ערך הזרימה על־ידי מציאת "מסלול שיפור", שעליו אפשר לחשוב כעל מסלול מן המקור s אל הבור t שלאורכו ניתן לדחוף עוד זרימה, ולאחר מכן שיפור הזרימה דרך מסלול זה. חוזרים על תהליך זה עד אשר לא ניתן עוד למצוא מסלולי שיפור. המשפט של זרימה־מקסימלית־חתך־מינימלי יראה שהזרימה המתקבלת עם סיומו של תהליך זה היא מקסימלית.

```
FORD-FULKERSON-METHOD(G, s, t)
1  initialize flow f to 0
2  while there exists an augmenting path p
3      do augment flow f along p
4  return f
```

(iii) הוכיחו שאם f ו־f' הן זרימות ב־G אז גם $f+f'$ היא זרימה ב־G.

איור 27.4 (i) זרימה f בגרף G עם מקור s ובור t. (ii) רשת השיורית G_f ביחס לזרימה f שבאיור (i). (iii) גרף G' ומינימום חתך (L', R') שלו. (iv) זרימה f' ב־G שערכה $|f'| = |f| + 4$.

ולכמות זו, לעומת זאת, מקבלים כי:

$$\begin{aligned}
\sum_{v \in V} (f + f')(u, v) &= \sum_{v \in V} (f(u, v) + f'(u, v)) \\
&= \sum_{v \in V} f(u, v) + \sum_{v \in V} f'(u, v) \\
&= 0 + 0 \\
&= 0
\end{aligned}$$

לכבוד שתכונה זו נשמרת לכל $u \in V - \{s, t\}$ מתקיים:

$$\sum_{v \in V} (f + f')(u, v) = \sum_{v \in V} (f(u, v) + f'(u, v))$$

לכבוד אילוצי הקיבול נשים לב כי עבור כל $u, v \in V$ מתקיים $f(u, v) + f'(u, v) \leq c_f(u, v)$, לפי משפט 27.5, מכאן:

$$\begin{aligned}
(f + f')(u, v) &= f(u, v) + f'(u, v) \\
&\leq f(u, v) + (c(u, v) - f(u, v)) \\
&= c(u, v)
\end{aligned}$$

לבסוף, עבור $u \in V - \{s, t\}$ מתקיים:

$$\begin{aligned}
(f + f')(u, V) &= f(u, V) + f'(u, V) \\
&= 0 + 0 \\
&= 0
\end{aligned}$$

מכאן ש־$f + f'$ היא זרימה ב־G שערכה $|f + f'| = |f| + |f'| = 4 + 4 = 8$, כפי שניתן לראות באיור 27.4(iv).

רשתות שיוריות

נתון גרף מכוון $G = (V, E)$ ובו קיבול $c(u, v)$ לכל קשת $(u, v) \in E$. נתונה גם זרימה f ב־G, ואנו מעוניינים לדעת האם ניתן לשלוח עוד יחידות זרימה מ־u אל v מבלי לחרוג מקיבולי הקשתות. לשם כך נגדיר גרף חדש, הנקרא **רשת שיורית** (residual network) של G ביחס לזרימה f, ומסומן $G_f = (V, E_f)$, שצמתיו הם צמתי G וקשתיו מתאימות לכמות השיורית שניתן לשלוח בהן. **הקיבול השיורי** (residual capacity) של (u, v), המסומן $c_f(u, v)$, הוא ההפרש בין הקיבול של (u, v) לבין כמות הזרימה שכבר עוברת בה. לדוגמה, אם $c(u, v) = 16$ ו־$f(u, v) = 11$, אז ניתן לשלוח עוד $c_f(u, v) = 5$ יחידות זרימה מ־u אל v. כאשר הזרימה עוברת בכיוון ההפוך, כלומר כאשר $f(u, v) > 0$, הקיבול השיורי $c_f(u, v)$ גדול מהקיבול $c(u, v)$. לדוגמה, אם $c(u, v) = 16$ ו־$f(u, v) = -4$, אז הקיבול השיורי $c_f(u, v)$ הוא 20. ניתן לפרש מצב זה באופן הבא: קיימות זרימה נכון של 4 יחידות מ־u אל v, עמה נוכל לחזור עוד 16 יחידות מ־u אל v בכדי להגיע למצב שאותה נוכל לקזז על־ידי דחיפת זרימה נכון של 4 יחידות מ־v אל u. מכאן נקבל כי $c_f(u, v) = 16 + 4 = 20$. ניתן לחזור על הקיבול על הקשת (v, u) אם כן באשר מתחילים עם זרימה נכון של $f(u, v) = -4$, ניתן לדחוף 20 יחידות זרימה מטפורת לצבור הקיבול.

הגדרה 27.1. בהינתן רשת זרימה $G = (V, E)$ וזרימה f, **הרשת השיורית** (residual network) של G המושרית על ידי f היא $G_f = (V, E_f)$ כאשר:

$$E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$$

כלומר, כמובטח לעיל, דרך כל קשת ברשת השיורית, דהיינו, דרך כל קשת ב־E_f, ניתן לשלוח כמות חיובית ממש. באיור 27.4(i) תוצגת ומופיעה רשת הזרימה G והזרימה f מאיור 27.1(ii) ובאיור 27.4(ii) מופיעה הרשת השיורית המתאימה G_f.

(i)

(ii)

(iii)

(iv)

יתברר בהמשך כי לרשת שיורית G_f יכולים להיות לכל היותר $2|E|$ קשתות, שכן $|E_f| \leq 2|E|$.

למה 27.2

תהי $G = (V, E)$ רשת זרימה עם מקור s ובור t, ותהי f זרימה ב־G. תהי G_f הרשת השיורית של G המושרית על־ידי f, ותהי f' זרימה ב־G_f. אז, סכום הזרימות $f + f'$ המוגדר על־ידי משוואה (27.4) הוא זרימה ב־G שערכה $|f + f'| = |f| + |f'|$.

הוכחה עלינו לוודא תחילה כי סכום הזרימות $f + f'$ הוא אכן זרימה חוקית, דהיינו, כי הוא מקיים את אילוצי הקיבול ואת שימור הזרימה. נתחיל תחילה באילוצי הקיבול. נשים לב כי לפי הגדרת E_f מתקיים $c_f(u, v) > 0$ לכל $(u, v) \in E_f$, ומכאן שמתקיים $f'(u, v) \leq c_f(u, v)$ לכל $u, v \in V$. מאחר ש־$E_f \subseteq E$ מתקיים $c_f(u, v) = c(u, v) - f(u, v)$ לכל $(u, v) \in E$. מכאן שאם $(u, v) \notin E$ אז $c(u, v) = 0$ ו־$f(u, v) = -f(v, u)$, ומכאן $f'(u, v) \leq -f(v, u)$. נשים לב כי אם $(u, v) \notin E$ אז $(v, u) \in E$, ומכאן שמתקיים $f(v, u) \geq 0$. לכן, עבור $(u, v) \notin E$ מתקיים $f'(u, v) \leq -f(v, u) \leq 0$. נסכם ונקבל כי עבור כל $u, v \in V$ מתקיים:

$$\begin{aligned}
(f + f')(u, v) &= f(u, v) + f'(u, v) \\
&\leq c(u, v) + (-f(v, u)) \\
&= -f(v, u) - f'(v, u) \\
&= -(f(v, u) + f'(v, u)) \\
&= -(f + f')(v, u)
\end{aligned}$$

$$(f + f')(u, v) = f(u, v) + f'(u, v) \leq c_f(u, v)$$

$$
\begin{aligned}
|f + f'| &= \sum_{v \in V} (f + f')(s, v) \\
&= \sum_{v \in V} (f(s, v) + f'(s, v)) \\
&= \sum_{v \in V} f(s, v) + \sum_{v \in V} f'(s, v) \\
&= |f| + |f'|
\end{aligned}
$$

■

מסלולי שיפור

בהינתן רשת זרימה $G = (V, E)$ וזרימה f, **מסלול שיפור** (augmenting path) p הוא מסלול פשוט מ־s אל t ברשת השיורית G_f. על פי הגדרתה של רשת שיורית, דרך כל קשת (u, v) על מסלול שיפור ניתן להעביר מ־u אל v זרימה נטו חיובית נוספת מסוימת בלי להפר את אילוץ הקיבול על קשת זו.

המסלול המוצלל באיור 27.4(ii) הוא מסלול שיפור. אם נתייחס לרשת השיורית G_f שבאיור כאל רשת זרימה, הרי שאנו יכולים להעביר עד 4 יחידות נוספות של זרימה נטו דרך כל אחת מן הקשתות על מסלול זה בלי להפר שום אילוץ קיבול, שכן הקיבול השיורי הקטן ביותר על מסלול זה הוא $c_f(v_2, v_3) = 4$. לכמות המקסימלית של זרימה נטו שניתן להעביר דרך קשתותיו של מסלול שיפור p קוראים **הקיבול השיורי** (residual capacity) של p, והוא נתון על־ידי:

$$c_f(p) = \min\{c_f(u, v) : p \text{ נמצאת על } (u, v)\}$$

הלמה הבאה, שאת ההוכחה שלה השארנו כתרגיל 27.2-7, מנסחת בצורה מדויקת יותר את הטיעון שלעיל.

למה 27.3

תהי $G = (V, E)$ רשת זרימה, תהי f זרימה ב־G, ויהי p מסלול שיפור ב־G_f. נגדיר פונקציה $f_p : V \times V \to \mathbf{R}$ על־ידי:

$$f_p(u, v) = \begin{cases} c_f(p) & \text{אם } (u, v) \text{ נמצאת על } p \\ -c_f(p) & \text{אם } (v, u) \text{ נמצאת על } p \\ 0 & \text{אחרת} \end{cases} \qquad (27.6)$$

אזי, f_p היא זרימה ב־G_f שערכה $|f_p| = c_f(p) > 0$.

■

המסקנה הבאה מראה שאם מוסיפים את f_p ל־f, מקבלים זרימה אחרת ב־G שערכה קרוב יותר למקסימום. באיור 27.4(iii) מוצגת תוצאת החיבור של f_p מאיור 27.4(ii) ל־f מאיור 27.4(i).

מסקנה 27.4

תהי $G = (V, E)$ רשת זרימה, תהי f זרימה ב־G, ויהי p מסלול שיפור ב־G_f. נגדיר את f_p כמו במשוואה (27.6). נגדיר פונקציה $f' : V \times V \to \mathbf{R}$ על־ידי $f' = f + f_p$. אזי, f' היא זרימה ב־G שערכה $|f'| = |f| + |f_p| > |f|$.

הוכחה נובעת ישירות מלמות 27.2 ו־27.3.

■

חתכים של רשתות זרימה

שיטת פורד־פולקרסון חוזרת ומשפרת את הזרימה לאורך מסלולי שיפור, עד למציאת זרימה מקסימלית. המשפט של זרימה־מקסימלית־חתך־מינימלי שנוכיח מיד, מראה שזרימה היא מקסימלית אם ורק אם הרשת השיורית שלה אינה מכילה שום מסלולי שיפור. אולם כדי להוכיח משפט זה עלינו לחקור תחילה את מושג החתך של רשת זרימה.

חתך (cut) (S, T) של רשת זרימה $G = (V, E)$ הוא חלוקה של V לשתי קבוצות S ו־$T = V - S$, כך ש־$s \in S$ ו־$t \in T$. (בדרך דומה הגדרנו בפרק 24 "חתך" עבור עצים פורשים מינימליים, אלא שכאן החתך הוא של גרף מכוון ולא של גרף בלתי־מכוון, ואנו עומדים על כך ש־$s \in S$ ו־$t \in T$.) אם f היא זרימה, אזי **הזרימה נטו** (net flow) דרך החתך מוגדרת על־ידי $f(S, T)$. **הקיבול** (capacity) של החתך (S, T) הוא $c(S, T)$. **חתך מינימלי** (minimum cut) של רשת הוא חתך שקיבולו מינימלי מבין כל החתכים של רשת הזרימה.

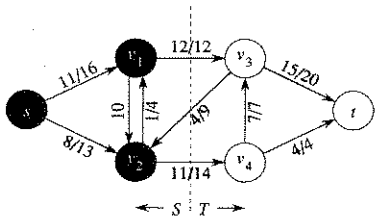
באיור 27.5 מוצג החתך $(\{s, v_1, v_2\}, \{v_3, v_4, t\})$ של רשת הזרימה מאיור 27.1(ii). הזרימה נטו דרך חתך זה היא:

$$
\begin{aligned}
f(v_1, v_3) + f(v_2, v_3) + f(v_2, v_4) &= 12 + (-4) + 11 \\
&= 19
\end{aligned}
$$

וקיבול החתך הוא:

$$
\begin{aligned}
c(v_1, v_3) + c(v_2, v_4) &= 12 + 14 \\
&= 26
\end{aligned}
$$

נשים לב כי הזרימה נטו דרך חתך עשויה לכלול זרימות נטו שליליות בין קדקודים, אולם קיבולו של חתך מורכב מערכים אי־שליליים בלבד.



איור 27.5 חתך (S, T) של רשת הזרימה מאיור 27.1(ii), כאשר $S = \{s, v_1, v_2\}$ ו־$T = \{v_3, v_4, t\}$. הקדקודים השחורים באיור הם קדקודי S והקדקודים הלבנים — קדקודי T. הזרימה נטו דרך (S, T) היא $f(S, T) = 19$, וקיבול החתך הוא $c(S, T) = 26$.

הלמה הבאה מראה שערכה של זרימה ברשת שווה לזרימה נטו דרך חתך כלשהו ברשת.

משפט 27.7 (זרימה-מקסימלית חתך-מינימלי)
אם f היא זרימה בשתי זרימה $G = (V, E)$ עם מקור s ובור t, אז התנאים הבאים שקולים זה לזה:

1. f היא זרימה מקסימלית ב-G.
2. רשת השיורית G_f אינה מכילה שום מסלול שיפור.
3. קיים חתך (S, T) ב-G שעבורו $|f| = c(S, T)$.

הוכחה (1) $\Rightarrow$ (2): נניח בדרך השלילה כי יש למקבץ G_f מסלול שיפור p. אזי, הזרימה f' שהוגדרה במשוואה (27.9) ב-G_f היא זרימה חוקית ב-G עם $|f'| = |f| + c_f(p) > |f|$, בסתירה לכך ש-$f$ היא זרימה מקסימלית.

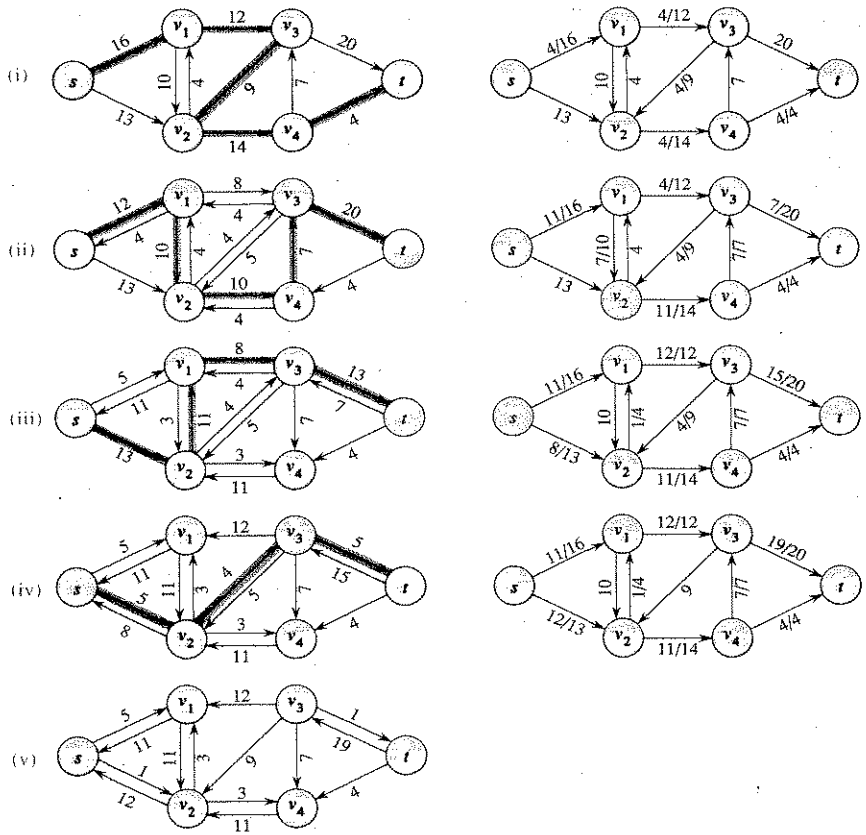
(2) $\Rightarrow$ (3): נניח כי G_f אינה מכילה שום מסלול שיפור, ונגדיר $S = \{v \in V : G_f$ ב-s אל v קיים מסלול$\}$ ו-$T = V - S$. (S, T) היא חתך של G שכן $s \in S$ ו-$t \notin S$ מאחר שאין מסלול שיפור ב-G_f. עבור כל זוג קודקודים u ו-v כך ש-$u \in S$ ו-$v \in T$, מתקיים $f(u, v) = c(u, v)$ שכן אחרת $(u, v) \in E_f$ ו-v היה שייך לקבוצה S. לכן, לפי למה 27.4, $|f| = f(S, T) = c(S, T)$.

(3) $\Rightarrow$ (1): על פי למה 27.6, עבור כל חתך (S, T) מתקיים $|f| = c(S, T)$. מאחר שאין זרימה f שערכה גדול מ-$c(S, T)$, נובע כי f היא זרימה מקסימלית.

■

אלגוריתם פורד-פולקרסון הבסיסי

אלגוריתם FORD-FULKERSON-METHOD איננו אלא הדרכה של השערה בדומה לשיטת הדגה שתוארה לעיל. איור 27.6 מציג את התאוותנות של כל איטרצי



איור 27.6 ביצוע אלגוריתם פורד־פולקרסון הבסיסי. **(i)**-**(iv)** איטרציות עוקבות של לולאת ה־**while**. בכל חלק באיור מוצגת משמאל הרשת השיורית G_f משורה 4, עם מסלול שיפור p מוצלל, בצד ימין מוצגת הזרימה החדשה f, המתקבלת מהוספת f_p ל־f. הרשת השיורית ב־(i) היא רשת הקלט G. **(v)** הרשת השיורית בעת הבדיקה באיטרציה האחרונה של לולאת ה־**while**. רשת זו אינה מכילה מסלולי שיפור, ולכן הזרימה f המוצגת ב־(iv) היא זרימה מקסימלית.

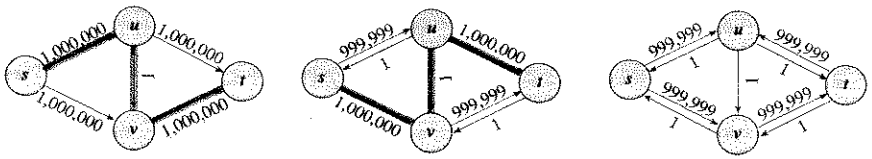
ניתוח של Ford-Fulkerson

זמן הריצה של Ford-Fulkerson תלוי באופן שבו נקבע מסלול השיפור בשורה 4. אם הוא נבחר בצורה גרועה, האלגוריתם עלול שלא לעצור כלל; ערך הזרימה אמנם יגדל עם השיפורים העוקבים, אבל הוא עלול שלא להתכנס לערך הזרימה המקסימלית. אך אם מסלול השיפור נבחר באמצעות חיפוש לרוחב (סעיף 23.2), האלגוריתם רץ בזמן פולינומיאלי. אולם בטרם נוכיח זאת, נראה כיצד מקבלים חסם פשוט עבור המקרה שבו מסלול השיפור נבחר באופן שרירותי וכל ערכי הקיבול הם מספרים שלמים.

ברוב המקרים שבהם מתעוררת בעיית הזרימה המקסימלית ביישומים מעשיים, ערכי הקיבול הם מספרים שלמים. אם ערכי הקיבול הם מספרים רציונליים, אפשר להפוך את כולם למספרים שלמים על־ידי כיול (כלומר, הכפלתם במספר גדול דיו). בהנחות אלה, מימוש ישיר של Ford-Fulkerson רץ בזמן $O(E\,|f^*|)$, כאשר f^* היא הזרימה המקסימלית שמוצא האלגוריתם. הניתוח נעשה כך: שורות 1-3 רצות בזמן $\Theta(E)$. לולאת ה־**while** בשורות 4-8 מבוצעת לכל היותר $|f^*|$ פעמים, שכן ערך הזרימה גדל ביחידה אחת לפחות בכל איטרציה.

את העבודה המתבצעת בתוך לולאת ה־**while** ניתן לייעל באמצעות ניהול יעיל של מבנה הנתונים המשמש למימוש הרשת $G = (V, E)$. הבה נניח שאנו מנהלים מבנה נתונים המייצג גרף מכוון $G' = (V, E')$, כאשר $E' = \{(u, v) : (u, v) \in E \text{ או } (v, u) \in E\}$. קשתות ברשת G הן גם קשתות ב־G', ולכן אין קושי לתחזק קיבולים וזרימות במבנה נתונים זה. בהינתן זרימה f ב־G, קבוצת הקשתות ברשת השיורית G_f מורכבת מכל הקשתות (u, v) ב־G' שעבורן $c(u, v) - f[u, v] \neq 0$. הזמן למציאת מסלול ברשת השיורית הוא אפוא $O(E') = O(E)$. אם משתמשים בחיפוש לעומק או בחיפוש לרוחב. לכן, כל איטרציה של לולאת ה־**while** מתבצעת בזמן $O(E)$, וזמן הריצה הכולל של Ford-Fulkerson הוא $O(E\,|f^*|)$.

כאשר ערכי הקיבול הם מספרים שלמים והערך האופטימלי של הזרימה קטן, זמן הריצה של אלגוריתם פורד־פולקרסון הוא טוב. איור 27.7(i) מדגים מה עלול לקרות ברשת זרימה פשוטה שבה $|f^*|$ גדול. ערכה של זרימה מקסימלית ברשת זו הוא 2,000,000: 1,000,000 יחידות זרימה עוברות דרך המסלול $s \to u \to t$ ו־1,000,000 יחידות נוספות עוברות דרך המסלול $s \to v \to t$. אם מסלול השיפור הראשון שמוצא Ford-Fulkerson הוא $s \to u \to v \to t$, כמוצג באיור 27.7(i), ערכה של הזרימה לאחר האיטרציה הראשונה הוא 1. הרשת השיורית המתקבלת מוצגת באיור 27.7(ii). אם האיטרציה השנייה מוצאת את מסלול השיפור $s \to v \to u \to t$, כמוצג באיור 27.7(ii), ערכה של הזרימה הוא אז 2. באיור 27.7(iii) מוצגת הרשת השיורית המתקבלת. ניתן להמשיך באופן זה, לבחור את מסלול השיפור $s \to u \to v \to t$ באיטרציות האי־זוגיות ואת מסלול השיפור $s \to v \to u \to t$ באיטרציות הזוגיות. מספר האיטרציות הכולל יגיע אז ל־2,000,000, כאשר בכל איטרציה, ערך הזרימה יגדל ביחידה אחת בלבד.



איור 27.7 **(i)** רשת זרימה שעבורה Ford-Fulkerson עלול לרוץ בזמן $\Theta(E\,|f^*|)$, כאשר f^* היא זרימה מקסימלית שערכה, במקרה זה, $|f^*| = 2{,}000{,}000$. ברשת מסומן מסלול שיפור שקיבולו השיורי 1. **(ii)** הרשת השיורית המתקבלת, ובה מסומן מסלול שיפור נוסף שקיבולו השיורי 1. **(iii)** הרשת השיורית המתקבלת.

ניתן לשפר את החסם על זמן הריצה של Ford-Fulkerson אם ממשים את חישוב מסלול השיפור p בשורה 4 באמצעות חיפוש לרוחב, דהיינו, אם מסלול השיפור הוא מסלול *קצר ביותר* מ־s אל t ברשת השיורית, כאשר אורכה (משקלה) של כל קשת הוא יחידה אחת. מימוש זה של שיטת Ford-Fulkerson נקרא **אלגוריתם אדמונדס־קארפ** (Edmonds-Karp algorithm). נוכיח עתה שזמן ריצתו של אלגוריתם אדמונדס־קארפ הוא $O(VE^2)$.

הניתוח מתבסס על מרחקים אל הקדקודים ברשת השיורית G_f. הלמה הבאה משתמשת בסימון $\delta_f(u, v)$ לציון מרחק המסלול הקצר ביותר מ־u אל v ב־G_f, כאשר אורכה של כל קשת הוא יחידה אחת.

למה 27.8

אם מריצים את אלגוריתם אדמונדס־קארפ על רשת זרימה $G = (V, E)$ עם מקור s ובור t, אזי עבור כל הקדקודים $v \in V - \{s, t\}$, מרחק המסלול הקצר ביותר $\delta_f(s, v)$ ברשת השיורית G_f גדל מונוטונית עם כל שיפור של הזרימה.

הוכחה נניח בדרך השלילה כי עבור קדקוד מסוים $v \in V - \{s, t\}$, קיים שיפור של הזרימה הגורם ל־$\delta_f(s, v)$ לקטון. תהי f הזרימה ממש לפני השיפור, ותהי f' הזרימה מיד אחריו. אזי,

$$\delta_{f'}(s, v) < \delta_f(s, v)$$

בלי הגבלת הכלליות ניתן להניח כי $\delta_{f'}(s, v) \le \delta_{f'}(s, u)$ לכל הקדקודים $u \in V - \{s, t\}$ שעבורם מתקיים $\delta_{f'}(s, u) < \delta_f(s, u)$. באופן שקול, ניתן להניח כי לכל הקדקודים $u \in V - \{s, t\}$, מתקיים:

$$\delta_{f'}(s, u) < \delta_{f'}(s, v) \Rightarrow \delta_f(s, u) \le \delta_{f'}(s, u) \qquad (27.7)$$

עתה ניקח מסלול קצר ביותר p' ב־$G_{f'}$ מן הצורה $s \rightsquigarrow u \to v$, ונתבונן בקדקוד u, הקודם של v על מסלול זה. על פי מסקנה 25.2, בהכרח מתקיים $\delta_{f'}(s, v) - 1 = \delta_{f'}(s, u)$, שכן (u, v) היא קשת על p', שהוא מסלול קצר ביותר מ־s אל v. לכן, על פי הנחתנו (27.7), מתקיים:

$$\delta_f(s, u) \le \delta_{f'}(s, u)$$

לאחר שקבענו כך את u ו־v, נוכל להתבונן בזרימה נטו f מ־u אל v לפני שיפור הזרימה ב־G_f. אם $f[u, v] < c(u, v)$, נקבל:

$$\begin{aligned} \delta_f(s, v) &\le \delta_f(s, u) + 1 && \text{(על פי למה 25.3)} \\ &\le \delta_{f'}(s, u) + 1 \\ &= \delta_{f'}(s, v) \end{aligned}$$

בסתירה להנחתנו ששיפור הזרימה מקטין את המרחק מ־s אל v.

אם כן, בהכרח מתקיים $f[u, v] = c(u, v)$, ופירוש הדבר כי $(u, v) \notin E_f$. כעת, מסלול השיפור p שנבחר ב־G_f כדי ליצור את $G_{f'}$ חייב להכיל את הקשת (u, v) *בכיוון מ־v אל u*, שכן $(u, v) \in E_{f'}$ (על פי ההנחה) ו־$(u, v) \notin E_f$ כפי שהראינו זה עתה. כלומר, שיפור הזרימה דרך המסלול p גורם לדחיפת זרימה *בחזרה* דרך (u, v), ו־v מופיע לפני u על p. מאחר ש־p הוא מסלול קצר ביותר מ־s אל t, גם תת־מסלולים שלו הם מסלולים קצרים ביותר (למה 25.1), ולכן אנו מקבלים $\delta_f(s, u) = \delta_f(s, v) + 1$. מכאן:

$$\begin{aligned} \delta_f(s, v) &= \delta_f(s, u) - 1 \\ &\le \delta_{f'}(s, u) - 1 \\ &= \delta_{f'}(s, v) - 2 \\ &< \delta_{f'}(s, v) \end{aligned}$$

בסתירה להנחתנו ההתחלתית. ■

המשפט הבא חוסם את מספר האיטרציות של אלגוריתם אדמונדס־קארפ.

משפט 27.9

אם מריצים את אלגוריתם אדמונדס־קארפ על רשת זרימה $G = (V, E)$ עם מקור s ובור t, אזי המספר הכולל של שיפורי זרימה שמבצע האלגוריתם הוא לכל היותר $O(VE)$.

הוכחה קשת (u, v) ברשת שיורית G_f נקראת קשת **קריטית** במסלול שיפור p אם הקיבול השיורי של p הוא הקיבול השיורי של (u, v), דהיינו, אם $c_f(p) = c_f(u, v)$. לאחר ששיפרנו זרימה דרך מסלול שיפור, כל קשת קריטית על המסלול נעלמת מן הרשת השיורית. יתר על כן, בכל מסלול שיפור חייבת להיות לפחות קשת קריטית אחת.

יהיו u ו־v קדקודים ב־V המחוברים על־ידי קשת ב־E. כמה פעמים יכולה (u, v) להיות קשת קריטית במהלך הרצה של אלגוריתם אדמונדס־קארפ? מאחר שמסלולי שיפור הם מסלולים קצרים ביותר, הרי שכאשר (u, v) היא קריטית בפעם הראשונה, מתקיים:

$$\delta_f(s, v) = \delta_f(s, u) + 1$$

מרגע ששופרה הזרימה, הקשת (u, v) נעלמת מן הרשת השיורית. היא אינה יכולה להופיע מאוחר יותר במסלול שיפור אחר אלא לאחר שהזרימה נטו מ־u אל v קָטְנָה, וזה קורה רק אם (v, u) מופיעה במסלול שיפור. אם f' היא הזרימה ב־G כאשר מתרחש מאורע זה, אזי:

$$\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1$$

מאחר שעל פי למה 27.8 מתקיים $\delta_f(s, v) \le \delta_{f'}(s, v)$, אנו מקבלים:

$$\begin{aligned} \delta_{f'}(s, u) &= \delta_{f'}(s, v) + 1 \\ &\ge \delta_f(s, v) + 1 \\ &= \delta_f(s, u) + 2 \end{aligned}$$

כתוצאה מכך, מהרגע שבו (u, v) הופכת להיות קריטית ועד הרגע שבו היא הופכת להיות קריטית בפעם הבאה, מרחקו של u מן המקור גדל ב־2 לפחות. במצב ההתחלתי, מרחקו של u מן המקור הוא לפחות 0, ועד שהוא הופך להיות בלתי ניתן להגעה מן המקור, אם בכלל, מרחקו של u מן המקור הוא לכל היותר $|V| - 2$. לפיכך, (u, v) יכולה להפוך להיות קריטית $O(V)$ פעמים לכל היותר. מכיוון שבגרף השיורי ישנם $O(E)$ זוגות קדקודים שביניהם יכולה לעבור קשת, המספר הכולל של קשתות קריטיות במהלך הרצה מלאה של אלגוריתם אדמונדס־קארפ הוא $O(VE)$. כל מסלול שיפור מכיל לפחות קשת קריטית אחת, ומכאן נובעת טענת המשפט. ■

מכיוון שכל איטרציה של FORD-FULKERSON ניתנת למימוש בזמן $O(E)$ כאשר מוצאים את מסלול השיפור באמצעות חיפוש לרוחב, זמן הריצה הכולל של אלגוריתם אדמונדס־קארפ הוא $O(VE^2)$. האלגוריתם שנביא בסעיף 27.4 מציג שיטה להשגת זמן ריצה של $O(V^2E)$, שהיא הבסיס לאלגוריתם של סעיף 27.5 שזמן ריצתו $O(V^3)$.

תרגילים

27.2-1

באיור 27.1(ii), מהי הזרימה דרך החתך $(\{s, v_2, v_4\}, \{v_1, v_3, t\})$? מהו קיבולו של חתך זה?

27.2-2

הראה את הרצתו של אלגוריתם אדמונדס־קארפ על רשת הזרימה מאיור 27.1(i).

27.2-3

בדוגמה המופיעה באיור 27.6, מהו החתך המינימלי המתאים לזרימה המקסימלית המוצגת? מבין מסלולי השיפור המופיעים בדוגמה, אילו שני מסלולים מקזזים זרימה שכבר הועברה קודם לכן?

27.2-4

הוכח שלכל זוג קדקודים u ו־v ולכל פונקציות קיבול וזרימה c ו־f, מתקיים $c_f(u, v) + c_f(v, u) = c(u, v) + c(v, u)$.

27.2-5

כזכור, הדרך שהצגנו בסעיף 27.1 להפיכת רשת זרימה מרובת מקורות ובורות לרשת עם מקור יחיד ובור יחיד כרוכה בהוספת קשתות בעלות קיבול אינסופי. הוכח שזרימה כלשהי ברשת המתקבלת היא בעלת ערך סופי אם הקשתות ברשת המקורית מרובת המקורות והבורות הן בעלות קיבולים סופיים.

27.2-6

נניח שכל מקור s_i ברשת מרובת מקורות ובורות מפיק בדיוק p_i יחידות זרימה, כך ש־$f(s_i, V) = p_i$. עוד נניח כי כל בור t_j צורך בדיוק q_j יחידות, כך ש־$f(V, t_j) = q_j$, כאשר $\sum_i p_i = \sum_j q_j$. הראה כיצד ניתן להמיר את הבעיה של מציאת זרימה f הכפופה לאילוצים נוספים אלה בבעיה של מציאת זרימה מקסימלית ברשת זרימה עם מקור יחיד ובור יחיד.

27.2-7

הוכח את למה 27.3.

27.2-8

הראה שתמיד ניתן למצוא זרימה מקסימלית ברשת $G = (V, E)$ באמצעות סדרה של $|E|$ מסלולי שיפור לכל היותר. (**רמז:** קבע את המסלולים *לאחר* מציאת הזרימה המקסימלית.)

27.2-9

הקשירות בקשתות (edge connectivity) של גרף בלתי־מכוון היא המספר המינימלי k של קשתות שיש לסלק כדי להפוך את הגרף לבלתי קשיר. לדוגמה, הקשירות בקשתות של עץ היא 1, הקשירות בקשתות של מעגל קדקודים היא 2. הראה כיצד ניתן לקבוע את הקשירות בקשתות של גרף בלתי־מכוון $G = (V, E)$ באמצעות הרצת אלגוריתם זרימה מקסימלית על $|V|$ רשתות זרימה לכל היותר, כאשר כל רשת היא בעלת $O(V)$ קדקודים ו־$O(E)$ קשתות.

27.2-10

נניח שהקשתות ברשת זרימה $G = (V, E)$ הן סימטריות, דהיינו, $(u, v) \in E$ אם ורק אם $(v, u) \in E$. הראה שאלגוריתם אדמונדס־קארפ עוצר לאחר $|V| \cdot |E|/4$ איטרציות לכל היותר. (**רמז:** עבור קשת (u, v) כלשהי, בדוק כיצד משתנים $\delta(s, u)$ ו־$\delta(v, t)$ בין הזמנים שבהם (u, v) היא קריטית.)

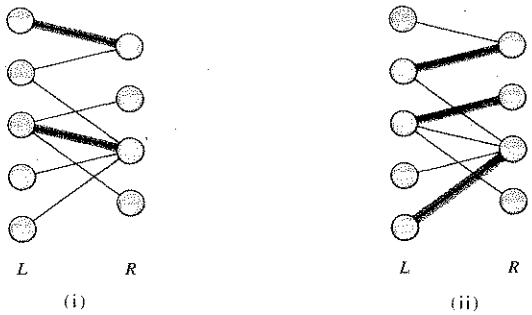
27.3 זיווג מקסימלי בגרף דו־צדדי

ישנן בעיות קומבינטוריות שניתן לנסחן בנקל כבעיות זרימה מקסימלית. בעיית הזרימה המקסימלית ברשת מרובת מקורות ובורות שהוצגה בסעיף 27.1 היא דוגמה אחת כזאת. קיימות בעיות קומבינטוריות אחרות אשר במבט ראשון נראה כאילו אין קשר בינן לבין רשתות זרימה, אולם למעשה ניתן לעשות רדוקציה שלהן לבעיות זרימה מקסימלית. בסעיף זה נציג בעיה אחת

כזאת: מציאת זיווג מקסימלי בגרף דו־צדדי (ראה סעיף 5.4). בפתרון בעיה זו ננצל את תכונת הפתרון בשלמים שמספקת שיטת פורד־פולקרסון. כמו כן נראה שבאמצעות שיטת פורד־פולקרסון ניתן לפתור את בעיית הזיווג המקסימלי בגרף דו־צדדי $G = (V, E)$ בזמן $O(VE)$.

בעיית הזיווג המקסימלי בגרף דו־צדדי

בהינתן גרף בלתי־מכוון $G = (V, E)$, **זיווג** (matching) הוא תת־קבוצה של קשתות $M \subseteq E$ כך שעבור כל קדקוד $v \in V$, קיימת ב־M לכל היותר קשת אחת הקשורה ל־v. קדקוד $v \in V$ נקרא **מזווג** (matched) על־ידי זיווג M אם קיימת ב־M קשת הקשורה ל־v; אחרת, v הוא **בלתי־מזווג** (unmatched). **זיווג מקסימלי** (maximum matching) הוא זיווג בעל עוצמה מקסימלית, דהיינו, זיווג M המקיים $|M'| \geq |M|$ עבור כל זיווג אחר M'. בסעיף זה נתמקד במציאת זיווגים מקסימליים בגרפים דו־צדדיים. אנו מניחים כי קבוצת הקדקודים ניתנת לחלוקה לשתי קבוצות, $V = L \cup R$, כאשר L ו־R זרות וכל הקשתות ב־E עוברות בין L ל־R. אנו מניחים עוד כי אין בגרף קדקודים מבודדים. איור 27.8 מדגים את רעיון הזיווג.



איור 27.8 גרף דו־צדדי $G = (V, R)$ עם חלוקת קדקודים $V = L \cup R$. **(i)** זיווג שעוצמתו 2. **(ii)** זיווג מקסימלי שעוצמתו 3.

מציאת זיווג מקסימלי בגרף דו־צדדי היא בעיה שיש לה יישומים מעשיים רבים. לדוגמה, ניתן לחשוב על זיווג בין קבוצה L של מכונות וקבוצה R של משימות שיש לבצע בו־זמנית. נוכחותה של קשת (u, v) ב־E פירושה שמכונה מסוימת $u \in L$ מסוגלת לבצע משימה מסוימת $v \in R$. זיווג מקסימלי מספק עבודה למספר גדול ככל האפשר של מכונות.

מציאת זיווג מקסימלי בגרף דו־צדדי

באמצעות שיטת פורד־פולקרסון ניתן למצוא זיווג מקסימלי בגרף דו־צדדי בלתי־מכוון $G = (V, E)$ בזמן פולינומיאלי ב־$|V|$ וב־$|E|$. הרעיון הוא לבנות רשת זרימה אשר הזרימות בה מתאימות לזיווגים, כמתואר באיור 27.9. נגדיר את **רשת הזרימה המתאימה** $G' = (V', E')$ עבור הגרף הדו־צדדי G באופן הבא: יהיו המקור s והבור t קדקודים חדשים שאינם שייכים ל־V, ותהי $V' = V \cup \{s, t\}$; אם חלוקת הקדקודים של G היא $V = L \cup R$, אזי קבוצת הקשתות המכוונות ב־G' היא:

$$\begin{aligned} E' = {} & \{(s, u) : u \in L\} \\ & \cup \{(u, v) : (u, v) \in E \text{ ו־} v \in R, u \in L\} \\ & \cup \{(v, t) : v \in R\} \end{aligned}$$

להשלמת הבנייה, אנו מייחסים לכל קשת ב־E' קיבול של יחידה אחת.

$O(VE)$ ומבוצעת בזמן ריצה לינארי במספר הקשתות של G. ■

משפט 27.11 (משפט המרזן השלם)

אם $G = (V, E)$ הוא רשת זרימה שבה הקיבולים של כל הקשתות הם מספרים שלמים, אז קיימת זרימה שלמה f שהיא מקסימלית.

הוכחה ההוכחה היא באינדוקציה על מספר האיטרציות של שיטת פורד-פלקרסון עבור G.

■

תרגיל 27.3-2

עדכן את האלגוריתם שבסעיף 27.1 כך שיחשב את הזרימה המקסימלית בגרף דו-צדדי G.

משקנה 27.12

בכל גרף דו-צדדי G שבו $|L| = |R|$ ובו לכל קודקוד יש דרגה d, קיים שידוך מושלם.

הוכחה נוכיח שקיים שידוך מושלם ב-G. ברור ש-$|L| \geq |R|$ כי אחרת אין שידוך מושלם. נניח בשלילה ש-$|M| < |L|$ עבור שידוך מקסימלי M. נגדיר רשת זרימה $G' = (V', E')$ כאשר $V' = V \cup \{s, t\}$ ו-$E' = \{(s, u) : u \in L\} \cup \{(u, v) : u \in L, v \in R, (u, v) \in M\} \cup \{(v, t) : v \in R\}$. כל קיבול בגרף G' הוא 1. ברור שקיים שידוך מושלם אם ורק אם קיים $|L|$ מסלולים זרים בקשתות בין s ל-t. מכיוון ש-M מקסימלי, קיים מספר שלם f שהוא זרימה מקסימלית ב-G' ו-$|f| = |M|$. לכל $u \in L$ מתקיים $f(s, u) = 1$, ולכל $v \in R$ מתקיים $f(v, t) = 1$. לכן, $|f| = |L| - |M|$. מכיוון ש-$|L| - |M| > 0$, נובע ש-$f(u, v) = 1$ עבור $u \in L$ ו-$v \in R$ כלשהם. לכן, M אינה מקסימלית, בסתירה להנחה. ■

משפט 27.13 (משפט המרזן)

אם $G = (V, E)$ הוא רשת זרימה שבה הקיבולים של כל הקשתות הם מספרים שלמים, אז קיימת זרימה מקסימלית f שהיא שלמה.

הוכחה מכיוון שכל קיבול הוא מספר שלם, השיפור הראשון שנבחר יהיה בעל ערך שלם חיובי. לכן, כל f_i שמתקבלת במהלך שיטת פורד-פלקרסון היא שלמה. מכיוון שהערך של הזרימה מקסימלי כאשר השיטה מסתיימת, המשפט נובע מיד מהמשפט שהוכחנו קודם לכן. ■

27.3 יישום מקסימלי בגרף דו-צדדי

בעיית השידוך המקסימלי בגרף לא מכוון $G = (V, E)$ היא למצוא קבוצת קשתות $M \subseteq E$ כך שלכל $v \in V$ מתקיים לכל היותר קשת אחת ב-M שפגועה ב-v, ו-$|M|$ מקסימלי. אומרים שקשת (u, v) **פוגעת** ב-u וב-v, ושקודקוד v **מושלם** ביחס ל-M אם קשת ב-M פוגעת ב-v. לכן, בבעיית השידוך המקסימלי אנו מחפשים קבוצה מקסימלית של קשתות שבה כל קודקוד מושלם לכל היותר.

בסעיף זה נראה כיצד ניתן לפתור בעיה זו בזמן $O(VE)$ בגרפים דו-צדדיים לא מכוונים. נניח שהגרף הנתון הוא $G = (V, E)$ ושקבוצת הקודקודים V היא איחוד של L ו-R, כאשר הקבוצות זרות. נניח גם שכל קשת ב-E פוגעת בקודקוד אחד ב-L ובקודקוד אחד ב-R. נראה כיצד ניתן לחשב את השידוך המקסימלי בזמן לינארי ב-$|V|$ ו-$|E|$.

איור 27.9 רשת הזרימה המתאימה לגרף דו-צדדי. (i) גרף דו-צדדי $G = (V, E)$ עם קבוצת הקודקודים L בצד שמאל וקבוצת הקודקודים R בצד ימין. קבוצת הקשתות המקסימלית המוצגת היא $M = \{(u_1, v_2), (u_2, v_1), (u_5, v_3)\}$. (ii) רשת הזרימה המתאימה G' עם זרימה בעלת ערך שלם המתאימה לשידוך M. הקשתות בקבוצה M מסומנות בקווים כהים, וכל קשת שאינה ב-M נושאת זרימה של 1. הקשתות (s, u_3) ו-(s, u_4) נושאות זרימה 0, אך אנו מקבלים שהן לא חלק מהמסלולים המוגדרים על ידי הזרימה.

משפט 27.10 (משפט הערכים השלמים)

אם $f(u, v)$ שלם עבור כל (u, v) בשיטת פורד-פלקרסון, אז $|f|$ שלם.

■

תרגיל 27.3-1

הוכח שמספר הקשתות ב-M שמקיימות $f(u, v) = 1$ הוא $|f|$.

תרגיל 27.3-2

הוכח שלכל $u \in L$ מתקיים $f(s, u) = 1$ או $f(s, u) = 0$, ושלכל $v \in R$ מתקיים $f(v, t) = 1$ או $f(v, t) = 0$.

תרגיל 27.3-3

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם $(u, v) \in M$.

תרגיל 27.3-4

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) \geq 0$. לפיכך, מכל קודקוד $u \in L$ יוצאת לכל היותר קשת אחת שנושאת זרימה חיובית, ולכל קודקוד $v \in R$ נכנסת לכל היותר קשת אחת שנושאת זרימה חיובית.

תרגיל 27.3-5

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 0$ או $f(u, v) = -1$.

תרגיל 27.3-6

הוכח שלכל $u \in L$ ו-$v \in R$, $f(s, u) + f(u, v) + f(v, t) \leq 1$.

תרגיל 27.3-7

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

הוכחה תחילה נראה שזיווג M ב-G מתאים לזרימה בעלת ערכים שלמים ב-G'. נגדיר f באופן הבא: אם $(u, v) \in M$, אז $f(u, v) = f(v, t) = f(s, u) = 1$, ו-$f(u, v) = -1$. עבור כל קשתות האחרות $(u, v) \in E'$, נגדיר $f(u, v) = 0$.

אינטואיטיבית, כל קשת $(u, v) \in M$ מתאימה למסלול זרימה אחת $s \to u \to v \to t$ ב-G'. קשתות המסלולים הזרים בקודקודים היחידים $u \in L$ ו-$v \in R$ שמקיימים $f(u, v) = 1$. כל קשת אחרת (u, v) שאינה ב-M מקיימת $f(u, v) = 0$, מכיוון שהיא אינה מתאימה לאף אחד מן המסלולים הזרים בקודקודים. לכן, f היא זרימה בעלת ערכים שלמים ב-G', ואת קיום ב-G יוצג M שמוצמנו $|f| = |M|$.

להוכחת הכיוון ההפוך, תהי f זרימה בעלת ערכים שלמים ב-G', ויהי:

$$M = \{(u, v) : u \in L, v \in R, f(u, v) > 0\}$$

למה 27.5 ערך הזרימה הוא $|M| = |f|$.

הוכחה נוכיח את הלמה באינדוקציה על מספר האיטרציות של שיטת פורד-פלקרסון. בתחילה, $f(u, v) = 0$ לכל (u, v), ולכן הלמה נכונה. נניח שהלמה נכונה עבור הזרימה f שהתקבלה לאחר איטרציות רבות, ונראה שהיא נכונה עבור איטרציה נוספת. נניח שהנתיב p שנוסף בשלב זה הוא $p = \langle s, u, v, t \rangle$, כאשר $(u, v) \in E_f$ ו-$u \in L$ ו-$v \in R$. לפי הגדרת E_f, $(u, v) \in E$ או $(v, u) \in E$. אם $(u, v) \in E$, אז $f(u, v) < c(u, v)$. אם $(v, u) \in E$, אז $f(v, u) > 0$. בכל מקרה, $|M|$ גדל בדיוק ב-1, מכיוון שהנתיב p מוסיף מסלול זר חדש ל-M. מכיוון שכל שאר המסלולים ב-M נשארים ללא שינוי, נובע שהלמה נכונה עבור $|M|$ החדש.

■

תרגיל 27.3-8

הוכח שלכל $u \in L$ ו-$v \in R$, $f(s, u) \leq 1$ ו-$f(u, v) \leq 1$. מכך, מלמה 27.1, מסתבר שלכל $u \in L$ ו-$v \in R$, $f(s, u) = 1$ אם ורק אם $(u, v) \in M$, ו-$f(u, v) = 0$ אחרת.

נוכיח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם $(u, v) \in M$, ו-$f(u, v) = 0$ אחרת. אם $(u, v) \in M$, אז (u, v) מתאימה לנתיב $s \to u \to v \to t$. אם $(u, v) \notin M$, אז אין נתיב כזה, מכיוון שהקשת היחידה היוצאת מ-u היא (u, v), והקשת היחידה הנכנסת ל-v היא (u, v). לכן, $f(u, v) = 0$.

תרגיל 27.3-9

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-10

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-11

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-12

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-13

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-14

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-15

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-16

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-17

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-18

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-19

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-20

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-21

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-22

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-23

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-24

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-25

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-26

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-27

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-28

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-29

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-30

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-31

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-32

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-33

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-34

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-35

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-36

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-37

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-38

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-39

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-40

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-41

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-42

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-43

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-44

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-45

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-46

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-47

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-48

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-49

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-50

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-51

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-52

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-53

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-54

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-55

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-56

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-57

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-58

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-59

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-60

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-61

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-62

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-63

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-64

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-65

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-66

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-67

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-68

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-69

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-70

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-71

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-72

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-73

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-74

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-75

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-76

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-77

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-78

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-79

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-80

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-81

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-82

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-83

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-84

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-85

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-86

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-87

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-88

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-89

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-90

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-91

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-92

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-93

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-94

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-95

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-96

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-97

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-98

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-99

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

תרגיל 27.3-100

הוכח שלכל $u \in L$ ו-$v \in R$, $f(u, v) = 1$ אם ורק אם (u, v) מתאימה לזרימה בעלת ערכים שלמים ב-G'.

אפן פעולתם של אלגוריתמי דחיפת־קדם־זרימה תהיה השוואתי בלבד, ולא נניח שום דבר לגבי
מדד־הפלקסיון במקום לבחון את רשת השיורית $G = (V, E)$ כלפי כיוי לפעמים נוח יותר בהשוואה לשיטת

27.4 * אלגוריתמי דחיפת־קדם־זרימה

בפרק זה נציג את הגישה של "דחיפת־קדם־זרימה" (preflow-push) לחישוב זרימת מקסימלית
האלגוריתמים המתוארים ביתר המובנים היום למציאת זרימות מקסימליות הם אלגוריתמי
דחיפת־קדם־זרימה, ובאמצעותם ניתן לפתור בעיות זרימה אחרות, כגון בעיית הזרימה
בעלות העלות המינימלית. "ההכללי" של גולדברג (Goldberg) את האלגוריתם
הטוב ביותר מבחינה אסימפטוטית עד כה. נציג אחד מאלגוריתמים אלה, שזמן ריצתו
$O(V^3)$. נשפר את האלגוריתם הכללי ונקבל
אלגוריתם דחיפת־קדם־זרימה אחר שזמן ריצתו $O(V^2E)$ — שיפור לעומת הזמן
$O(V^2E^2)$ של אלגוריתם אדמונדס־קארפ. בסעיף 27.5 נשפר עוד יותר את זמן הריצה
לכל מקרה שבו $E = \Omega(V^2)$, ונראה אלגוריתם שזמן ריצתו $O(VE \lg(V^2/E))$ — שיפור
קל לראות שזמן זה הוא $O(V^3)$ לכל מקרה שבו $E = O(V^2)$, ולכל מקרה שבו $E = \Omega(V^2)$
הוא $O(VE)$, שהוא שיפור לעומת $O(V^2E^2)$ של אלגוריתם אדמונדס־קארפ.

בשונה מאלגוריתמי שיפור־השביל, שבהם כל פעולה משנה את הזרימה בגרף שלם, אלגוריתמי
דחיפת־קדם־זרימה מתבססים על פעולות מקומיות יותר, שכל אחת מהן משנה את הזרימה
רק בקשת אחת או בקודקוד אחד. בנוסף, כל אחד מהאלגוריתמים שנציג בפרק זה מתבסס
על הרעיון של קודקודים שהזרימה עודפת בהם, כלומר קודקודים שכמות הזרימה הנכנסת
בהם עולה על הכמות היוצאת ממנה. נתחיל בפרק זה בהצגת שיטה כללית לדחיפת זרימה,
ולאחר מכן נתאר אלגוריתם יעיל לשיטה זו. לבסוף נראה שימוש בשיטה זו לפתרון בעיית
הזרימה בעלות המינימלית.

זרימה עודפת

הפעולה הבסיסית של אלגוריתמי דחיפת־קדם־זרימה היא דחיפת זרימה עודפת מקודקוד
אל שכן שלו. הגדרה זו מתבססת על הגדרה הבאה.

הגדרה. תהי $G = (V, E)$ רשת זרימה עם קיבולת c ותהי s מקור כלשהו של G. נגדיר
זרימה עודפת $f(V, s) \to \mathbf{R}$ בקודקוד $u \in V - \{s\}$ ביחס לזרימה f על ידי
$e(u) = f(V, u)$ (27.8)
כך שקודקוד u נתון נקרא **גולש** (overflowing) אם $e(u) > 0$.

נגדיר **קיבול שיורי** $c_f(u, v)$ של קשת $(u, v) \in V \times V$ ביחס לזרימה f כמו קודם,
כלומר $c_f(u, v) = c(u, v) - f(u, v)$. נגדיר **רשת שיורית** $G_f = (V, E_f)$ כמו קודם,
כלומר $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$. כל קשת ברשת שיורית היא
לפחות בעלת קיבול 1. לכל קשת (u, v) שיש לה קיבול חיובי ברשת המקורית G, גם
(v, u) שייכת ל־E_f, כך שגם (v, u) יכולה להיות קשת ברשת שיורית, אפילו אם
$(v, u) \notin E$. כפי שנראה בהמשך, אלגוריתמי דחיפת־קדם־זרימה מתבססים על כך
שקודקוד u נתון יכול לדחוף זרימה עודפת לקודקוד v אם (u, v) שייכת לרשת השיורית.

תרגילים

27.3-1
תרגם את אלגוריתם פורד־פולקרסון על רשת הזרימה מאיור 27.9(iii) והראה את הרשת השיורית
לאחר כל איטרציה של הלולאה while. קשקש את הקודקודים ב־L במטרה מ־1 עד 5, ואת
הקודקודים ב־R מלמעלה למטה מ־6 עד 9. בכל איטרציה, בחר את מסלול השיפור המסן ביותר
לקשקודקודים.

27.3-2
הוכח את משפט 27.11.

27.3-3
יהי $G = (V, E)$ גרף דו־צדדי עם חלוקת קודקודים $V = L \cup R$, ותהי G' רשת הזרימה
המתאימה לו. מצא חסם עליון טוב על אורכו של כל מסלול שיפור שמצא ב־G אלגוריתם
Ford-Fulkerson במהלך הריצתו.

27.3-4 *
זיווג מושלם (perfect matching) הוא זיווג שבו כל הקודקודים ממוינים. יהי $G = (V, E)$ גרף
דו־צדדי בעל חלוקת קודקודים $V = L \cup R$, כאשר $|L| = |R|$. עבור כל $X \subseteq V$, נגדיר את **הסביבה** (neighborhood) של X על־ידי
$N(X) = \{y \in V : (x, y) \in E \text{ עבור } x \in X \text{ כלשהו}\}$
כלומר, $N(X)$ היא קבוצת הקודקודים המתמזגים לקודקודים השייכים ל־X. הוכח את **משפט חול** (Hall's theorem): קיים ב־G זיווג מושלם אם ורק אם $|A| \le |N(A)|$ עבור כל תת־קבוצה $A \subseteq L$.

27.3-5 *
גרף דו־צדדי $G = (V, E)$ עם חלוקת קודקודים $V = L \cup R$ נקרא **d־רגולרי** (d-regular) אם
דרגתו של כל קודקוד $v \in V$ היא d בדיוק. כל גרף דו־צדדי d־רגולרי מקיים $|L| = |R|$. הוכח
שבכל גרף דו־צדדי d־רגולרי קיים זיווג בעל עוצמה $|L|$, ואשר מצאת הטיעון שקיבלת של חתך
מינימלי ברשת הזרימה המתאימה הוא $|L|$.

בסופו של דבר, יתקבל מצב שבו הגיעה לבור כל הזרימה שיכולה להגיע אליו. שום זרימה נוספת אינה יכולה להגיע מכיוון שהצינורות מציתים לאילוצי הקיבול; כמות הזרימה העוברת דרך חתך עדיין מוגבלת על־ידי קיבולו של החתך. בשלב זה, כדי להפוך את הקדם־זרימה לזרימה "חוקית", האלגוריתם שולח את העודף שנאסף במאגרים של קדקודים גולשים בחזרה אל המקור על־ידי כך שהוא ממשיך ומגביה קדקודים מעל לגובהו הקבוע $|V|$ של המקור. (משלוח העודף בחזרה אל המקור מבוצע למעשה על־ידי ביטול הזרימות הגורמות לעודף.) כפי שנראה, ברגע שרוקנו כל המאגרים, הקדם־זרימה אינה רק זרימה "חוקית", אלא היא גם זרימה מקסימלית.

הפעולות הבסיסיות

מן הדיון שלעיל עולה שאלגוריתם דחיפת־הקדם־זרימה מבצע שתי פעולות בסיסיות: דחיפת זרימה עודפת מקדקוד לאחד משכניו והרמה של קדקוד. ישימותן של פעולות אלה תלויה בגובהי הקדקודים, שאותם נגדיר עתה במדויק.

תהי $G = (V, E)$ רשת זרימה עם מקור s ובור t, ותהי f קדם־זרימה ב־G. פונקציה $h : V \to \mathbf{N}$ **היא פונקציית גובה** (height function) אם $h(s) = |V|$, $h(t) = 0$, ועבור כל קשת שיורית $(u, v) \in E_f$ מתקיים:

$$h(u) \leq h(v) + 1$$

מן ההגדרה מתקבלת ישירות הלמה הבאה.

למה 27.13

תהי $G = (V, E)$ רשת זרימה, תהי f קדם־זרימה ב־G, ותהי h פונקציית גובה על V. עבור כל שני קדקודים $u, v \in V$, אם $h(u) > h(v) + 1$, אזי (u, v) אינה קשת בגרף השיורי. ■

הפעולה הבסיסית $\textsc{Push}(u, v)$ ניתנת לביצוע אם u הוא קדקוד גולש, $c_f(u, v) > 0$ ו־$h(u) = h(v) + 1$. הפסאודו־קוד שלהלן מעדכן את הקדם־זרימה f ברשת משתמעת $G = (V, E)$. הפסאודו־קוד מניח שהקיבולים נתונים על־ידי פונקציה c קבועה בזמן ושגם את הקיבולים השיוריים ניתן לחשב בזמן קבוע, בהינתן c ו־f. הזרימה העודפת המאוחסנת בקדקוד u נשמרת ב־$e[u]$, והגובה של u נשמר ב־$h[u]$. הביטוי $d_f(u, v)$ הוא משתנה זמני המכיל את כמות הזרימה שניתן לדחוף מ־u אל v.

```
PUSH(u, v)
1  ▷ Applies when: u is overflowing, c_f[u, v] > 0, and h[u] = h[v] + 1.
2  ▷ Action: Push d_f(u, v) = min(e[u], c_f(u, v)) units of flow
                  from u to v.
3  d_f(u, v) ← min(e[u], c_f(u, v))
4  f[u, v] ← f[u, v] + d_f(u, v)
5  f[v, u] ← − f[u, v]
6  e[u] ← e[u] − d_f(u, v)
7  e[v] ← e[v] + d_f(u, v)
```

השגרה PUSH פועלת באופן הבא: ההנחה היא כי הקדקוד u הוא בעל זרימה עודפת חיובית $e[u]$, וכי הקיבול השיורי של (u, v) הוא חיובי. לפיכך, ניתן לשלוח עד $d_f(u, v) = \min(e[u], c_f(u, v))$ יחידות זרימה מ־u אל v בלי ש־$e[u]$ תהפוך להיות שלילית ובלי לעבור את הקיבול $c(u, v)$. ערך זה מחושב בשורה 3. העברת הזרימה מ־u אל v

מתבצעת על־ידי עדכון f בשורות 4-5 ועדכון e בשורות 6-7. לכן, אם f היא קדם־זרימה לפני הקריאה ל־PUSH, היא נותרת קדם־זרימה גם אחרי הקריאה.

נשים לב כי בקוד עבור PUSH, שום דבר אינו תלוי בגובהי u ו־v, ובכל זאת איננו מרשים לבצע קריאה ל־PUSH אלא אם כן $h[u] = h[v] + 1$. משום כך, זרימה עודפת נדחפת במורד רק בהפרש גבהים של יחידה אחת. על פי למה 27.13, לא קיימות קשתות שיוריות בין שני קדקודים שהפרש הגובה ביניהם גדול מ־1, ולכן לא נוכל להרוויח דבר מכך שנרשה דחיפה של זרימה במורד בהפרש גבהים גדול מ־1.

את הפעולה $\textsc{Push}(u, v)$ אנו מכנים **דחיפה** (push) מ־u אל v. אם פעולת דחיפה ניתנת לביצוע דרך קשת מסוימת (u, v) היוצאת מקדקוד u, אנו גם אומרים שניתן לבצע על u פעולת דחיפה. זוהי **דחיפה מרווה** (saturating push) אם בעקבותיה הקשת (u, v) הופכת להיות **רוויה** (saturated) (דהיינו, אם לאחר הדחיפה, $c_f(u, v) = 0$); אחרת, זוהי **דחיפה בלתי־מרווה** (nonsaturating push). אם קשת היא רוויה, היא אינה מופיעה ברשת השיורית.

הפעולה הבסיסית $\textsc{Lift}(u)$ ניתנת לביצוע אם u גולש, ואם $c_f(u, v) > 0$ גורר $h[u] \leq h[v]$ עבור כל הקדקודים v. במילים אחרות, ניתן להרים קדקוד גולש u אם עבור כל קדקוד v שעבורו קיים קיבול שיורי מ־u אל v, לא ניתן לדחוף זרימה מ־u אל v משום ש־v אינו נמוך מ־u. (נזכור כי על פי ההגדרה, המקור s וגם הבור t אינם יכולים לגלוש, ולכן לא ניתן להרים את s או את t.)

```
LIFT(u)
1  ▷ Applies when: u is overflowing and for all v ∈ V,
                   (u, v) ∈ E_f implies h[u] ≤ h[v].
2  ▷ Action: Increase the height of u.
3  h[u] ← 1 + min{h[v] : (u, v) ∈ E_f}
```

כאשר אנו קוראים לפעולה $\textsc{Lift}(u)$, אנו אומרים שהקדקוד u **מורם** (lifted). חשוב לציין שכאשר u מורם, E_f חייבת להכיל לפחות קשת אחת שיוצאת מ־u, כדי שחישוב המינימום בקוד לא יתבצע על קבוצה ריקה. עובדה זו נובעת מן ההנחה ש־u גולש. מכיוון ש־$e[u] > 0$, מתקיים $e[u] = f[V, u] > 0$, ומכאן שבהכרח קיים לפחות קדקוד אחד v שעבורו $f[v, u] > 0$. אולם אז:

$$\begin{aligned} c_f(u, v) &= c(u, v) - f[u, v] \\ &= c(u, v) + f[v, u] \\ &> 0 \end{aligned}$$

ומכאן נובע כי $(u, v) \in E_f$. הפעולה $\textsc{Lift}(u)$ מרימה אפוא את u לגובה המרבי המותר על פי האילוצים של פונקציות הגובה.

האלגוריתם הכללי

אלגוריתם דחיפת־הקדם־זרימה הכללי קורא לשגרה שלהלן כדי ליצור קדם־זרימה התחלתית ברשת הזרימה.

```
INITIALIZE-PREFLOW(G, s)
 1  for each vertex u ∈ V[G]
 2      do h[u] ← 0
 3         e[u] ← 0
 4  for each edge (u, v) ∈ E[G]
 5      do f[u, v] ← 0
 6         f[v, u] ← 0
 7  h[s] ← |V[G]|
 8  for each vertex u ∈ Adj[s]
 9      do f[s, u] ← c(s, u)
10         f[u, s] ← −c(s, u)
11         e[u] ← c(s, u)
```

INITIALIZE-PREFLOW יוצרת קדם־זרימה התחלתית f המוגדרת על־ידי:

$$f[u,v] = \begin{cases} c(u,v) & \text{אם } u = s \\ -c(v,u) & \text{אם } v = s \\ 0 & \text{אחרת} \end{cases} \tag{27.9}$$

כלומר, ממלאים כל קשת היוצאת מן המקור עד למלוא קיבולה, וכל הקשתות האחרות אינן נושאות שום זרימה. עבור כל קדקוד v הסמוך למקור מתקיים במצב ההתחלתי $e[v] = c(s, v)$. כמו כן, האלגוריתם הכללי מתחיל עם פונקציית גובה התחלתית h הנתונה על־ידי:

$$h[u] = \begin{cases} |V| & \text{אם } u = s \\ 0 & \text{אחרת} \end{cases}$$

זוהי פונקציית גובה מכיוון שהקשתות (u, v) היחידות שעבורן $h[u] > h[v] + 1$ הן אלה שעבורן $u = s$, וקשתות אלה רוויות, ומכאן שאינן שייכות לרשת השיורית.

האלגוריתם שלהלן מייצג את אלגוריתם הקדם־זרימה הכללי.

```
GENERIC-PREFLOW-PUSH(G)
1  INITIALIZE-PREFLOW(G, s)
2  while there exists an applicable push or lift operation
3      do select an applicable push or lift operation and perform it
```

לאחר אתחול הזרימה, האלגוריתם הכללי מבצע שוב ושוב את הפעולות הבסיסיות, בסדר כלשהו ובכל מקום שבו ניתן לבצען. הלמה הבאה מראה שכל עוד קיים קדקוד גולש, ניתן לבצע לפחות אחת משתי הפעולות.

למה 27.14 (על צומת גולש ניתן לבצע דחיפה או הרמה)
תהי $G = (V, E)$ רשת זרימה עם מקור s ובור t, תהי f קדם־זרימה, ותהי h פונקציית גובה כלשהי עבור f. אם u הוא קדקוד גולש כלשהו, אזי ניתן להפעיל עליו דחיפה או הרמה.

הוכחה עבור כל קשת שיורית (u, v), מתקיים $h(u) \le h(v) + 1$, מכיוון ש־h היא פונקציית גובה. אם לא ניתן לבצע על u פעולת דחיפה, פירוש הדבר שעבור כל הקשתות השיוריות (u, v) מתקיים בהכרח $h(u) < h(v) + 1$, ומכאן נובע כי $h(u) \le h(v)$. לכן ניתן לבצע על u פעולת הרמה. ■

נכונות שיטת דחיפת־הקדם־זרימה

כדי להראות שאלגוריתם דחיפת־הקדם־זרימה הכללי פותר את בעיית הזרימה המקסימלית, נוכיח תחילה שאם הוא עוצר, הקדם־זרימה f היא זרימה מקסימלית. מאוחר יותר נוכיח שהאלגוריתם אכן עוצר. נתחיל במספר הבחנות לגבי פונקציית הגובה h.

למה 27.15 (גובהם של קדקודים לעולם אינו קטן)
במהלך ביצוע GENERIC-PREFLOW-PUSH על רשת זרימה $G = (V, E)$, הגובה $h[u]$ של כל קדקוד $u \in V$ לעולם אינו קטן. יתירה מזו, בכל פעם שמבוצעת פעולת הרמה על קדקוד u, גובהו $h[u]$ גדל ב־1 לפחות.

הוכחה מכיוון שגובהי קדקודים משתנים אך ורק במהלך פעולות הרמה, די להוכיח רק את החלק השני של טענת הלמה. אם מבצעים על קדקוד u פעולת הרמה, אזי עבור כל הקדקודים v כך ש־$(u, v) \in E_f$, מתקיים $h[u] \le h[v]$; מכאן נובע כי $h[u] < 1 + \min\{h[v] : (u, v) \in E_f\}$, ולכן פעולת ההרמה בהכרח מגדילה את $h[u]$. ■

למה 27.16
תהי $G = (V, E)$ רשת זרימה עם מקור s ובור t. במהלך ביצוע GENERIC-PREFLOW-PUSH על G, המאפיין h מתוחזק כפונקציית גובה.

הוכחה ההוכחה היא באינדוקציה על מספר הפעולות הבסיסיות המבוצעות. כפי שראינו, במצב ההתחלתי h היא אכן פונקציית גובה.

אנו טוענים שאם h היא פונקציית גובה, אזי היא נותרת פונקציית גובה גם לאחר ביצוע פעולת LIFT(u). אם נתבונן בקשת שיורית $(u, v) \in E_f$ היוצאת מ־u, אזי הפעולה LIFT(u) מבטיחה כי לאחריה $h[u] \le h[v] + 1$. עתה נתבונן בקשת שיורית (w, u) הנכנסת ל־u. על פי למה 27.15, אם $h[w] \le h[u] + 1$ לפני ביצוע LIFT(u), אזי $h[w] < h[u] + 1$ אחרי ביצוע הפעולה. לפיכך, לאחר ביצוע LIFT(u), h היא עדיין פונקציית גובה.

עתה נתבונן בפעולה PUSH(u, v). פעולה זו עשויה להוסיף את הקשת (u, v) ל־E_f, והיא עשויה גם לסלק את הקשת (u, v) מ־E_f. במקרה הראשון, $h[v] = h[u] - 1$, ולכן h נותרת פונקציית גובה. במקרה השני, הסרתה של (u, v) מן הרשת השיורית גורמת להסרת האילוץ המתאים, ולכן גם הפעם h נותרת פונקציית גובה. ■

הלמה הבאה מציגה תכונה חשובה של פונקציות גובה.

למה 27.17
תהי $G = (V, E)$ רשת זרימה עם מקור s ובור t, תהי f קדם־זרימה ב־G, ותהי h פונקציית גובה על V. אזי, ברשת השיורית G_f לא קיים מסלול מן המקור s אל הבור t.

הוכחה נניח בדרך השלילה כי קיים ב־G_f מסלול $p = \langle v_0, v_1, \ldots, v_k \rangle$ מ־s ל־t, כאשר $v_0 = s$ ו־$v_k = t$. בלי הגבלת הכלליות, p הוא מסלול פשוט, ולכן $k < |V|$. עבור $i = 0, 1, \ldots, k-1$, מתקיים $(v_i, v_{i+1}) \in E_f$. מכיוון ש־h היא פונקציית גובה, $h(v_i) \le h(v_{i+1}) + 1$ עבור $i = 0, 1, \ldots, k-1$. צירוף האי־שוויונים הללו על כל המסלול p נותן $h(s) \le h(t) + k$. אולם מאחר ש־$h(t) = 0$, אנו מקבלים $h(s) \le k < |V|$, בסתירה לדרישה שבפונקציית גובה יתקיים $h(s) = |V|$. ■

עתה אנו ערוכים להוכחת הטענה שאם אלגוריתם דחיפת־הקדם־זרימה הכללי עוצר, אזי הקדם־זרימה שהוא מחשב היא זרימה מקסימלית.

משפט 27.18 (נכונותו של אלגוריתם דחיפת־הקדם־זרימה הכללי)
אם האלגוריתם GENERIC-PREFLOW-PUSH עוצר כאשר מריצים אותו על רשת זרימה $G = (V, E)$ עם מקור s ובור t, אזי הקדם־זרימה f שהוא מחשב היא זרימה מקסימלית ב־G.

הוכחה אם האלגוריתם הכללי עוצר, אזי כל קדקוד ב־$V - \{s, t\}$ חייב להיות בעל זרימה עודפת 0, שכן על פי למות 27.14 ו־27.16 והאינווריאנטה ש־f היא תמיד קדם־זרימה, לא קיימים קדקודים גולשים. לכן, f היא זרימה. מאחר ש־h היא פונקציית גובה, הרי שעל פי למה 27.17 לא קיים מסלול מ־s אל t ברשת השיורית G_f לכן, על פי המשפט של זרימה־מקסימלית־חתך־מינימלי, f היא זרימה מקסימלית. ■

ניתוח שיטת דחיפת־הקדם־זרימה

כדי להראות שאלגוריתם דחיפת־הקדם־זרימה הכללי אכן עוצר, נחסום את מספר הפעולות שהוא מבצע. נחסום בנפרד כל אחד משלושת סוגי הפעולות — הרמה, דחיפה מרווה, ודחיפה לא־מרווה. ידיעת החסמים האלה מאפשרת בנייה מיידית של אלגוריתם שרץ בזמן $O(V^2E)$. אולם לפני שנתחיל בניתוח, נוכיח למה חשובה.

למה 27.19
תהי $G = (V, E)$ רשת זרימה עם מקור s ובור t, ותהי f קדם־זרימה ב־G. אזי, עבור כל קדקוד גולש u, קיים מסלול פשוט מ־u אל s ברשת השיורית G_f.

הוכחה תהי $U = \{v : v$ קיים ב־G_f מסלול פשוט מ־u אל $v\}$, ונניח בדרך השלילה כי $s \notin U$. תהי $\bar{U} = V - U$.

אנו טוענים שעבור כל זוג קדקודים $v \in U$ ו־$w \in \bar{U}$ מתקיים $f(w, u) \le 0$. מדוע? אם $f(w, u) > 0$, אזי $f(v, w) < 0$, ומכך נובע כי $c_f(v, w) = c(v, w) - f(v, w) > 0$. מכאן שקיימת קשת $(v, w) \in E_f$, ולכן קיים ב־G_f מסלול פשוט מן הצורה $u \rightsquigarrow v \rightarrow w$, בסתירה לאופן שבו בחרנו את w.

לכן, בהכרח מתקיים $f(\bar{U}, U) \le 0$, שכן כל אחד מהביטויים בסכום מובלע זה הוא אי־חיובי. לכן, על פי משוואה (27.8) ולמה 27.1, נוכל להסיק כי:

$$\begin{aligned} e(U) &= f(V, U) \\ &= f(\bar{U}, U) + f(U, U) \\ &= f(\bar{U}, U) \\ &\le 0 \end{aligned}$$

הזרימות העודפות הן אי־שליליות עבור כל הקדקודים ב־$V - \{s\}$, מאחר שהנחנו כי $U \subseteq V - \{s\}$, הרי שבהכרח מתקיים $e(v) = 0$ עבור כל הקדקודים $v \in U$. בפרט $e(u) = 0$, בסתירה להנחתנו ש־u גולש. ■

הלמה הבאה חוסמת את גובהי הקדקודים, ומסקנתה חוסמת את המספר הכולל של פעולות ההרמה המבוצעות.

למה 27.20
תהי $G = (V, E)$ רשת זרימה עם מקור s ובור t. בכל זמן נתון במהלך ביצועה של GENERIC-PREFLOW-PUSH על G, מתקיים $h[u] \le 2|V| - 1$ עבור כל הקדקודים $u \in V$.

הוכחה גובה המקור s וגובה הבור t אינם משתנים לעולם, מכיוון שעל פי ההגדרה קדקודים אלה אינם גולשים. לכן, תמיד מתקיים $h[s] = |V|$ ו־$h[t] = 0$.

מאחר שפעולת הרמה מתבצעת אך ורק על קדקוד גולש, נוכל להתבונן בקדקוד גולש כלשהו $u \in V - \{s, t\}$. על פי למה 27.19, קיים ב־G_f מסלול פשוט p מ־u אל s. יהי $p = \langle v_0, v_1, \ldots, v_k \rangle$, כאשר $v_0 = u$, $v_k = s$, ו־$k \le |V| - 1$ משום ש־p פשוט. עבור $i = 0, 1, \ldots, k - 1$ מתקיים $(v_i, v_{i+1}) \in E_f$, ולכן, $h[v_i] \le h[v_{i+1}] + 1$, על פי למה 27.16. אם נרחיב את האי־שוויונים האלה למסלול p כולו, נקבל $h[u] = h[v_0] \le h[v_k] + k \le h[s] + (|V| - 1) = 2|V| - 1$. ■

מסקנה 27.21 (חסם על מספר פעולות ההרמה)
תהי $G = (V, E)$ רשת זרימה עם מקור s ובור t. אזי, במהלך ביצוע האלגוריתם GENERIC-PREFLOW-PUSH על G, מספר פעולות ההרמה המבוצעות על כל קדקוד הוא לכל היותר $2|V| - 1$, ומספר פעולות ההרמה המבוצעות בסך הכל הוא לכל היותר $(2|V| - 1)(|V| - 2) < 2|V|^2$.

הוכחה פעולת הרמה ניתן לבצע רק על הקדקודים ב־$V - \{s, t\}$, שמספרם $|V| - 2$. יהי $u \in V - \{s, t\}$. הפעולה $\text{LIFT}(u)$ מגדילה את $h[u]$. ערכו של $h[u]$ הוא בתחילה 0, ועל פי למה 27.20, הערך גדל עד $2|V| - 1$ לכל היותר. לכן, על כל קדקוד $u \in V - \{s, t\}$ מתבצעות לכל היותר $2|V| - 1$ פעולות הרמה, ומספר פעולות ההרמה המבוצעות בסך הכל הוא לכל היותר $(2|V| - 1)(|V| - 2) < 2|V|^2$. ■

בעזרת למה 27.20 ניתן גם לחסום את מספר הדחיפות המרוות.

למה 27.22 (חסם על מספר הדחיפות המרוות)
במהלך ביצוע האלגוריתם GENERIC-PREFLOW-PUSH על רשת זרימה כלשהי $G = (V, E)$, מספר הדחיפות המרוות הוא לכל היותר $2|V||E|$.

הוכחה עבור זוג קדקודים כלשהו $u, v \in V$, נתבונן בדחיפות המרוות מ־u אל v ומ־v אל u. אם קיימות דחיפות כאלה, הרי שלפחות אחת מן הקשתות (u, v) ו־(v, u) היא קשת בפועל ב־E. עתה, נניח שהתבצעה דחיפה מרווה מ־u אל v. כדי שתתבצע מאוחר יותר דחיפה נוספת מ־u אל v, האלגוריתם חייב תחילה לדחוף זרימה מ־v אל u, ודחיפה כזאת אינה יכולה להתבצע לפני ש־$h[v]$ גדל ב־2 לפחות. באופן דומה, $h[u]$ חייב לגדול ב־2 לפחות בין שתי דחיפות מרוות מ־v אל u.

נתבונן בסדרת השלמים A הנתונה על־ידי $h[u] + h[v]$ עבור כל דחיפה מרווה בין הקדקודים u ו־v. ברצוננו לחסום את אורכה של סדרה זו. בעת ביצוע הדחיפה הראשונה באחד הכיוונים בין u ו־v, בהכרח מתקיים $h[u] + h[v] \ge 1$; לכן, ערכו של המספר הראשון ב־A הוא לפחות 1. כאשר מתבצעת הדחיפה האחרונה, מתקיים, על פי למה 27.20, $h[u] + h[v] \le (2|V| - 1) + (2|V| - 2) = 4|V| - 3$; לכן, ערכו של המספר האחרון ב־A הוא לכל היותר $4|V| - 3$. לפי הטיעון מן הפסקה הקודמת, ב־A יכול להופיע לכל היותר כל שלם שני. לכן, מספר השלמים בסדרה A הוא $2|V| - 1 + 1/2((4|V| - 3) - 1) = 4|V| - 2$ לכל היותר. (אנו מוסיפים 1 כדי להבטיח שאנו סופרים גם את שני קצות הסדרה.) לפיכך, המספר הכולל של דחיפות מרוות בין הקדקודים u ו־v הוא לכל היותר $4|V| - 2$. נכפול מספר זה במספר הקשתות ונקבל כי המספר הכולל של דחיפות מרוות הוא לכל היותר $(4|V| - 2)|E| < 4|V||E|$. ■

הלמה הבאה חוסמת את מספר הדחיפות הלא־מרוות שמבצע אלגוריתם דחיפת־הקדם־זרימה הכללי.

למה 27.23 (חסם על מספר הדחיפות הלא־מרוות)
במהלך ביצוע האלגוריתם GENERIC-PREFLOW-PUSH על רשת זרימה כלשהי $G = (V, E)$, מספר הדחיפות הלא־מרוות הוא לכל היותר $4|V|^2(|V| + |E|)$.

הוכחה נגדיר פונקציית פוטנציאל $\Phi = \sum_{v \in X} h[v]$, כאשר $X \subseteq V$ היא קבוצת הקדקודים הגולשים. במצב ההתחלתי, $\Phi = 0$. נשים לב כי הרמת קדקוד u מגדילה את Φ ב־$2|V|$ לכל היותר, שכן הקבוצה שעליה מסכמים היא אותה קבוצה ולא ניתן להרים את u יותר מן הגובה המקסימלי האפשרי שלו, שהוא, על פי למה 27.20, $2|V|$. לכל היותר. כמו כן, דחיפה מרווה מקדקוד u אל קדקוד v מגדילה את Φ ב־$2|V|$ לכל היותר, שכן גובהי הקדקודים אינם משתנים, ורק הקדקוד v, שגובהו לכל היותר $2|V|$, ייתכן שיהפוך לקדקוד גולש. לבסוף, נשים לב כי דחיפה לא־מרווה מ־u אל v מקטינה את Φ ב־1 לפחות, שכן לאחר הדחיפה, u כבר אינו גולש, v גולש גם אם לא גלש לפני הדחיפה, וכן $h[v] - h[u] = -1$.

לכן, במהלך ביצוע האלגוריתם, הגידול הכולל בערכה של Φ מוגבל על־פי מסקנה 27.21 ולמה 27.22 ל־$(2|V|)(2|V|^2) + (2|V|)(2|V|\cdot|E|) = 4|V|^2(|V| + |E|)$ לכל היותר. מאחר ש־$\Phi \geq 0$, הרי שהירידה הכוללת בערכה של Φ, ולכן גם המספר הכולל של דחיפות לא־מרוות, הוא $4|V|^2(|V| + |E|)$ לכל היותר. ■

עתה ערוכה הבמה להצגת הניתוח הבא של השגרה GENERIC-PREFLOW-PUSH, ומכאן גם של כל אלגוריתם המבוסס על שיטת דחיפת־הקדם־זרימה.

משפט 27.24
במהלך ביצוע GENERIC-PREFLOW-PUSH על רשת זרימה כלשהי $G = (V, E)$, מספר הפעולות הבסיסיות הוא $O(V^2E)$.

הוכחה נובעת ישירות ממסקנה 27.21 ומלמות 27.22 ו־27.23. ■

מסקנה 27.15
קיים מימוש של אלגוריתם דחיפת־הקדם־זרימה הכללי שרץ בזמן $O(V^2E)$ על רשת זרימה כלשהי $G = (V, E)$.

הוכחה בתרגיל 27.4-1 תתבקש להראות כיצד לממש את האלגוריתם הכללי עם תקורה של $O(V)$ לפעולת הרמה ו־$O(1)$ לפעולת דחיפה. המסקנה נובעת מתוצאת התרגיל. ■

תרגילים

27.4-1
הראה כיצד ניתן לממש את אלגוריתם דחיפת־הקדם־זרימה הכללי כך שפעולת הרמה תתבצע בזמן $O(V)$ ופעולת דחיפה תתבצע בזמן $O(1)$, ולכן זמן הריצה הכולל יהיה $O(V^2E)$.

27.4-2
הוכח שאלגוריתם דחיפת־הקדם־זרימה הכללי משתמש בזמן כולל של $O(VE)$ בלבד לביצוע כל $O(V^2)$ פעולות ההרמה.

27.4-3
נניח שמצאנו זרימה מקסימלית ברשת זרימה $G = (V, E)$, באמצעות אלגוריתם דחיפת־קדם־זרימה. כתוב אלגוריתם מהיר למציאת חתך מינימלי ב־G.

27.4-4
כתוב אלגוריתם דחיפת־קדם־זרימה יעיל למציאת זיווג מקסימלי בגרף דו־צדדי. נתח את האלגוריתם שלך.

27.4-5
נניח שכל קיבולי הקשתות ברשת זרימה $G = (V, E)$ שייכים לקבוצה $\{1, 2, \ldots, k\}$. נתח את זמן הריצה של אלגוריתם דחיפת־הקדם־זרימה הכללי במונחים של $|V|$, $|E|$ ו־k. (**רמז:** כמה דחיפות לא־מרוות ניתן לבצע על כל קשת לפני שהיא הופכת להיות רוויה?)

27.4-6
הראה שניתן לשנות את שורה 7 ב־INITIALIZE-PREFLOW לשורה:

$$h(s) \leftarrow |V[G]| - 2$$

בלי להשפיע על נכונותו של אלגוריתם דחיפת־הקדם־זרימה הכללי או על ביצועיו האסימפטוטיים.

27.4-7
יהי $\delta_f(u, v)$ המרחק (מספר הקשתות) מ־u אל v ברשת השיורית G_f. הראה שהשגרה GENERIC-PREFLOW-PUSH שומרת על התכונות הבאות: $h[u] < |V|$ גורר $h[u] < \delta_f(u, t)$, וכן $h[u] \geq |V|$ גורר $h[u] - |V| \leq \delta_f(u, s)$.

**** 27.4-8***
כמו בתרגיל הקודם, יהי $\delta_f(u, v)$ המרחק מ־u אל v ברשת השיורית G_f. הראה כיצד ניתן לשנות את אלגוריתם דחיפת־הקדם־זרימה הכללי כך שישמור על התכונה: $h[u] < |V|$ גורר $h[u] = \delta_f(u, t)$ וגם $h[u] \geq |V|$ גורר $h[u] - |V| = \delta_f(u, s)$. הזמן הכולל שיקדיש האלגוריתם שלך לשמירה על תכונה זו צריך להיות $O(VE)$.

27.4-9
הראה שמספר הדחיפות הלא־מרוות שמבצע GENERIC-PREFLOW-PUSH על רשת זרימה $G = (V, E)$ הוא לכל היותר $4|V|^2|E|$, עבור $|V| \geq 4$.

* 27.5 אלגוריתם הרם־וקדם

השיטה של דחיפת־קדם־זרימה מאפשרת לנו לבצע את הפעולות הבסיסיות בכל סדר שהוא. אולם, על־ידי בחירה קפדנית של סדר ביצוע הפעולות ועל־ידי ניהול יעיל של מבנה הנתונים המייצג את הרשת, ניתן לפתור את בעיית הזרימה המקסימלית בזמן טוב יותר מאשר החסם $O(V^2E)$ שניתן על־ידי מסקנה 27.25. אנו נבחן עתה את האלגוריתם הרם־וקדם, אלגוריתם דחיפת־קדם־זרימה שזמן ריצתו $O(V^3)$, שהוא מהיר אסימפטוטית לפחות כמו $O(V^2E)$.

האלגוריתם הרס־וקדם מנהל רשימה של הקדקודים ברשת. האלגוריתם סורק את הרשימה, מראשה ועד סופה, בוחר בכל פעם קדקוד גולש u ואז "פורק" אותו, כלומר, מבצע פעולות דחיפה והרמה עד אשר u אינו מכיל עוד זרימה עודפת חיובית. בכל פעם שמבוצעת על קדקוד פעולת הרמה, הוא מועבר לראש הרשימה (מכאן השם "הרס־וקדם") והאלגוריתם מתחיל מחדש את סריקתו.

נכונותו של האלגוריתם הרס־וקדם וניתוחו מבוססים על המושג "קשתות מותרות": אותן קשתות ברשת השיורית שדרכן ניתן לדחוף זרימה. לאחר שנוכיח כמה תכונות שמקיימת רשת הקשתות המותרות, נחקור את פעולת הפריקה ואז נציג את האלגוריתם הרס־וקדם עצמו וננתח אותו.

קשתות ורשתות מותרות

אם $G = (V, E)$ היא רשת זרימה עם מקור s ובור t, f היא קדם־זרימה ב־G, ו־h היא פונקציית גובה, אזי (u,v) מכונה **קשת מותרת** (admissible edge) אם $c_f(u, v) > 0$ ו־$h(u) = h(v) + 1$. אחרת, (u, v) היא **בלתי־מותרת** (inadmissible). **הרשת המותרת** (admissible network) היא $G_{f,h} = (V, E_{f,h})$, כאשר $E_{f,h}$ היא קבוצת הקשתות המותרות.

הרשת המותרת מורכבת מאותן קשתות שדרכן ניתן לדחוף זרימה. הלמה הבאה מראה שרשת זו היא גרף מכוון ללא מעגלים (גמ"ל).

למה 27.26 (הרשת המותרת היא חסרת מעגלים)
אם $G = (V, E)$ היא רשת זרימה, f היא קדם־זרימה ב־G, ו־h היא פונקציית גובה על G, אזי הרשת המותרת $G_{f,h} = (V, E_{f,h})$ היא חסרת מעגלים.

הוכחה ההוכחה היא בדרך השלילה. נניח כי $G_{f,h}$ מכילה מעגל $p = \langle v_0, v_1, \ldots, v_k \rangle$, כאשר $v_0 = v_k$ ו־$k > 0$. מכיוון שכל קשת ב־p היא מותרת, מתקיים $h(v_{i-1}) = h(v_i) + 1$ עבור $i = 1, 2, \ldots, k$. נסכם על המעגל כולו ונקבל:

$$\begin{aligned}\sum_{i=1}^{k} h(v_{i-1}) &= \sum_{i=1}^{k} (h(v_i)+1) \\ &= \sum_{i=1}^{k} h(v_i) + k\end{aligned}$$

מאחר שכל קדקוד במעגל p מופיע פעם אחת בכל אחד מן הסכומים, אנו מקבלים כי $k = 0$, בסתירה להנחתנו. ■

שתי הלמות הבאות מראות כיצד פעולות דחיפה והרמה משנות את הרשת המותרת.

למה 27.27
תהי $G = (V, E)$ רשת זרימה, תהי f קדם־זרימה ב־G, ותהי h פונקציית גובה. אם קדקוד u גולש ו־(u, v) היא קשת מותרת, אזי ניתן לבצע $\textsc{Push}(u, v)$. הפעולה אינה יוצרת קשתות מותרות חדשות אולם היא עשויה לגרום לכך שהקשת (u, v) תהפוך להיות בלתי־מותרת.

הוכחה על פי הגדרתה של קשת מותרת, ניתן לדחוף זרימה מ־u אל v. מאחר ש־u גולש, ניתן לבצע $\textsc{Push}(u, v)$. הקשת השיורית החדשה היחידה שעשויה להיווצר כתוצאה מדחיפת זרימה מ־u אל v היא (v, u). מאחר ש־$h(v) = h(u) - 1$, הקשת (v, u) אינה יכולה להיעשות מותרת. אם הדחיפה היא דחיפה מרווה, אזי לאחריה $c_f(u, v) = 0$, ו־(u, v) נעשית בלתי־מותרת. ■

למה 27.28
תהי $G = (V, E)$ רשת זרימה, תהי f קדם־זרימה ב־G, ותהי h פונקציית גובה. אם קדקוד u גולש ולא קיימות קשתות מותרות היוצאות מ־u, אזי ניתן לבצע $\textsc{Lift}(u)$. לאחר פעולת ההרמה, קיימות לפחות קשת מותרת אחת היוצאת מ־u, אולם אין קשתות מותרות הנכנסות אל u.

הוכחה אם u גולש, אזי על פי למה 27.14, ניתן לבצע עליו פעולת דחיפה או פעולת הרמה. אם לא קיימות קשתות מותרות היוצאות מ־u, לא ניתן לדחוף זרימה מ־u ולפיכך ניתן לבצע $\textsc{Lift}(u)$. לאחר פעולת ההרמה, $h[u] = 1 + \min\{h[v] : (u, v) \in E_f\}$. לכן, אם v הוא קדקוד שעבורו מתקבל המינימום בקבוצה זו, אזי הקשת (u, v) נעשית מותרת. מכאן שלאחר פעולת ההרמה קיימת לפחות קשת מותרת אחת היוצאת מ־u.

כדי להראות שלאחר פעולת הרמה לא קיימות קשתות מותרות הנכנסות ל־u, נניח שקיים קדקוד v כך ש־(v, u) מותרת. אזי, $h[v] = h[u] + 1$ לאחר ההרמה, ולכן $h[v] > h[u] + 1$ ממש לפני ההרמה. אולם על פי למה 27.13, לא קיימות קשתות שיוריות בין קדקודים שהפרש הגבהים שלהם גדול מ־1. יתירה מזו, הרמת קדקוד אינה משנה את הרשת השיורית. לכן, (u, v) אינה שייכת לרשת השיורית, ומכאן שהיא אינה יכולה להשתייך לרשת המותרת. ■

רשימות שכנים

האלגוריתם הרס־וקדם מארגן קשתות ב"רשימות שכנים". בהינתן רשת זרימה $G = (V, E)$, **רשימת השכנים** (neighbor list) $N[u]$ של קדקוד $u \in V$ היא רשימה חד־מקושרת של הקדקודים השכנים ל־u ב־G. לכן, קדקוד v מופיע ברשימה $N[u]$ אם $(u, v) \in E$ או $(v, u) \in E$. רשימת השכנים $N[u]$ מכילה בדיוק אותם קדקודים v שעבורם תיתכן קשת שיורית (u, v). על הקדקוד הראשון ב־$N[u]$ מצביע $head[N[u]]$. $next\text{-}neighbor[v]$ מצביע על הקדקוד הבא אחרי v ברשימת השכנים; מצביע זה הוא NIL אם v הוא הקדקוד האחרון ברשימת השכנים.

האלגוריתם הרס־וקדם סורק כל אחת מרשימות השכנים בסדר שרירותי שהוא קבוע במהלך ביצוע האלגוריתם. עבור כל קדקוד u, השדה $current[u]$ מצביע על הקדקוד הנוכחי הנבחן ב־$N[u]$. במצב ההתחלתי, מציבים ב־$current[u]$ את $head[N[u]]$.

פריקת קדקוד גולש

פריקה (discharge) של קדקוד גולש u מתבצעת על־ידי דחיפת כל הזרימה העודפת שלו דרך קשתות מותרות לקדקודים שכנים, תוך הרמת u ככל שנדרש כדי להפוך קשתות היוצאות מ־u למותרות. להלן הפסאודו־קוד לביצוע פריקה.

```
DISCHARGE(u)
1  while e[u] > 0
2      do v ← current[u]
3         if v = NIL
4            then LIFT(u)
5                 current[u] ← head[N[u]]
6         elseif c_f(u, v) > 0 and h[u] = h[v] + 1
7            then PUSH(u, v)
8         else current[u] ← next-neighbor[v]
```

באיור 27.10 מוצגות איטרציות אחדות של לולאת ה־**while** שבשורות 1-8, המתבצעת כל עוד הקדקוד u מכיל זרימה עודפת חיובית. כל איטרציה מבצעת בדיוק אחת משלוש פעולות, בתלות בקדקוד הנוכחי v מרשימת השכנים $N[u]$:

1. אם v הוא NIL, פירוש הדבר שעברנו את סופה של $N[u]$. שורה 4 מבצעת הרמה של הקדקוד u, ואז שורה 5 קובעת את השכן הנוכחי של u כקדקוד הראשון ב־$N[u]$. (למה 27.29, המובאת בהמשך, קובעת כי במצב זה ניתן לבצע פעולת הרמה.)
2. אם v אינו NIL ו־(u, v) היא קשת מותרת (דבר זה נקבע על־ידי הבדיקה שבשורה 6), אזי שורה 7 דוחפת חלק מן הזרימה העודפת של u (או אולי את כולה) אל הקדקוד v.
3. אם v אינו NIL אולם (u, v) בלתי־מותרת, אזי שורה 8 **מקדמת** את $current[u]$ למיקום הבא ברשימת השכנים.

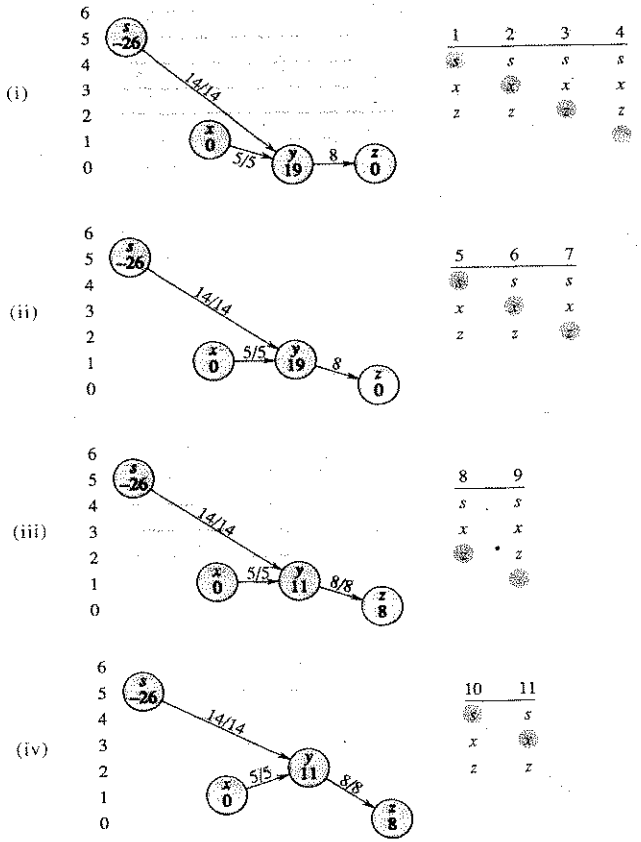
נשים לב לכך שאם מריצים את DISCHARGE על קדקוד גולש u, אזי הפעולה האחרונה שמבצעת DISCHARGE היא בהכרח דחיפת זרימה מ־u. מדוע? השגרה עוצרת רק כאשר $e[u]$ מתאפס, ואילו פעולת ההרמה וקידום המצביע $current[u]$ אינם משפיעים על ערכו של $e[u]$.

עלינו להיות בטוחים שכאשר DISCHARGE קוראת ל־PUSH או ל־LIFT, אכן ניתן לבצע פעולה זו. הלמה הבאה מוכיחה זאת.

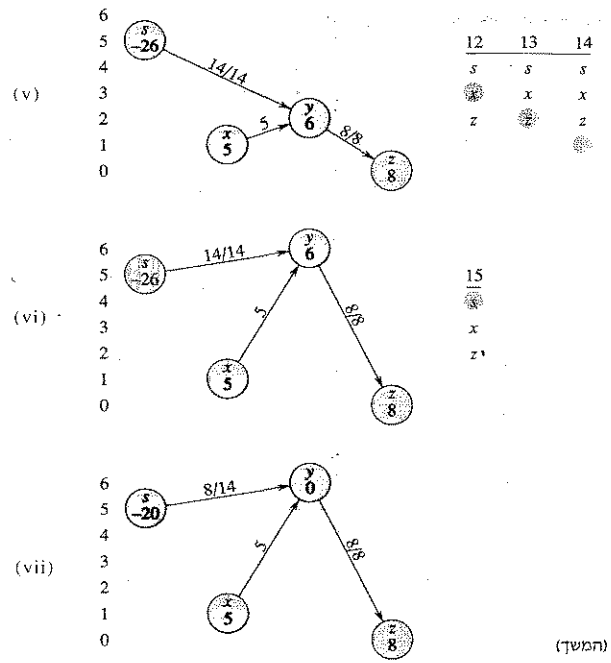
למה 27.29
אם DISCHARGE קוראת ל־$\text{PUSH}(u, v)$ בשורה 7, ניתן לבצע על (u, v) פעולת דחיפה. אם DISCHARGE קוראת ל־$\text{LIFT}(u)$ בשורה 4, ניתן לבצע על u פעולת הרמה.

הוכחה הבדיקות בשורות 1 ו־6 מבטיחות שפעולת דחיפה מתרחשת רק אם ניתן לבצעה, ובכך הוכחנו את הטענה הראשונה בלמה.

כדי להוכיח את הטענה השנייה, בהתאם לבדיקה שבשורה 1 ועל פי למה 27.28, כל שעלינו להראות הוא שכל הקשתות היוצאות מ־u הן בלתי־מותרות. נשים לב כי ככל ש־$\text{DISCHARGE}(u)$ חוזרת ונקראת, המצביע $current[u]$ הולך ויורד ברשימה $N[u]$. כל "מעבר" מתחיל בראש הרשימה $N[u]$ ומסתיים כאשר $current[u] = \text{NIL}$, ובנקודה זו u מורם ומתחיל מעבר חדש. כדי שהמצביע $current[u]$ יחלוף על פני קדקוד $v \in N[u]$ במהלך המעבר על הרשימה, הבדיקה שבשורה 6 חייבת לקבוע כי הקשת (u, v) היא בלתי־מותרת. לכן, כאשר מסתיים מעבר על הרשימה, פירוש הדבר שכל קשת היוצאת מ־u נמצאה בלתי־מותרת בזמן זה או אחר במהלך המעבר. ההבחנה החשובה היא שעם סיום המעבר, כל קשת היוצאת מ־u היא עדיין בלתי־מותרת. מדוע? על פי למה 27.27, דחיפות אינן יכולות ליצור קשתות מותרות כלשהן, ובוודאי שלא כזאת שיוצאת מ־u. לכן, כל קשת מותרת נוצרת בהכרח על־ידי פעולת הרמה. אולם הקדקוד u אינו מורם במהלך המעבר, ועל פי למה 27.28, לכל קדקוד אחר v המורם במהלך המעבר אין קשתות מותרות הנכנסות אליו. לכן, עם סיום המעבר, כל הקשתות היוצאות מ־u נשארות בלתי־מותרות, ובכך הוכחנו את הלמה. ■



איור 27.10 פריקת קדקוד. נדרשות 15 איטרציות של לולאת ה־**while** בשגרה DISCHARGE כדי לדחוף את כל הזרימה העודפת מהקדקוד y. באיור מופיעים רק הקדקודים השכנים ל־y והקשתות הנכנסות ל־y או יוצאות ממנו. בכל חלק באיור, המספר המופיע בתוך כל קדקוד מציין את הזרימה העודפת של קדקוד זה בתחילת האיטרציה הראשונה המוצגת בחלק זה, וכל קדקוד מוצג בגובה שלו במהלך כל החלק הזה. מימין מוצגת רשימת השכנים $N[y]$ בתחילת כל איטרציה, ומעליה מספר האיטרציה. השכן המוצלל הוא $current[y]$. **(i)** במצב ההתחלתי, יש לדחוף מ־y 19 יחידות זרימה עודפת, ו־$current[y] = s$. איטרציות 1, 2 ו־3 פשוט מקדמות את $current[y]$, שכן לא קיימות קשתות מותרות היוצאות מ־y. באיטרציה 4, $current[y] = \text{NIL}$ (הדבר מוצג על־ידי כך שהעיגול המוצלל מופיע מנחת לרשימת השכנים), ולפיכך y מורם ו־$current[y]$ עובר לראש רשימת השכנים. **(ii)** לאחר ההרמה, גובהו של הקדקוד y הוא 1. איטרציות 5 ו־6 מוצאות שהקשתות (y, s) ו־(y, x) שתיהן בלתי־מותרות, אולם באיטרציה 7 נדחפות 8 יחידות זרימה עודפת מ־y אל z. עקב פעולת הדחיפה, $current[y]$ אינו מקודם באיטרציה זו. **(iii)** מאחר שפעולת הדחיפה באיטרציה 7 הפכה את הקשת (y, z) לרוויה, הרי שאיטרציה 8 מוצאת אותה בלתי־מותרת. באיטרציה 9, $current[y] = \text{NIL}$, ולכן y שוב מורם ו־$current[y]$ עובר לראש הרשימה. **(iv)** באיטרציה 10, (y, s) בלתי־מותרת, אולם באיטרציה 11 נדחפות 5 יחידות זרימה עודפת מ־y אל x. **(v)** מאחר ש־$current[y]$ לא קודם באיטרציה 11, איטרציה 12 מוצאת כי (y, x) בלתי־מותרת. איטרציה 13 מוצאת כי (y, z) בלתי־מותרת, ואיטרציה 14 מרימה את הקדקוד y ומחזירה שוב את $current[y]$ לראש הרשימה. **(vi)** איטרציה 15 דוחפת 6 יחידות זרימה עודפת מ־y אל s. **(vii)** הקדקוד y אינו מכיל עתה כל זרימה עודפת ו־DISCHARGE עוצרת. בדוגמה זו, DISCHARGE מתחילה וגם עוצרת כאשר המצביע לקדקוד הנוכחי מצביע על ראש רשימת השכנים, אולם באופן כללי הדבר אינו תמיד כך.



איור 27.10 (המשך)

האלגוריתם הרם־וקדם

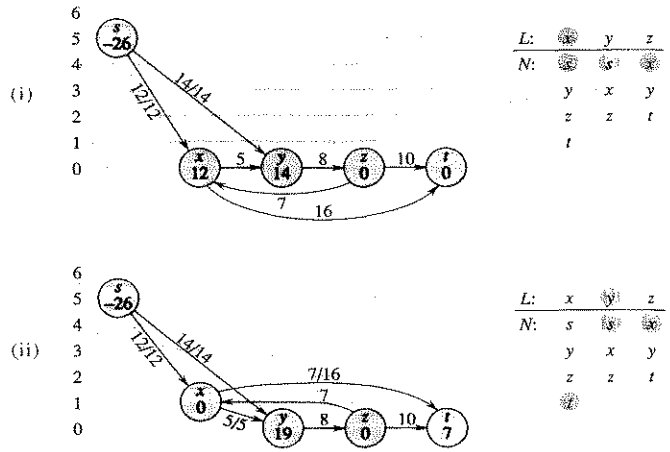
באלגוריתם הרם־וקדם אנו מתחזקים רשימה מקושרת L המורכבת מכל הקדקודים ב־$V - \{s, t\}$. תכונת מפתח של L היא שהקדקודים בה מסודרים בסדר טופולוגי על פי הרשת המותרת. (כזכור מלמה 27.26, הרשת המותרת היא גמ"ל.)

הפסאודו־קוד עבור האלגוריתם הרם־וקדם מניח שכבר נוצרו רשימות השכנים $N[u]$ עבור כל הקדקודים u. כמו כן, האלגוריתם מניח כי $next[u]$ מצביע על הקדקוד העוקב ל־u ברשימה L, וכי, כרגיל, אם u הוא הקדקוד האחרון ברשימה אזי $next[u] = \text{NIL}$.

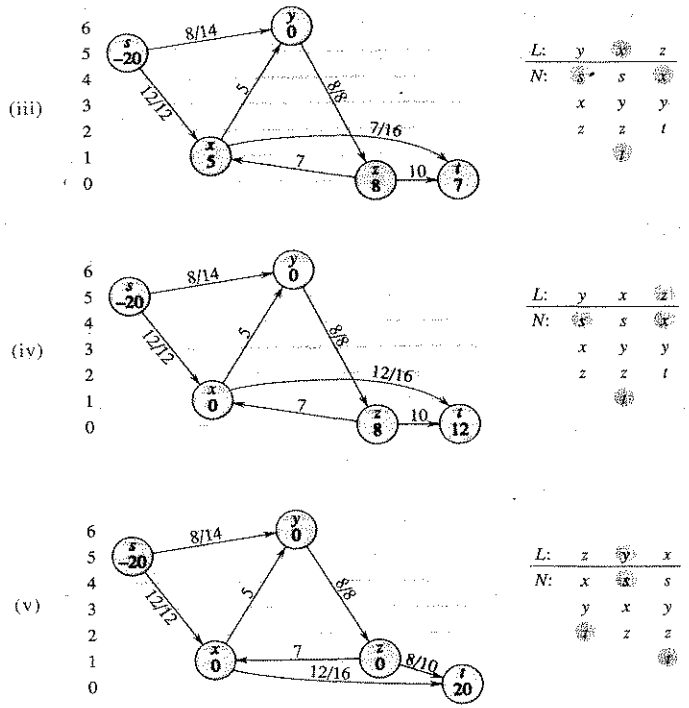
```
LIFT-TO-FRONT(G, s, t)
 1  INITIALIZE-PREFLOW(G, s)
 2  L ← V[G] − {s, t}, in any order
 3  for each vertex u ∈ V[G] − {s, t}
 4      do current[u] ← head[N[u]]
 5  u ← head[L]
 6  while u ≠ NIL
 7      do old-height ← h[u]
 8         DISCHARGE(u)
 9         if h[u] > old-height
10           then move u to the front of list L
11         u ← next[u]
```


האלגוריתם הרם־וקדם, כפי שהוא מנוסח בשגרה LIFT-TO-FRONT, פועל כדלקמן: שורה 1 מאתחלת את הקדם־זרימה ואת גובהי הקדקודים לאותם ערכים כמו אלגוריתם דחיפת־הקדם־זרימה הכללי. שורה 2 מאתחלת את הרשימה L כך שתכיל את כל הקדקודים שייתכן כי יגלשו, בסדר כלשהו. שורות 3-4 מאתחלות את המצביע ***current*** של כל קדקוד u כך שיצביע על הקדקוד הראשון ברשימת השכנים של u.

כמתואר באיור 27.11, לולאת ה־**while** בשורות 6-11 סורקת את הרשימה L ופורקת קדקודים. שורה 5 גורמת לכך שהסריקה תתחיל בקדקוד הראשון ברשימה. בכל איטרציה של הלולאה, קדקוד u נפרק בשורה 8. אם השגרה DISCHARGE הרימה את u, אזי שורה 10 מקדמת אותו לראש הרשימה L. הבדיקה אם u הורם או לא מתבצעת על־ידי כך ששומרים את גובהו של u לפני פעולת הפריקה במשתנה ***old-height*** (שורה 7), ומשווים גובה זה לגובהו של u לאחר הפריקה (שורה 9). שורה 11 גורמת לכך שהאיטרציה הבאה של לולאת ה־**while** תופעל על הקדקוד העוקב ל־u ברשימה L. אם u הועבר לראש הרשימה, האיטרציה הבאה פועלת על הקדקוד העוקב ל־u במיקומו החדש ברשימה.



איור 27.11 פעולת השגרה LIFT-TO-FRONT. **(i)** רשת זרימה ממש לפני האיטרציה הראשונה של לולאת ה־**while**. במצב ההתחלתי, 26 יחידות זרימה יוצאות מן המקור s. מימין מוצגת הרשימה ההתחלתית $L = \langle x, y, z \rangle$, כאשר במצב ההתחלתי $u = x$. מתחת לכל קדקוד ברשימה L מופיעה רשימת השכנים שלו, כאשר הקדקוד הנוכחי מוצלל. הקדקוד x נפרק. הוא מורם לגובה 1, 5 יחידות זרימה עודפת נדחפות אל y, ו־7 היחידות הנותרות של זרימה עודפת נדחפות לבור t. מכיוון ש־x מורם, הוא מועבר לראש הרשימה L, העברה אשר במקרה זה אינה משנה את המבנה של L. **(ii)** לאחר x, הקדקוד הבא ברשימה L שנפרק הוא y, אינד 27.10 מתאר בפירוט את פעולת הפריקה של y במצב זה. מכיוון ש־y מורם, הוא מועבר לראש הרשימה L. **(iii)** הקדקוד x הוא עתה הקדקוד העוקב של y ב־L, ולכן הוא נפרק שוב, וכל 5 יחידות הזרימה העודפת שלו נדחפות ל־t. מכיוון שהקדקוד x אינו מורם במהלך פעולת פריקה זו, הוא נותר במקומו ברשימה L. **(iv)** מכיוון שהקדקוד z עוקב ל־x ב־L, הוא נפרק. הוא מורם לגובה 1, וכל 8 יחידות הזרימה העודפת שלו נדחפות ל־t. מכיוון ש־z מורם, הוא מועבר לראש הרשימה L. **(v)** הקדקוד y הוא עתה העוקב של z ב־L, ולפיכך הוא נפרק. אולם מכיוון ש־y אינו מכיל כל זרימה עודפת, DISCHARGE חוזרת מיד, ו־y נותר במקומו ב־L. לאחר מכן נפרק הקדקוד x. מכיוון שגם הוא אינו מכיל כל זרימה עודפת, DISCHARGE שוב חוזרת מיד, ו־x נותר במקומו ב־L. LIFT-TO-FRONT כבר הניעה לסוף הרשימה L ועוצרת. אין עוד קדקודים גולשים והקדם־זרימה היא זרימה מקסימלית.



איור 27.11 (המשך)

כדי להראות כי השגרה LIFT-TO-FRONT מחשבת זרימה מקסימלית, נראה כי היא מימוש של אלגוריתם דחיפת־הקדם־זרימה הכללי. תחילה נשים לב כי השגרה מבצעת פעולות דחיפה והרמה רק כאשר אכן ניתן לבצען, שכן למה 27.29 מבטיחה כי DISCHARGE מבצעת את הפעולות הללו רק כאשר ניתן לעשות זאת. נותר להראות שכאשר LIFT-TO-FRONT עוצרת, לא ניתן לבצע שום פעולה בסיסית. נשים לב לכך שאם u מגיע לסוף L, כל קדקוד ב־L (פרט אולי לראשון, שאינו מכיל זרימה עודפת) בהכרח נפרק בלי לגרום להרמה. למה 27.30, שאותה נוכיח מיד, קובעת שהרשימה L מתוחזקת כרשימה ממוינת טופולוגית של הרשת המותרת. לכן, פעולת דחיפה גורמת לכך שזרימה עודפת נעה אל קדקודים שבהמשך הרשימה (או ל־s או t). לפיכך, אם המצביע u מגיע לסוף הרשימה, הרי שהזרימה העודפת של כל קדקוד היא 0, ולא ניתן לבצע שום פעולה בסיסית.

למה 27.30

אם מריצים את LIFT-TO-FRONT על רשת זרימה $G = (V, E)$ עם מקור s ובור t, אזי כל איטרציה של לולאת ה־**while** שבשורות 6-11 משמרת את האינווריאנטה כי הרשימה L היא רשימה ממוינת טופולוגית של הקדקודים ברשת המותרת $G_{f,h} = (V, E_{f,h})$.

הוכחה מיד לאחר הרצת INITIALIZE-PREFLOW, $h[s] = |V|$, ו־$h[v] = 0$ עבור כל $v \in V - \{s\}$. מכיוון ש־$|V| \geq 2$ (שכן V מכילה לפחות את s ואת t), לא ייתכנו במצב זה קשתות מותרות. לכן, $E_{f,h} = \phi$, וכל סידור של $V - \{s, t\}$ הוא מיון טופולוגי של $G_{f,h}$.

עתה נראה שהאינווריאנטה משתמרת בכל איטרציה של לולאת ה־**while**. הרשת המותרת משתנה רק על־ידי פעולות דחיפה והרמה. על פי למה 27.27, פעולות דחיפה יכולות רק להפוך קשתות לבלתי־מותרות. לכן, קשתות מותרות יכולות להיווצר רק על־ידי פעולות הרמה. אבל, על פי למה 27.28, לאחר הרמתו של קדקוד u, לא קיימות קשתות מותרות הנכנסות אל u אבל עשויות להיות קשתות מותרות היוצאות ממנו. לכן, על־ידי העברת u לראש הרשימה L, האלגוריתם מבטיח שכל קשת מותרת היוצאת מ־u תקיים את סדר המיון הטופולוגי. ■

ניתוח

עתה נראה כי LIFT-TO-FRONT רצה בזמן $O(V^3)$ על רשת זרימה $G = (V, E)$ כלשהי. מכיוון שאלגוריתם הרם־וקדם זה הוא מימוש של אלגוריתם דחיפת־הקדם־זרימה הכללי, נעשה שימוש במסקנה 27.21 המציבה חסם של $O(V)$ על מספר פעולות ההרמה המבוצעות על כל קדקוד וחסם של $O(V^2)$ על המספר הכולל של פעולות הרמה. בנוסף, תרגיל 27.4-2 מציב חסם של $O(VE)$ על הזמן הכולל המושקע בביצוע פעולות הרמה, ולמה 27.22 מציבה חסם של $O(VE)$ על המספר הכולל של דחיפות מרוות.

משפט 27.31

זמן הריצה של LIFT-TO-FRONT על רשת זרימה $G = (V, E)$ כלשהי הוא $O(V^3)$.

הוכחה הבה נכנה בשם "שלב" של האלגוריתם LIFT-TO-FRONT את הזמן בין שתי פעולות הרמה עוקבות. מספר השלבים הוא $O(V^2)$, שכן ישנן $O(V^2)$ פעולות הרמה. בכל שלב מתבצעות לכל היותר $|V|$ קריאות ל־DISCHARGE, דבר שניתן לראותו כדלקמן: אם DISCHARGE אינה מבצעת פעולת הרמה, אזי הקריאה הבאה ל־DISCHARGE היא על קדקוד שנמצא בהמשך הרשימה L, ואורכה של L קטן מ־$|V|$. אם DISCHARGE מבצעת פעולת הרמה, הקריאה הבאה ל־DISCHARGE כבר שייכת לשלב אחר. מאחר שכל שלב מכיל לכל היותר $|V|$ קריאות ל־DISCHARGE ומספר השלבים הוא $O(V^2)$, הרי שמספר הפעמים ש־DISCHARGE נקראת בשורה 8 של LIFT-TO-FRONT הוא $O(V^3)$. לכן, העבודה הכוללת שמבצעת לולאת ה־**while** של LIFT-TO-FRONT, לא כולל העבודה המבוצעת בתוך DISCHARGE, היא לכל היותר $O(V^3)$.

עתה עלינו לחסום את העבודה המבוצעת בתוך DISCHARGE במהלך ריצתו של האלגוריתם. כל איטרציה של לולאת ה־**while** בתוך DISCHARGE מבצעת אחת משלוש פעולות. אנו ננתח את כמות העבודה הכוללת המכרוכה בביצוע כל אחת מפעולות אלה.

נתחיל בפעולות הרמה (שורות 4-5). תרגיל 27.4-2 מספק חסם זמן של $O(VE)$ על כל $O(V^2)$ פעולות ההרמה המבוצעות.

עתה, נניח שהפעולה מעדכנת את המצביע $current[u]$ בשורה 8. פעולה זו מתרחשת $O(\text{degree}(u))$ פעמים בכל פעם שמתבצעת הרמה של קדקוד u, ובסך הכל $O(V \cdot \text{degree}(u))$. פעמים עבור הקדקוד. לפיכך, עבור כל הקדקודים, כמות העבודה הכוללת המושקעת בקידום מצביעים לאורכן של רשימות שכנים הוא $O(VE)$, על פי למת לחיצות הידיים (תרגיל 1-5.4).

הסוג השלישי של פעולות שמבצעת DISCHARGE הוא פעולות דחיפה (שורה 7). אנו כבר יודעים כי המספר הכולל של דחיפות מרוות הוא $O(VE)$. נשים לב שאם מתבצעת דחיפה לא־מרווה, DISCHARGE חוזרת מיד, שכן הדחיפה מקטינה ל־0 את הזרימה העודפת. לכן, יכולה להיות לכל היותר דחיפה לא־מרווה אחת לכל קריאה ל־DISCHARGE. כפי שכבר ראינו, DISCHARGE נקראת $O(V^3)$ פעמים, ולכן הזמן הכולל המושקע בביצוע דחיפות לא־מרוות הוא $O(V^3)$.

זמן הריצה של LIFT-TO-FRONT הוא אפוא $O(V^3 + VE)$, שהוא $O(V^3)$. ■

תרגילים

27.5-1

הדגם הרצה של LIFT-TO-FRONT, על פי דוגמת איור 27.11, על רשת הזרימה מאיור 27.1(i). הנח שהסידור ההתחלתי של הקדקודים ב־L הוא $\langle v_1, v_2, v_3, v_4 \rangle$ ושרשימות השכנים הן:

$$N[v_1] = \langle s, v_2, v_3 \rangle$$
$$N[v_2] = \langle s, v_1, v_3, v_4 \rangle$$
$$N[v_3] = \langle v_1, v_2, v_4, t \rangle$$
$$N[v_4] = \langle v_2, v_3, t \rangle$$

* 27.5-2

ברצוננו לממש אלגוריתם דחיפת־קדם־זרימה שבו אנו מתחזקים תור נבנס־ראשון־יוצא־ראשון של קדקודים גולשים. האלגוריתם פורק שוב ושוב את הקדקוד שבראש התור, ומוסיף לסוף התור כל קדקוד שלא גלש לפני הפריקה אולם גולש אחריה. לאחר שפורקים את הקדקוד שבראש התור, הקדקוד מוסר מן התור. כאשר התור ריק, האלגוריתם עוצר. הראה שניתן לממש אלגוריתם זה כך שיחשב זרימה מקסימלית בזמן $O(V^3)$.

27.5-3

הראה שהאלגוריתם הכללי פועל כהלכה גם אם LIFT מעדכנת את $h[u]$ פשוט על־ידי ההצבה $h[u] \leftarrow h[u] + 1$. כיצד משפיע שינוי זה על הניתוח של LIFT-TO-FRONT?

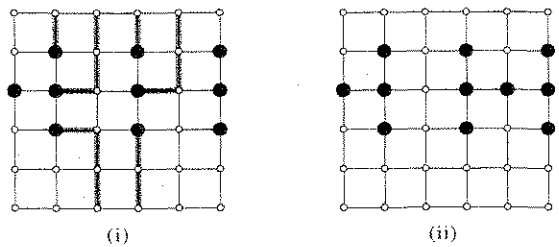
* 27.5-4

הראה שאם פורקים תמיד את הקדקוד הגולש הגבוה ביותר, שיטת דחיפת־הקדם־זרימה ניתנת למימוש בזמן $O(V^3)$.

בעיות

27-1 בעיית דרכי המילוט

שריג (grid) שממדיו $n \times n$ הוא גרף בלתי־מכוון המורכב מ־n שורות ו־n עמודות של קדקודים, כמתואר באיור 27.12. את הקדקוד בשורה ה־i ובעמודה ה־j אנו מסמנים על־ידי (i, j). לכל הקדקודים בשריג יש בדיוק ארבעה שכנים, פרט לקדקודי השפה, שהם הנקודות (i, j) שעבורן $i = 1$ או $i = n$ או $j = 1$ או $j = n$.



איור 27.12 שריגים עבור בעיית דרכי המילוט. נקודות ההתחלה הן שחורות והקדקודים האחרים בשריג לבנים. **(i)** שריג עם דרכי מילוט, המוצגות על־ידי המסלולים המוצללים. **(ii)** שריג ללא דרכי מילוט.

בהינתן $m \le n^2$ נקודות התחלה בשריג, $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$, **בבעיית דרכי המילוט** (escape problem) יש לקבוע אם קיימים בשריג m מסלולים זרים־בקדקודיהם מנקודות ההתחלה ל־m נקודות שונות כלשהן על שפת השריג. לדוגמה, בשריג שבאיור 27.12(i) ישנן דרכי מילוט אולם בשריג שבאיור 27.12(ii) אין כאלה.

א. נתבונן ברשת זרימה אשר בה גם הקדקודים, ולא רק הקשתות, הם בעלי קיבולים. כלומר, הזרימה נטו החיובית הנכנסת לקדקוד נתון כפופה לאילוץ הקיבול של קדקוד זה. הראה שמציאת זרימה מקסימלית ברשת עם קשתות וקדקודים בעלי קיבול ניתנת לרדוקציה לבעיית זרימה מקסימלית רגילה על רשת זרימה מאותו סדר גודל.

ב. תאר אלגוריתם יעיל לפתרון בעיית דרכי המילוט ונתח את זמן הריצה שלו.

27-2 כיסוי מינימלי על־ידי מסלולים

כיסוי על־ידי מסלולים (path cover) של גרף מכוון $G = (V, E)$ הוא קבוצה P של מסלולים זרים־בקדקודיהם, כך שכל קדקוד ב־V נכלל בדיוק במסלול אחד ב־P. מסלולים יכולים להתחיל ולהסתיים בכל נקודה שהיא, ולהיות בעלי אורך כלשהו, כולל 0. **כיסוי מינימלי על־ידי מסלולים** (minimum path cover) של G הוא כיסוי על־ידי מסלולים המכיל את מספר המסלולים הקטן ביותר האפשרי.

א. כתוב אלגוריתם יעיל למציאת כיסוי מינימלי על־ידי מסלולים של גרף מכוון ללא מעגלים $G = (V, E)$. (**רמז:** בהנחה כי $V = \{1, 2, \ldots, n\}$, בנה את הגרף $G' = (V', E')$, כאשר:

$$V' = \{x_0, x_1, \ldots, x_n\} \cup \{y_0, y_1, \ldots, y_n\}$$
$$E' = \{(x_0, x_i) : i \in V\} \cup \{(y_i, y_0) : i \in V\} \cup \{(x_i, y_j) : (i, j) \in E\}$$

והרץ אלגוריתם זרימה מקסימלית.)

ב. האם האלגוריתם שלך פועל כהלכה כאשר הגרף המכוון מכיל מעגלים? הסבר.

27-3 ניסויים במעבדת חלל

פרופסור ספוק משמש כיועץ לנאס"א, המתכננת סדרת טיסות של מעבורת חלל וצריכה להחליט אילו ניסויים מסחריים לבצע ואילו מכשירים לקחת בכל טיסה. עבור כל טיסה, נאס"א בוחנת סדרה $E = \langle E_1, E_2, \ldots, E_m \rangle$ של ניסויים, כאשר הממון המסחרי של ניסוי E_j הסכים לשלם לנאס"א p_j דולרים עבור תוצאות הניסוי. הניסויים עושים שימוש בקבוצה $I = \langle I_1, I_2, \ldots, I_n \rangle$ של מכשירים; לכל ניסוי E_j דרושים כל המכשירים בתת־קבוצה $R_j \subseteq I$. עלות נשיאת המכשיר I_k היא c_k דולרים. משימתו של הפרופסור היא למצוא אלגוריתם יעיל שיקבע אילו ניסויים לבצע ואילו מכשירים לקחת עבור טיסה נתונה במטרה להגדיל למקסימום את ההכנסה נטו, שהיא ההכנסה מכל הניסויים המבוצעים פחות ההוצאות על כל המכשירים הנלקחים לטיסות.

נתונה הרשת G הבאה: הרשת מכילה קדקוד מקור s, קדקודים $I_1, I_2, \ldots, I_n$, קדקודים $E_1, E_2, \ldots, E_m$ וקדקוד בור t. עבור $k = 1, 2, \ldots, n$, קיימת קשת (s, I_k) בעלת קיבול c_k, ועבור $k = 1, 2, \ldots, n$ קיימת קשת (E_j, t) בעלת קיבול p_j. עבור $j = 1, 2, \ldots, m$ ו־$k = 1, 2, \ldots, n$, אם $I_k \in R_j$, אזי קיימת קשת (I_k, E_j) בעלת קיבול אינסופי.

א. הראה שאם $E_j \in T$ עבור חתך (S, T) של G בעל קיבול סופי, אזי $I_k \in T$ עבור כל $I_k \in R_j$.

ב. הראה כיצד ניתן לחשב את ההכנסה נטו המקסימלית מתוך הקיבול של החתך המינימלי של G והערכים p_j הנתונים.

ג. כתוב אלגוריתם יעיל הקובע אילו ניסויים לבצע ואילו מכשירים לקחת. נתח את זמן הריצה של האלגוריתם שלך במונחי m, n ו־$r = \sum_{j=1}^{m} |R_j|$.

27-4 עדכון זרימה מקסימלית

תהי $G = (V, E)$ רשת זרימה עם מקור s, בור t, וקיבולים בעלי ערכים שלמים. נניח שנתונה לנו זרימה מקסימלית ב־G.

א. נניח שמגדילים ב־1 את קיבולה של קשת יחידה $(u, v) \in E$. כתוב אלגוריתם שזמן ריצתו $O(V + E)$ אשר מעדכן את הזרימה המקסימלית.

ב. נניח שמקטינים ב־1 את קיבולה של קשת יחידה $(u, v) \in E$. כתוב אלגוריתם שזמן ריצתו $O(V + E)$ אשר מעדכן את הזרימה המקסימלית.

27-5 זרימה מקסימלית באמצעות כיול

תהי $G = (V, E)$ רשת זרימה עם מקור s, בור t, וקיבול $c(u, v)$ בעל ערך שלם על כל קשת $(u, v) \in E$. יהי $C = \max_{(u, v) \in E} c(u, v)$.

א. הראה כי קיבולו של חתך מינימלי ב־G הוא לכל היותר $C|E|$.

ב. בהינתן מספר K, הראה כי ניתן למצוא בזמן $O(E)$ מסלול שיפור שקיבולו לפחות K, אם קיים מסלול כזה.

באמצעות הגרסה הבאה של השגרה FORD-FULKERSON-METHOD ניתן לחשב זרימה מקסימלית ב־G.

```
MAX-FLOW-BY-SCALING(G, s, t)
1  C ← max_{(u,v)∈E} c(u, v)
2  initialize flow f to 0
3  K ← 2^⌊lg C⌋
4  while K ≥ 1
5      do while there exists an augmenting path p of capacity at least K
6             do augment flow f along p
7      K ← K/2
8  return f
```

ג. הראה כי השגרה MAX-FLOW-BY-SCALING מחזירה זרימה מקסימלית.

ד. הראה שבכל פעם שמבוצעת שורה 4, קיבולו של חתך מינימלי בגרף השיורי G_f הוא לכל היותר $2K|E|$.

ה. הראה כי לולאת ה־**while** הפנימית בשורות 5-6 מבוצעת $O(E)$ פעמים עבור כל ערך של K.

ו. הסק כי ניתן לממש את השגרה MAX-FLOW-BY-SCALING כך שתרוץ בזמן $O(E^2 \lg C)$.

27-6 זרימה מקסימלית עם קיבולים שליליים

נניח שאנו מרשים כי ברשת זרימה יהיו קיבולי קשתות שליליים. ברשת כזאת לא בהכרח קיימת זרימה אפשרית.

א. נתונה קשת (u, v) בעלת קיבול $c(u, v) < 0$ ברשת זרימה $G = (V, E)$. הסבר בקצרה מה משמעותה של קשת כזאת בעלת קיבול שלילי, במונחים של הזרימה החיובית בין u ו־v.

תהי $G = (V, E)$ רשת זרימה עם קיבולים שליליים, ויהיו s ו־t המקור והבור של G. בנה את רשת הזרימה הרגילה $G' = (V, E')$ עם פונקציית קיבול c', מקור s' ובור t', כאשר:

$$
\begin{aligned}
V' &= V \cup \{s', t'\} \\
E' &= E \cup \{(u, v) : (v, u) \in E\} \\
&\quad \cup \{(s', v) : v \in V\} \\
&\quad \cup \{(u, t') : u \in V\} \\
&\quad \cup \{(s, t), (s, t)\}
\end{aligned}
$$

את קיבולי הקשתות אנו קובעים באופן הבא: עבור כל קשת $(u, v) \in E$, אנו מציבים $c'(u, v) = c'(v, u) = (c(u, v) + c(v, u))/2$. עבור כל קדקוד $u \in V$, אנו מציבים:

$$c'(s', u) = \max(0, (c(V, u) - c(u, V))/2)$$

$$c'(u, t') = \max(0, (c(u, V) - c(V, u))/2)$$

כמו כן אנו מציבים $c'(s, t) = c'(t, s) = \infty$.

ב. הוכח שאם קיימת ב־G זרימה אפשרית, אזי כל הקיבולים ב־G' הם אי־שליליים וקיימת ב־G' זרימה מקסימלית שבה כל הקשתות הנכנסות לבור t' רוויות.

ג. הוכח את הטענה ההפוכה לסעיף ב. על הוכחתך להיות הוכחה בונה, דהיינו, בהינתן זרימה ב־G' המרווה את כל הקשתות הנכנסות אל t', על הוכחתך להראות כיצד לבנות זרימה אפשרית ב־G.

ד. תאר אלגוריתם המוצא זרימה אפשרית מקסימלית ב־G. נסמן ב־$MF(n, m)$ את זמן הריצה במקרה הגרוע של אלגוריתם רגיל למציאת זרימה מקסימלית על גרף בעל n קדקודים ו־m קשתות. נתח את האלגוריתם שלך לחישוב זרימה מקסימלית ברשת זרימה עם קיבולים שליליים במונחי MF.

הערות ביבליוגרפיות לפרק 27

Even [65], Lawler [132], Papadimitriou & Steiglitz [154] ו־Tarjan [188] הם מקורות טובים לרשתות זרימה ולאלגוריתמים קרובים. Goldberg, Tardos & Tarjan [83] מציגים סקירה נאה של אלגוריתמים עבור בעיות של רשתות זרימה.

שיטת פורד־פולקרסון היא של Ford & Fulkerson [71], אשר ניסחו לראשונה רבות מן הבעיות בתחום רשתות הזרימה, כולל בעיית הזרימה המקסימלית ובעיית הזיווג בגרף דו־צדדי. רבים מן המימושים המוקדמים של שיטת פורד־פולקרסון מצאו מסלולי שיפור באמצעות חיפוש־לרוחב; Edmonds & Karp [63] הוכיחו שאסטרטגיה זו מניבה אלגוריתם זמן־פולינומיאלי. Karzanov [119] פיתח את רעיון הקדם־זרימה. שיטת דחיפת־הקדם־זרימה היא של Goldberg [82]. האלגוריתם דחיפת־קדם־זרימה המהיר ביותר המוכר כיום הוא של Goldberg & Tarjan [85], המשיגים זמן ריצה של $O(VE \lg(V^2/E))$. האלגוריתם הטוב ביותר המוכר כיום למציאת זיווג מקסימלי בגרף דו־צדדי, שהתגלה על ידי Hopcroft & Karp [101], רץ בזמן $O(\sqrt{V}E)$.

VII נושאים נבחרים

מקבילי.

בפרק 32 נחקור פעולות על פולינומים ונראה כי באמצעות שיטה עבור אחת ידועה — טרנספורם פוריה המהיר (the fast Fourier transform או FFT) — ניתן לכפול שני פולינומים ממעלה n בזמן $O(n \lg n)$. בפרק 33 נחקור אלגוריתמים יעילים על מספרי שלמים, כגון כפל ופירוק לגורמים ראשוניים. בפרק 34 נראה כי ניתן לקבוע אם מחרוזת טקסט נתונה מופיעה בתוך מחרוזת אחרת בזמן לינארי ביחס לאורכי המחרוזות. בפרק 35 נבחן בעיה של מציאת הנקודה הקרובה ביותר בקבוצה של n נקודות במישור, ונראה כי ניתן למצוא תשובה זו בזמן $O(n \lg n)$.

בפרק 36 נראה כיצד ניתן לחשב את כל שורשי המשוואה המרוכבת n מהצורה $1 = x^n$ בזמן $O(n \lg n)$. נראה גם כיצד ניתן לחשב את כל שורשי המשוואה בזמן $O(n^{2.81})$ באמצעות אלגוריתם של שטראסן (Strassen). בפרק 37 נחקור אלגוריתמים יעילים על מטריצות גדולות דלילות. נראה כיצד ניתן לכפול שתי מטריצות $n \times n$ בזמן $O(n^3)$ — פחות מהזמן הנדרש לכפל מלא של מטריצות — ונראה כיצד ניתן לפתור מערכת משוואות לינאריות $Ax = b$ בזמן $O(n^3)$ עבור מטריצה A דלילה בעלת n שורות ועמודות. נראה כיצד ניתן לפתור מערכת משוואות לינאריות בזמן $O(n^{2.81})$ עבור מטריצה A בעלת n שורות ועמודות.

בפרק 38 נציג מודל בעל חישוב מקבילי המכונה PRAM, וכן מספר שיטות מקבליות בסיסיות: הקפצת מצביעים (pointer jumping), חישוב רישות (prefix computation) ושיטת שבירת הסימטריה (symmetry breaking). נראה כיצד ניתן לשלב שיטות אלו כדי לבנות אלגוריתם דטרמיניסטי, מהיר ויעיל למיון רשימה מקושרת בת n איברים. נראה גם אלגוריתם מקבילי יעיל למיון רשימות (list ranking) ואלגוריתם דטרמיניסטי, מהיר ויעיל בהרבה, לחישוב פונקציית הרישות בעץ שורשי.

בפרק 39 נציג מודל חישוב מקבילי שונה, המכונה רשת השוואה (comparison network). נראה כיצד ניתן לבנות רשת השוואה בגודל $O(\lg n)$ למיזוג שני מערכים ממוינים בני n מספרים כל אחד. נראה גם כיצד ניתן לבנות רשת השוואה בגודל $O(n \lg n)$ למיון n מספרים. נראה גם כיצד ניתן לבנות מחסם (adder) מסוג חיבור עם נשיאה מקדימה (carry-lookahead adder) בגודל $O(\lg n)$ המחבר שני מספרים בני n סיביות.

בפרק 40 נציג מודל חישוב מקבילי שלישי: מעגל בוליאני. נראה כיצד ניתן לבנות רשת השוואה בגודל $O(\lg^2 n)$ מספרים בזמן $O(\lg^2 n)$.

סיכום

ישנן דרכים רבות לפתור בעיה חישובית. רוב הספרים מתמקדים בשיטות מסוימות, אך לא בכולן. שיטה אחת שבה נתמקד היא שיטת הפרד ומשול, שבה נפרק בעיה לתת-בעיות, נפתור כל תת-בעיה באופן רקורסיבי ולבסוף נשלב את הפתרונות כדי לקבל פתרון לבעיה המקורית. נלמד גם שיטות נוספות, כגון תכנון דינמי ואלגוריתמים חמדניים. הבנת השיטות השונות לפתרון בעיות תאפשר לכם לפתח אלגוריתמים טובים בעצמכם, ותסייע לכם לזהות אלגוריתמים טובים המוצעים על ידי אחרים. לעתים קרובות ישנם אלגוריתמים רבים היעילים באופן אסימפטוטי, אך ביצועיהם בפועל תלויים מאוד בקבועים ובגודל הקלט שאתם מתמודדים עמו. ייתכן שאלגוריתם אחד יהיה היעיל ביותר עבור קלט קטן, אלגוריתם אחר יהיה היעיל ביותר עבור קלט בגודל בינוני, ואלגוריתם שלישי יהיה היעיל ביותר עבור קלט גדול. כדי לבחור את האלגוריתם הנכון, עליכם לקבוע את גודל הקלט שאתם מתמודדים עמו, ולהעריך את ביצועי האלגוריתמים השונים עבור גודל קלט זה.

בפרק 33 נציג אלגוריתמים בתורת המספרים. לאחר חזרה על יסודות תורת המספרים, נציג את האלגוריתם של אוקלידס (Euclid) לחישוב מחלק משותף מקסימלי. אחר־כך נציג אלגוריתמים לפתרון משוואות לינאריות מודולריות ולהעלאת מספר אחד בחזקה מודולו מספר אחר. בהמשך נציג יישום מעניין של אלגוריתמי תורת המספרים: שיטת ההצפנה RSA המבוססת על מפתח ציבורי. לא זו בלבד שבאמצעות מערכת הצפנה זו ניתן להצפין הודעות כך שיריב לא יוכל לקרוא אותן, אלא שניתן להשתמש בה גם לקבלת חתימות ספרתיות. לאחר מכן נציג את המבחן האקראי של מילר (Miller) ורבין (Rabin), למספרים ראשוניים, אשר באמצעותו ניתן למצוא ביעילות מספרים ראשוניים גדולים — דרישה חיונית בשיטת RSA. לבסוף, נסקור את היוריסטיקת "רו" של פולארד (Pollard) עבור פירוק לגורמים של מספר שלמים, ונדון במצב הידע כיום בתחום הפירוק לגורמים של מספרים שלמים.

בפרק 34 נחקור את הבעיה של מציאת כל המופעים של תבנית נתונה במחרוזת טקסט נתונה, בעיה המתעוררת לעתים תכופות בתכניות עיבוד תמלילים. תחילה, נבחן את גישתם האלגנטית של רבין (Rabin) וקארפ (Karp). בהמשך, לאחר שנבחן פתרון יעיל המבוסס על אוטומטים סופיים, נציג את אלגוריתם קנוט־מוריס־פראט (Knuth-Morris-Pratt), אשר מצליח להשיג יעילות גבוהה על־ידי עיבוד מקדים מחוכם של התבנית. נסיים את הפרק בהצגת ההיוריסטיקה של בויאר ומור (Boyer & Moore) להשוואת מחרוזות.

גיאומטריה חישובית היא נושאו של פרק 35. לאחר שנדון בפרימיטיבים הבסיסיים של הגיאומטריה החישובית, נראה כיצד באמצעות שיטת סריקה, ניתן לקבוע ביעילות אם קבוצת קטעים ישרים מכילה חיתוכים כלשהם. שני אלגוריתמים חכמים למציאת הקמור של קבוצת נקודות — **הסריקה** של גראהאם (Graham) ו**הצעדה** של גיארביס (Jarvis) — מדגימים גם הם את כוחן של שיטות סחיפה. נסיים את הפרק בהצגת אלגוריתם יעיל למציאת הזוג הקרוב ביותר מבין קבוצה נתונה של נקודות במישור.

בפרק 36 נעסוק בבעיות שלמות ב־NP, או, בקצרה, בעיות NP־שלמות. בעיות חישוביות מעניינות רבות הן NP־שלמות, אולם כיום לא ידוע אלגוריתם זמן־פולינומיאלי לפתרון אף אחת מהן. בפרק זה נציג שיטות שבאמצעותן ניתן לקבוע אם בעיה היא NP־שלמה או לא. עבור מספר בעיות קלאסיות קיימות הוכחות להיותן NP־שלמות: האם גרף מכיל מעגל המילטוני, האם ניתן לספק נוסחה בוליאנית נתונה, האם קבוצה נתונה של מספרים מכילה תת־קבוצה שסכום איבריה הוא ערך מטרה נתון. בפרק זה גם נוכיח שבעיית הסוכן־הנוסע המפורסמת היא שלמה ב־NP.

בפרק 37 נראה כיצד ניתן להשתמש באלגוריתמי קירוב כדי למצוא ביעילות פתרונות מקורבים לבעיות NP־שלמות. קיימות בעיות NP־שלמות שעבורן קל למדי למצוא פתרונות המתקרבים לפתרון האופטימלי, אולם עבור בעיות אחרות, גם אלגוריתמי הקירוב הטובים ביותר המוכרים כיום, ביצועיהם הולכים ונעשים גרועים ככל שגודל הבעיה גדל. לעומתן, קיימות בעיות שעבורן ככל שמשקיעים זמן חישוב רב יותר מתקבלים פתרונות מקורבים טובים יותר. בפרק זה נדגים אפשרויות אלה בעזרת בעיית כיסוי הצמתים, בעיית הסוכן־הנוסע, בעיית כיסוי הקבוצות ובעיית סכום האיברים בתת־קבוצה.

28 רשתות מיון

בחלק II של הספר בחנו אלגוריתמי מיון עבור מחשבים סדרתיים (המכונים גם מכונות גישה־אקראית (random access machines) או בקיצור RAM's). על מחשבים מסוג זה ניתן לבצע בכל פעם פעולה אחת בלבד. בפרק זה נחקור אלגוריתמי מיון המבוססים על המודל החישובי של רשתות השוואה, אשר בו ניתן לבצע בו בזמן פעולות השוואה רבות.

רשתות השוואה נבדלות ממכונות גישה־אקראית בשני היבטים חשובים. ראשית, הן מסוגלות לבצע אך ורק השוואות. לכן, אלגוריתם כמו מיון מנייה (ראה סעיף 9.2) לא ניתן למימוש על רשת השוואה. שנית, בניגוד למודל RAM שבו פעולות מבוצעות באופן סדרתי — כלומר, בזו אחר זו — פעולות ברשת השוואה עשויות להתבצע בו־בזמן או "במקביל". כפי שנראה, תכונה זו מאפשרת לבנות רשתות השוואה עשויות להתבצע בו־בזמן או "במקביל". כפי שנראה, תכונה זו מאפשרת לבנות רשתות השוואה המממיינות n ערכים בזמן תת־לינארי.

נתחיל, בסעיף 28.1, בהגדרת רשתות השוואה ורשתות מיון. נביא הגדרה טבעית של "זמן הריצה" של רשת השוואה במונחי עומק הרשת. בסעיף 28.2 נוכיח את "עקרון ה־0-1", אשר מקל מאוד את ניתוח נכונותן של רשתות מיון.

רשת המיון היעילה שנבנה היא במהותה גרסה מקבילית של האלגוריתם מיון־מיזוג מסעיף 1.3.1. הבנייה תכיל שלושה שלבים: בסעיף 28.3 נציג את מבנהו של ממיין "ביטוני" שיהיה אבן הבניין הבסיסית שלנו; בסעיף 28.4 נשנה במעט את הממיין הביטוני כדי לקבל רשת מיזוג שיכולה למזג שני סדרות ממוינות לסדרה ממוינת אחת. לבסוף, בסעיף 28.5, נרכיב את רשתות המיזוג הללו לרשת מיון היכולה למיין n ערכים בזמן $O(\lg^2 n)$.

28.1 רשתות השוואה

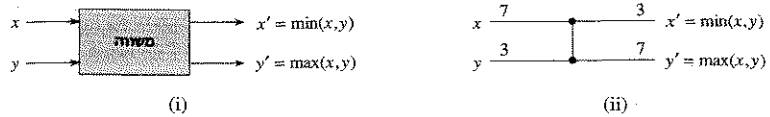
רשתות מיון הן רשתות השוואה אשר ממיינות תמיד את הקלטים שלהן, ולכן הגיוני לפתוח את הדיון ברשתות השוואה ובתכונותיהן. רשת השוואה מורכבת אך ורק מתילים וממשווים. **משווה** (comparator), המתואר באיור 28.1(i), הוא התקן שנכנסים אליו שני קלטים, x ו־y, ויוצאים ממנו שני פלטים, x' ו־y', והוא מבצע את הפונקציה הבאה:

$$x' = \min(x, y)$$

$$y' = \max(x, y)$$

מכיוון שהצגתו החזותית של משווה כמו באיור 28.1(i) גדולה מדי לצרכינו, נאמץ סימון מקובל של משווה כקו מאונך יחיד, כמתואר באיור 28.1(ii). הקלטים מופיעים משמאל לקו והפלטים

מיימינו, כאשר ערך הקלט הקטן יותר מופיע כפלט העליון וערך הקלט הגדול יותר מופיע כפלט התחתון. נוכל אפוא לראות משווה כיחידה הממיינת את שני קלטיה.

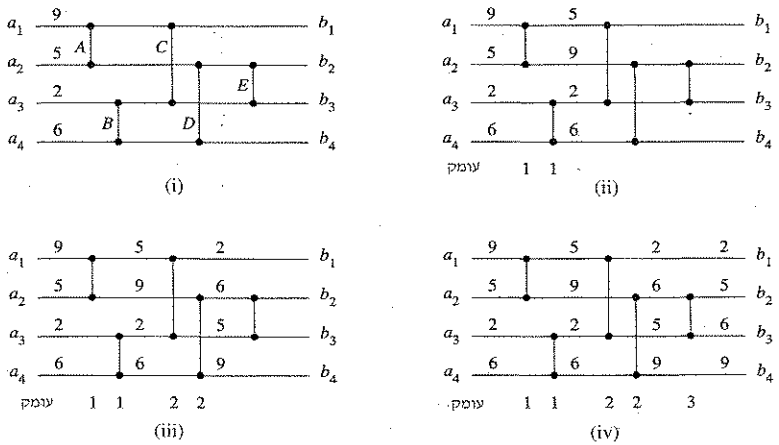


איור 28.1 **(i)** משווה עם קלטים x ו־y ופלטים x' ו־y'. **(ii)** אותו משווה כאשר הוא מוצג כקו מאונך יחיד. באיור מוצגים הקלטים $x = 7$ ו־$y = 3$ והפלטים $x' = 3$ ו־$y' = 7$.

אנו נניח שכל משווה מבצע את פעולתו בזמן $O(1)$. במילים אחרות, אנו מניחים כי הזמן העובר בין הופעתם של ערכי הקלט x ו־y, לבין הפקתם של ערכי הפלט x' ו־y' הוא קבוע.

תיל (wire) מעביר ערך ממקום למקום. ישנם תילים המחברים פלט של משווה אחד לקלט של משווה אחר, אך כל התילים האחרים הם תילי קלט או תילי פלט של הרשת. במהלך פרק זה נניח שרשת השוואה מכילה n **תילי קלט** $a_1, a_2, \ldots, a_n$, שדרכם נכנסים לרשת הערכים שיש למיין, ו־n **תילי פלט** $b_1, b_2, \ldots, b_n$, הפולטים את התוצאות שחושבו על־ידי הרשת. אנו נדבר גם על **סדרת הקלט** $\langle a_1, a_2, \ldots, a_n \rangle$ **וסדרת הפלט** $\langle b_1, b_2, \ldots, b_n \rangle$, כשהכוונה היא לערכים על תילי הקלט והפלט. כלומר, נשתמש באותו שם גם עבור התיל וגם עבור הערך שהוא נושא. כוונתנו תהיה תמיד ברורה מן ההקשר.

באיור 28.2 מוצגת **רשת השוואה** (comparison network) המורכבת מקבוצה של משווים המחוברים ביניהם באמצעות תילים. אנו משרטטים רשת השוואה עבור n קלטים כאוסף של n **קווים** אופקיים שביניהם עוברים קווים מאונכים המציינים משווים. יש לשים לב לכך שקו אינו מייצג תיל יחיד, אלא סדרה של תילים נפרדים המחברים משווים שונים. לדוגמה, הקו האופקי העליון באיור 28.2 מייצג שלושה תילים: תיל הקלט a_1 המתחבר לקלט של המשווה A; תיל המחבר את הפלט העליון של המשווה A לקלט של המשווה C; ותיל פלט b_1 היוצא מן הפלט העליון של המשווה C. כל קלט למשווה מחובר לתיל שהוא אחד מ־n תילי הקלט של הרשת, $a_1, a_2, \ldots, a_n$, או שהוא מחובר לפלט של משווה אחר. באופן דומה, כל פלט של משווה מחובר לתיל שהוא אחד מ־n תילי הפלט של הרשת $b_1, b_2, \ldots, b_n$ או שהוא מחובר לקלט של משווה אחר. הדרישה העיקרית מן החיבורים בין משווים היא, שגרף החיבורים לא יכיל מעגלים: אם מתווים מסלול מפלט של משווה נתון לקלט של משווה שני, לפלט של המשווה השני וממנו לקלט של משווה שלישי, וכן הלאה, אסור שהמסלול שמתווים יעבור פעמיים דרך אותו משווה ובכך יסגור מעגל. לכן אפשר, כמו באיור 28.2, לשרטט רשת השוואה שבה הקלטים לרשת מופיעים משמאל והפלטים מימין: הנתונים עוברים ברשת משמאל לימין.



איור 28.2 **(i)** רשת השוואה בעלת 4 קלטים ו־5 פלטים, המהווה למעשה רשת מיון. בזמן 0, ארבעת ערכי הקלט הרשומים מופיעים על ארבעת תילי הקלט. **(ii)** בזמן 1, הערכים הרשומים מופיעים על הפלטים של המשווים A ו־B, המצאים בעומק 1. **(iii)** בזמן 2, הערכים הרשומים מופיעים על הפלטים של המשווים C ו־D, המצאים בעומק 2. תילי הפלט b_1 ו־b_4 הם עתה בעלי הערכים הסופיים שלהם, אולם תילי הפלט b_2 ו־b_3 – עדיין לא. **(iv)** בזמן 3, הערכים הרשומים מופיעים על הפלטים של המשווה E, המצא בעומק 3. תילי הפלט b_2 ו־b_3 הם עתה בעלי הערכים הסופיים שלהם.

כל משווה מפיק את ערכי הפלט שלו רק כאשר שני ערכי הקלט שלו זמינים לו. באיור 28.2(i), נניח כי הסדרה $\langle 9, 5, 2, 6 \rangle$ מופיעה על תילי הקלט בזמן 0. אזי, בזמן 0, רק המשווים A ו־B הם במצב שכל ערכי הקלט שלהם זמינים. בהנחה שלכל משווה דרושה יחידת זמן אחת לחישוב ערכי הפלט שלו, המשווים A ו־B מפיקים את הפלטים שלהם בזמן 1; הערכים המתקבלים מוצגים באיור 28.2(ii). נשים לב כי המשווים A ו־B מפיקים את ערכי הפלט שלהם באותו זמן, או "במקביל". כעת, בזמן 1, המשווים C ו־D, אבל לא המשווה E, הם במצב שכל ערכי הקלט שלהם זמינים להם. יחידת זמן אחת מאוחר יותר, בזמן 2, C ו־D מפיקים את הפלטים שלהם, כמתואר באיור 28.2(iii). גם המשווים C ו־D פועלים במקביל. הפלט העליון של המשווה C והפלט התחתון של המשווה D מתחברים אל תילי הפלט b_1 ו־b_4, בהתאמה, של רשת ההשוואה, ולכן תילים אלה נושאים את ערכיהם הסופיים בזמן 2. בינתיים, בזמן 2, ערכי הקלט של המשווה E זמינים לו, ואיור 28.3(iv) מראה כי הוא מפיק את ערכי הפלט שלו בזמן 3. ערכים אלה נישאים עתה על תילי הפלט b_2 ו־b_3, וסדרת הפלט $\langle 2, 5, 6, 9 \rangle$ הושלמה.

בהנחה שכל משווה מבצע את פעולתו ביחידת זמן אחת, נוכל להגדיר את "זמן הריצה" של רשת השוואה, דהיינו, הזמן החולף מרגע שתילי הקלט קיבלו את ערכיהם ועד לרגע שכל תילי הפלט קיבלו את ערכיהם שלהם. באופן לא פורמלי, זמן זה הוא המספר הגדול ביותר של משווים שאיבר קלט כלשהו יכול לעבור דרכם בעת תנועתו מתיל קלט לתיל פלט. בניסוח פורמלי יותר, נגדיר את **העומק** (depth) של תיל באופן הבא: תיל קלט לרשת השוואה הוא בעל עומק 0. כעת, אם למשווה ברשת יש שני תילי קלט שעומקיהם d_x ו־d_y, אזי עומקם של תילי הפלט שלו הוא $\max(d_x, d_y) + 1$. מכיוון שרשת השוואה אינה מכילה מסלולים מעגליים של משווים, עומקו של תיל מוגדר היטב, ועומקו של משווה מוגדר כעומק תילי הפלט שלו. באיור 28.2 מוצגים עומקי המשווים. עומקה של רשת השוואה הוא העומק המקסימלי של תילי הפלט או, באופן שקול, העומק המקסימלי של המשווים. לדוגמה, עומקה של רשת ההשוואה מאיור 28.2 הוא 3, שכן עומקו של המשווה E הוא 3. אם כל משווה מפיק את הפלטים שלו לאחר יחידת זמן אחת, ואם

הקלטים לרשת מופיעים על תילי הקלט בזמן 0, אזי משוואה שעומקו d מפיק את הפלטים שלו בזמן d; עומק הרשת שווה אפוא לזמן הדרוש לרשת להפיק ערכים על כל תילי הפלט שלה.

רשת מיון (sorting network) היא רשת השוואה אשר עבור כל סדרת קלט מפיקה סדרת פלט *ממוינת* מונוטונית (דהיינו, $b_1 \le b_2 \le \cdots \le b_n$). כמובן, לא כל רשת השוואה היא רשת מיון, אולם הרשת שבאיור 28.2 היא אכן כזאת. כדי לראות מדוע, נשים לב כי לאחר זמן 1, הערך המינימלי מבין ארבעת ערכי הקלט מתקבל בפלט העליון של המשוואה A או בפלט העליון של המשוואה B. לכן, לאחר זמן 2, הוא חייב להופיע בפלט העליון של המשוואה C. טיעון סימטרי מראה כי לאחר זמן 2, המקסימום מבין ארבעת ערכי הקלט מתקבל בפלט התחתון של המשוואה D. כל שנותר למשוואה E לעשות הוא להבטיח ששני ערכי הביניים יתפסו את מקומם הנכון בסדרת הפלט, ודבר זה אכן קורה בזמן 3.

רשת השוואה דומה לשגרה בכך שהיא מפרטת אילו השוואות יש לבצע ומתי, אולם היא שונה משגרה בכך שה**גודל** שלה, דהיינו, מספר המשוואים שהיא מכילה, תלוי במספר הקלטים והפלטים. משום כך נתאר למעשה "משפחות" של רשתות השוואה. לדוגמה, מטרת פרק זה היא לפתח משפחה SORTER של רשתות מיון יעילות. אנו מציינים רשת נתונה בתוך משפחה על פי שם המשפחה ומספר הקלטים (השווה למספר הפלטים). לדוגמה, רשת מיון בעלת n קלטים ו־n פלטים במשפחה SORTER נקראת $\text{SORTER}[n]$.

תרגילים

28.1-1
הצג את הערכים המופיעים על כל תילי הרשת מאיור 28.2 כאשר סדרת הקלט היא $\langle 9, 6, 5, 2 \rangle$.

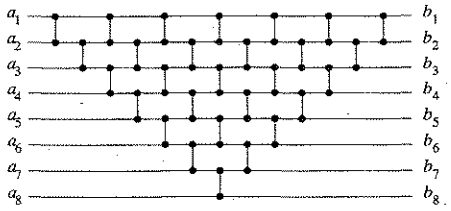
28.1-2
נניח ש־n הוא חזקה מדויקת של 2. הראה כיצד ניתן לבנות רשת השוואה בעלת n קלטים ו־n פלטים שעומקה $\lg n$, אשר בה תיל הפלט העליון נושא תמיד את ערך הקלט המינימלי ותיל הפלט התחתון נושא תמיד את ערך הקלט המקסימלי.

28.1-3
פרופסור נילסן טוען שאם מוסיפים משווה במקום כלשהו ברשת מיון, הרשת המתקבלת גם היא ממיינת. הראה שהפרופסור טועה על־ידי הוספת משווה לרשת שבאיור 28.2 כך שהרשת המתקבלת אינה ממיינת כל תמורת קלט.

28.1-4
הוכח שעומקה של רשת מיון כלשהי בעלת n קלטים הוא לפחות $\lg n$.

28.1-5
הוכח שמספר המשווים ברשת מיון כלשהי בעלת n קלטים הוא לפחות $\Omega(n \lg n)$.

28.1-6
הוכח שרשת ההשוואה המוצגת באיור 28.3 היא למעשה רשת מיון, ותאר כיצד המבנה שלה קשור לזה של מיון־הכנסה (סעיף 1.1).



איור 28.3 רשת מיון המבוססת על מיון־הכנסה עבור תרגיל 28.1-6.

28.1-7
ניתן לייצג רשת השוואה בעלת n קלטים המכילה c משווים כרשימה של c זוגות של מספרים שלמים בתחום מ־1 עד n. אם שני זוגות מכילים אותו שלם, אזי סדר המשווים המתאימים ברשת נקבע על פי סדר הזוגות ברשימה. בהינתן ייצוג זה, תאר אלגוריתם (סדרתי) שרץ בזמן $O(n + c)$ לקביעת עומקה של רשת השוואה.

28.1-8 *
נניח כי בנוסף למשווה מן הסוג הרגיל אנו משתמשים גם במשווה "הפוך" אשר מפיק את המינימום שלו על תיל הפלט התחתון ואת המקסימום שלו על תיל הפלט העליון. הראה כיצד ניתן להמיר כל רשת מיון המורכבת מ־c משווים רגילים והפוכים לרשת מיון המורכבת מ־c משווים רגילים בלבד. הוכח את נכונותה של שיטת ההמרה שלך.

28.2 עקרון ה־0-1

עקרון ה־0-1 (zero-one principle), קובע שאם רשת מיון פועלת כהלכה כאשר קלטיה שייכים לקבוצה $\{0, 1\}$, אזי היא פועלת כהלכה על מספרי קלט כלשהם (המספרים יכולים להיות שלמים, ממשיים או, באופן כללי, קבוצת ערכים כלשהי הלקוחה מתוך קבוצה סדורה לינארית כלשהי). כאשר אנו עוסקים בבנייתן של רשתות מיון ורשתות השוואה אחרות, עקרון ה־0-1 מאפשר לנו להתמקד בפעולתן של הרשתות על סדרות קלט המורכבות אך ורק מ־0־ים ומ־1־ים. ברגע שהצלחנו לבנות רשת מיון והוכחנו שהיא ממיינת כהלכה כל סדרה של 0־ים ו־1־ים, נוכל להסתמך על עקרון ה־0-1 כדי להוכיח שהרשת ממיינת כהלכה סדרות המורכבות מערכים כלשהם.

הוכחת עקרון ה־0-1 מבוססת על מושג הפונקציה המונוטונית העולה (סעיף 2.2).

למה 28.1
אם רשת השוואה ממירה סדרת קלט $a = \langle a_1, a_2, \ldots, a_n \rangle$ לסדרת פלט $b = \langle b_1, b_2, \ldots, b_n \rangle$, אזי עבור כל פונקציה מונוטונית עולה f, הרשת ממירה את סדרת הקלט:

$$f(a) = \langle f(a_1), f(a_2), \ldots, f(a_n) \rangle$$

לסדרת הפלט:

$$f(b) = \langle f(b_1), f(b_2), \ldots, f(b_n) \rangle.$$

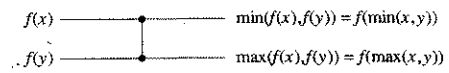
הוכחה תחילה נוכיח את הטענה שאם f היא פונקציה מונוטונית עולה, אזי משוואה יחיד המקבל את הקלטים $f(x)$ ו־$f(y)$ מפיק את הפלטים $f(\min(x, y))$ ו־$f(\max(x, y))$. לאחר מכן נשתמש באינדוקציה כדי להוכיח את הלמה.

כדי להוכיח את הטענה, נתבונן במשוואה שערכי הקלטים שלו הם x ו־y. הפלט העליון של המשוואה הוא $\min(x, y)$ והפלט התחתון שלו הוא $\max(x, y)$. נניח שאנו מזינים עתה למשוואה את הקלטים $f(x)$ ו־$f(y)$, כמתואר באיור 28.4. פעולת המשוואה מניבה את הערך $\min(f(x), f(y))$ על הפלט העליון ואת הערך $\max(f(x), f(y))$ על הפלט התחתון. מאחר ש־f היא מונוטונית עולה, הרי ש־$x \le y$ גורר $f(x) \le f(y)$. לכן מתקיימות הזהויות:

$$\min(f(x), f(y)) = f(\min(x, y))$$

$$\max(f(x), f(y)) = f(\max(x, y))$$

מכאן, שהמשוואה מפיק את הערכים $f(\min(x, y))$ ו־$f(\max(x, y))$ כאשר הקלטים שלו הם $f(x)$ ו־$f(y)$, ובכך נשלמה הוכחת הטענה.

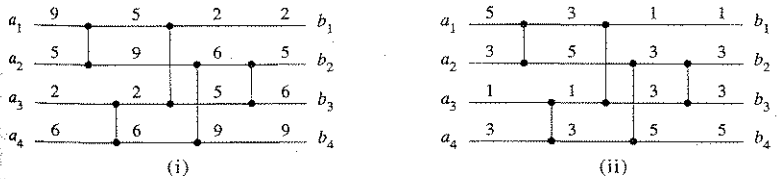


איור 28.4 פעולתו של המשוואה מהוכחת למה 28.1. הפונקציה f היא מונוטונית עולה.

באמצעות אינדוקציה על עומקו של כל תיל ברשת השוואה כללית, ניתן להוכיח תוצאה חזקה יותר מטענת הלמה: אם תיל מקבל את הערך a_i כאשר סדרת הקלט לרשת היא a, אזי התיל מקבל את הערך $f(a_i)$ כאשר סדרת הקלט לרשת היא $f(a)$. מכיוון שטענה זו כוללת גם את תילי הפלט, הוכחתה תוכיח גם את הלמה.

עבור בסיס האינדוקציה, נתבונן בתיל בעומק 0, כלומר, תיל קלט a_i. התוצאה נובעת באופן טריוויאלי: אם סדרת הקלט לרשת היא a, אזי תיל הקלט הזה יישא את הערך $f(a_i)$. עבור צעד האינדוקציה, נתבונן בתיל בעומק d, כאשר $d \ge 1$. תיל זה הוא הפלט של משוואה בעומק d, ועומקם של תילי הקלט למשוואה זה קטן ממש מ־d. לכן, על פי הנחת האינדוקציה, אם תילי הקלט למשוואה נושאים את הערכים a_i ו־a_j כאשר סדרת הקלט היא a, אזי הם נושאים את הערכים $f(a_i)$ ו־$f(a_j)$ כאשר סדרת הקלט היא $f(a)$. על פי טענתנו הקודמת, תילי הפלט של משוואה זה נושאים אז את הערכים $f(\min(a_i, a_j))$ ו־$f(\max(a_i, a_j))$. מאחר שהם נושאים את הערכים $\min(a_i, a_j)$ ו־$\max(a_i, a_j)$ כאשר סדרת הקלט היא a, הוכחנו את הלמה. ■

כדוגמה ליישומה של למה 28.1, איור 28.5 מראה את רשת המיון מאיור 28.2 כאשר מפעילים על הקלטים שלה את הפונקציה המונוטונית העולה $f(x) = \lceil x/2 \rceil$. הערך על כל תיל הוא הערך המתקבל מהפעלת f על הערך שעל תיל זה באיור 28.2.



איור 28.5 **(i)** רשת המיון מאיור 28.2 עם סדרת הקלט $\langle 9, 5, 2, 6 \rangle$. **(ii)** אותה רשת מיון לאחר הפעלת הפונקציה המונוטונית העולה $f(x) = \lceil x/2 \rceil$ על סדרת הקלט מ־(i). ערכו של כל תיל ברשת שווה לערך המתקבל מהפעלת f על ערכו של תיל זה ב־(i).

כאשר רשת השוואה היא רשת מיון, למה 28.1 מאפשרת לנו להוכיח את התוצאה המפתיעה הבאה.

משפט 28.2 (עקרון ה־0-1)
אם רשת השוואה בעלת n קלטים ממיינת כהלכה את כל 2^n הסדרות האפשריות של 0־ים ו־1־ים, אזי היא ממיינת כהלכה את כל הסדרות המורכבות ממספרים כלשהם.

הוכחה נניח, בדרך השלילה, שהרשת ממיינת כהלכה את כל הסדרות של 0־ים ו־1־ים, אבל קיימת סדרה המורכבת ממספרים כלשהם שהרשת אינה ממיינת כהלכה. כלומר, קיימת סדרת קלט $\langle a_1, a_2, \ldots, a_n \rangle$ המכילה איברים a_i ו־a_j כך ש־$a_i < a_j$, אולם הרשת מציבה את a_j לפני a_i בסדרת הפלט. נגדיר פונקציה מונוטונית עולה f באופן הבא:

$$f(x) = \begin{cases} 0 & \text{אם } x \le a_i \\ 1 & \text{אם } x > a_i \end{cases}$$

מאחר שהרשת מציבה את a_j לפני a_i בסדרת הפלט כאשר סדרת הקלט היא $\langle a_1, a_2, \ldots, a_n \rangle$, אזי למה 28.1 גובע שהיא מציבה את $f(a_j)$ לפני $f(a_i)$ בסדרת הפלט כאשר סדרת הקלט היא $\langle f(a_1), f(a_2), \ldots, f(a_n) \rangle$. אולם, מאחר ש־$f(a_j) = 1$ ו־$f(a_i) = 0$, הרי שמכאן נובע כי הרשת אינה ממיינת כהלכה את סדרת ה־0-1 $\langle f(a_1), f(a_2), \ldots, f(a_n) \rangle$, וזוהי סתירה. ■

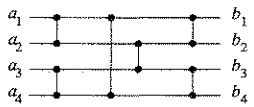
תרגילים

28.2-1
הוכח שהפעלת פונקציה מונוטונית עולה על סדרה ממוינת יוצרת סדרה ממוינת.

28.2-2
הוכח שרשת השוואה בעלת n קלטים ממיינת כהלכה את סדרת הקלט $\langle n, n-1, \ldots, 1 \rangle$ אם ורק אם היא ממיינת כהלכה את $n-1$ סדרות ה־0-1:

$$\langle 1, 0, 0, \ldots, 0, 0 \rangle, \langle 1, 1, 0, \ldots, 0, 0 \rangle, \ldots, \langle 1, 1, 1, \ldots, 1, 0 \rangle$$

28.2-3
הוכח בעזרת עקרון ה־0-1 כי רשת ההשוואה המוצגת באיור 28.6 היא רשת מיון.



איור 28.6 רשת מיון למיון 4 מספרים.

28.2-4
נסח והוכח מקבילה לעקרון ה־0-1 עבור המודל של עצי־החלטה. (**רמז:** ודא שאתה מטפל כהלכה במצבי שוויון.)

28.2-5
הוכח שרשת מיון בעלת n קלטים מכילה בהכרח לפחות משוואה אחד בין הקו האופקי ה־i לקו האופקי ה־$i+1$, עבור כל $i = 1, 2, \ldots, n-1$.

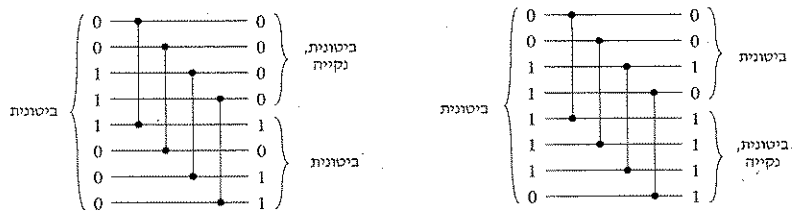
28.3 רשת מיון ביטונית

הצעד הראשון בבנייתה של רשת מיון יעילה הוא בנייתה של רשת השוואה הממיינת כל **סדרה ביטונית** (bitonic sequence): סדרה אשר עולה מונוטונית ואחר־כך יורדת מונוטונית, או שניתן להופכה לכזאת על־ידי הזזה מעגלית. לדוגמה, הסדרות $\langle 1, 4, 6, 8, 3, 2 \rangle$, $\langle 6, 9, 4, 2, 3, 5 \rangle$ ו־$\langle 9, 8, 3, 2, 4, 6 \rangle$ הן כולן ביטוניות. סדרות של 0־ים ו־1־ים שהן ביטוניות הן בעלות מבנה פשוט: כולן מהצורה $0^i1^j0^k$ או מהצורה $1^i0^j1^k$ עבור $i, j, k \geq 0$ כלשהם. יש לשים לב כי סדרות מונוטוניות עולות או מונוטוניות יורדות גם הן ביטוניות.

הממיין הביטוני שנבנה הוא רשת השוואה הממיינת סדרות ביטוניות של 0־ים ו־1־ים. בתרגיל 28.3-6 תתבקש להוכיח שהממיין הביטוני ממיין כהלכה סדרות ביטוניות המורכבות ממספרים כלשהם.

מנקה־מחצית

ממיין ביטוני מורכב מכמה שלבים שכל אחד מהם נקרא **מנקה־מחצית** (half-cleaner). כל מנקה־מחצית הוא רשת השוואה שעומקה 1, אשר בה קו הקלט ה־i מושווה עם קו הקלט ה־$n/2 + i$, עבור $i = 1, 2, \ldots, n/2$. (אנו מניחים ש־n זוגי.) באיור 28.7 מוצג HALF-CLEANER[8], מנקה־מחצית בעל 8 קלטים ו־8 פלטים.



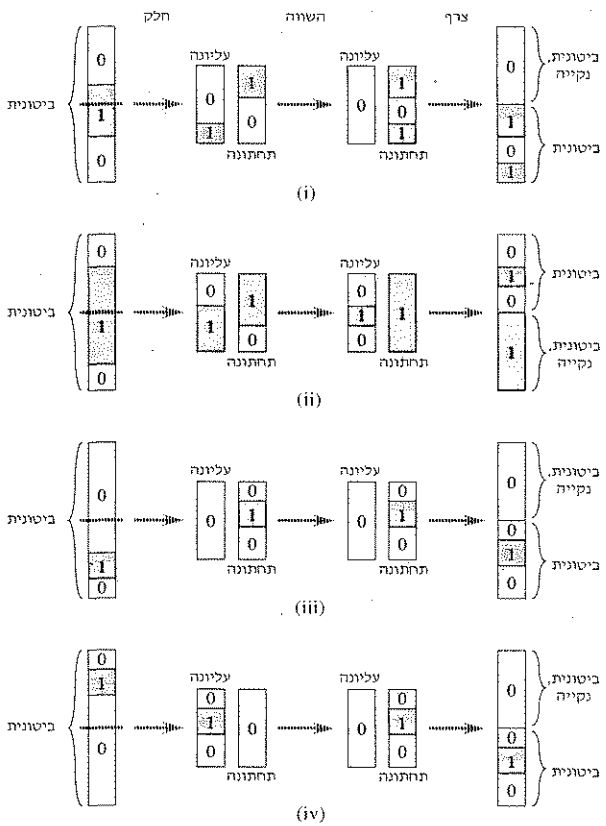
איור 28.7 רשת ההשוואה HALF-CLEANER[8]. באיור מוצגות שתי דוגמאות שונות לסדרות קלט ופלט של 0־ים ו־1־ים. ההנחה היא שסדרות הקלט ביטוניות. מנקה־מחצית מבטיח שכל האיברים במחצית העליונה של הפלט אינם גדולים מכל האיברים במחצית הפלט התחתונה. יתירה מזו, שתי המחציות הן ביטוניות, ולפחות אחת מהן "נקייה".

כאשר סדרה ביטונית של 0־ים ו־1־ים נמסרת כקלט למנקה־מחצית, מנקה־המחצית יוצר סדרת פלט שבה הערכים הקטנים יותר נמצאים במחצית העליונה, הערכים הגדולים יותר נמצאים במחצית התחתונה, ושתי המחציות הן ביטוניות. למעשה, לפחות אחת מן המחציות היא **נקייה** — דהיינו, מורכבת כולה מ־0־ים בלבד או מ־1־ים בלבד — ומכאן השם "מנקה־מחצית". (נשים לב לכך שכל הסדרות הנקיות הן ביטוניות.) הלמה הבאה מוכיחה תכונות אלה של מנקי־מחצית.

למה 28.3

אם הקלט למנקה־מחצית הוא סדרה ביטונית של 0־ים ו־1־ים, אזי הפלט מקיים את התכונות הבאות: המחצית העליונה וגם המחצית התחתונה הן ביטוניות; כל איבר במחצית העליונה קטן מכל אחד מן האיברים במחצית התחתונה או שווה לו; ולפחות מחצית אחת היא נקייה.

הוכחה רשת ההשוואה HALF-CLEANER[n] משווה את הקלט ה־i והקלט ה־$i + n/2$, עבור $i = 1, 2, \ldots, n/2$. בלי הגבלת הכלליות, נניח שהקלט הוא מן הצורה $0\ldots0\ldots1\ldots1$. (המצב שבו הקלט הוא מן הצורה $1\ldots1\ldots0\ldots0$ הוא סימטרי.) קיימים שלושה מקרים אפשריים, ואלה תלויים בגוש של ה־0־ים או ה־1־ים העוקבים שבו נמצאת נקודת האמצע $n/2$ של סדרת הקלט. אחד המקרים האלה (זה שבו נקודת האמצע נמצאת בגוש של ה־1־ים) מתחלק שוב לשני מקרים. ארבעת המקרים מוצגים באיור 28.8. בכל אחד מן המקרים המוצגים מתקיימת הלמה. ■



איור 28.8 ההשוואות האפשריות שמבצע HALF-CLEANER[n]. ההנחה היא כי סדרת הקלט היא סדרה ביטונית של 0־ים ו־1־ים, ובלי הגבלת הכלליות אנו מניחים שסדרת הקלט היא מהצורה $0\ldots0\ldots1\ldots1\ldots0\ldots0$. תת־סדרות של 0־ים מופיעות בלבן, ותת־סדרות של 1־ים באפור. אנו יכולים לחשוב על n הקלטים כאילו הם מחולקים לשתי מחציות, כך ששווואות מתבצעות בין הקלט ה־i וה־$i + n/2$, עבור $i = 1, 2, \ldots, n/2$. **(i)**-**(ii)** מקרים שבהם החלוקה מתרחשת בתוך תת־סדרה של 0־ים. בכל המקרים, אף איבר במחצית העליונה של הפלט אינו גדול מאף אחד מן האיברים במחצית התחתונה, שתי המחציות ביטוניות, ולפחות אחת מהן נקייה. **(iii)**-**(iv)**

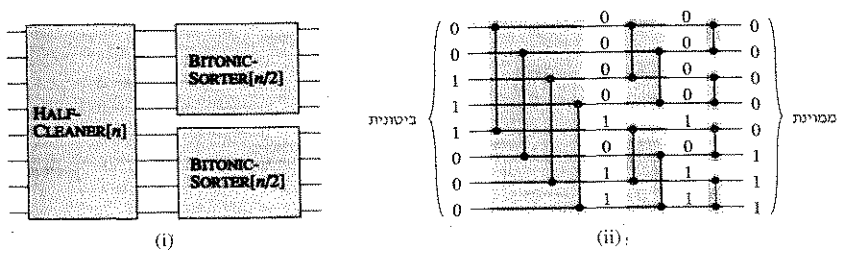
הממיין הביטוני

על־ידי צירוף רקורסיבי של מנקי־מחצית, כמתואר באיור 28.9, ניתן לבנות **ממיין ביטוני** (bitonic sorter), שהוא רשת הממיינת סדרות ביטוניות. השלב הראשון של BITONIC-SORTER$[n]$ מורכב מ־HALF-CLEANER$[n]$, אשר, על־פי למה 28.3, מפיק שתי סדרות ביטוניות שגודלן מחצית מסדרת הקלט המקורית, כך שאף איבר במחצית העליונה אינו גדול מאף איבר במחצית התחתונה. לכן, נוכל להשלים את המיון על־ידי כך שנשתמש בשני עותקים של BITONIC-SORTER$[n/2]$ למיון רקורסיבי של שתי המחציות. באיור 28.9(i) מוצג המבנה הרקורסיבי של הממיין הביטוני, ובאיור 28.9(ii) נפרשה הרקורסיה כדי להראות את מנקי־המחצית ההולכים וקטנים, אשר מרכיבים את שארית הממיין הביטוני. העומק $D(n)$ של BITONIC-SORTER$[n]$ נתון על־ידי נוסחת הנסיגה:

$$D(n) = \begin{cases} 0 & \text{אם } n = 1 \\ D(n/2) + 1 & \text{אם } k \geq 1 \text{ וגם } n = 2^k \end{cases}$$

שפתרונה הוא $D(n) = \lg n$.

לכן, סדרה ביטונית של 0־ים ו־1־ים באורך n ניתנת למיון על־ידי BITONIC-SORTER$[n]$, שעומקו $\lg n$. מן האנלוגיה לעקרון ה־0-1 הנתונה כתרגיל 28.3-6 נובע שבאמצעות רשת זו ניתן למיין כל סדרה ביטונית המורכבת ממספרים כלשהם.



איור 28.9 רשת ההשוואה BITONIC-SORTER$[n]$ עבור $n = 8$. **(i)** המבנה הרקורסיבי: HALF-CLEANER$[n]$ ואחריו שני עותקים של BITONIC-SORTER$[n/2]$ הפועלים במקביל. **(ii)** הרשת לאחר פרישת הרקורסיה. כל מנקה מחצית מוצג על־ידי מלבן אפור. באיור מוצגת פעולתה של הרשת על דוגמה של סדרת 0־ים ו־1־ים.

תרגילים

28.3-1

כמה סדרות ביטוניות של 0־ים ו־1־ים באורך n קיימות?

28.3-2

הראה כי BITONIC-SORTER$[n]$, כאשר n הוא חזקה מדויקת של 2, מכיל $\Theta(n \lg n)$ משווים.

28.3-3

תאר כיצד ניתן לבנות ממיין ביטוני בעומק $O(\lg n)$ כאשר מספר הקלטים n אינו חזקה מדויקת של 2.

28.3-4

הוכח שאם הקלט למנקה־מחצית הוא סדרה ביטונית של מספרים כלשהם, אזי הפלט מקיים את התכונות הבאות: המחצית העליונה וגם המחצית התחתונה של הפלט הן ביטוניות, ואף איבר במחצית העליונה אינו גדול מאף איבר במחצית התחתונה.

28.3-5

בהינתן שתי סדרות של 0־ים ו־1־ים, הוכח שאם אף איבר בסדרה אחת אינו גדול מאף איבר בסדרה האחרת, אזי אחת משתי הסדרות נקייה.

28.3-6

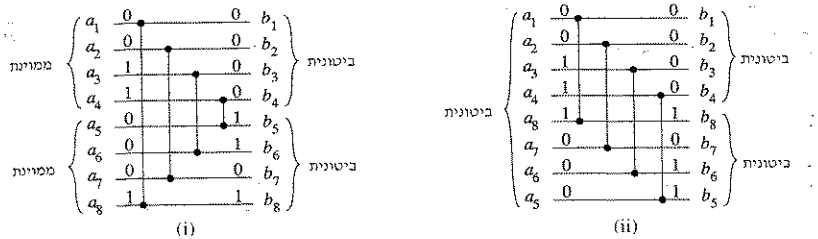
הוכח את האנלוגיה הבאה של עקרון ה־0-1 עבור רשתות מיון ביטוניות: רשת השוואה הממיינת כהלכה סדרה ביטונית כלשהי של 0־ים ו־1־ים, ממיינת כהלכה כל סדרה ביטונית המורכבת ממספרים כלשהם.

28.4 רשת מיזוג

רשת המיון שלנו תהיה בנויה מ**רשתות מיזוג** (merging networks), שהן רשתות המסוגלות למזג שתי סדרות קלט ממוינות לסדרת פלט ממוינת אחת. אנו נשנה את BITONIC-SORTER$[n]$ כדי ליצור את רשת המיזוג MERGER$[n]$. את נכונותה של רשת המיזוג נוכיח רק עבור סדרות קלט של 0־ים ו־1־ים, כפי שעשינו זאת לגבי הממיין הביטוני. בתרגיל 28.4-1 תתבקש להראות כיצד ניתן להכליל את ההוכחה לערכי קלט כלשהם.

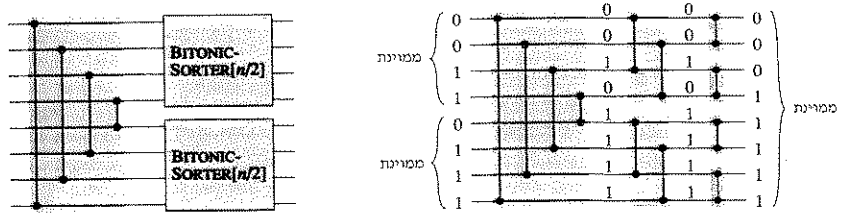
רשת המיזוג מבוססת על האינטואיציה הבאה: בהינתן שתי סדרות ממוינות, אם הופכים את סדר האיברים בסדרה השנייה ואז משרשרים את שתי הסדרות, מתקבלת סדרה ביטונית. לדוגמה, בהינתן שתי סדרות ממוינות של 0־ים ו־1־ים $X = 00000111$ ו־$Y = 00001111$, נהפוך את Y ונקבל $Y^R = 11110000$. שרשור של X ו־Y^R נותן את הסדרה 0000011111110000 שהיא ביטונית. לכן, כדי למזג את שתי סדרות הקלט X ו־Y, די לבצע מיון ביטוני על השרשור של X ו־Y^R.

נוכל לבנות MERGER$[n]$ על־ידי שינוי מנקה־המחצית הראשון של BITONIC-SORTER$[n]$. המפתח הוא לבצע במובלע את הפיכת המחצית השנייה של הקלטים. בהינתן שתי סדרות ממוינות $\langle a_1, a_2, \ldots, a_{n/2} \rangle$ ו־$\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$ שיש למזג, ברצוננו לקבל למעשה את תוצאת המיון הביטוני של הסדרה $\langle a_1, a_2, \ldots, a_{n/2}, a_n, a_{n-1}, \ldots, a_{n/2+1} \rangle$. מכיוון שמנקה־המחצית של BITONIC-SORTER$[n]$ משווה את הקלטים i ו־$n/2 + i$, עבור $i = 1, 2, \ldots, n/2$, אנו בונים את השלב הראשון של רשת המיזוג כך שישווה את הקלטים i ו־$n - i + 1$. באיור 28.10 ניתן לראות את ההקבלה. הנקודה העדינה היחידה היא, שסדר הפלטים מן המחצית התחתונה של השלב הראשון של MERGER$[n]$ הפוך מסדר הפלטים של מנקה־מחצית רגיל. אולם, מאחר שהסדרה ההפוכה לסדרה ביטונית גם היא ביטונית, הרי ששתי מחציות הפלט, העליונה והתחתונה, של השלב הראשון של רשת המיזוג מקיימות את התכונות המובאות בלמה 28.3, ולפיכך ניתן למיין אותן מיון ביטוני במקביל, כדי לקבל את הפלט הממוין של רשת המיזוג.



איור 28.10 השוואה בין השלב הראשון של MERGER[n] לבין HALF-CLEANER[n] עבור $n = 8$. **(i)** השלב הראשון של רשת המיזוג MERGER[n] ממיר את שתי סדרות הקלט המונוטוניות $\langle a_1, a_2, \ldots, a_{n/2} \rangle$ ו־$\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$ לשתי סדרות ביטוניות $\langle b_1, b_2, \ldots, b_{n/2} \rangle$ ו־$\langle b_{n/2+1}, b_{n/2+2}, \ldots, b_n \rangle$. **(ii)** הפעולה השקולה ב־HALF-CLEANER[n]. סדרת הקלט הביטונית $\langle a_1, a_2, \ldots, a_{n/2-1}, a_{n/2}, a_n, a_{n-1}, \ldots, a_{n/2+2}, a_{n/2+1} \rangle$ מומרת לשתי סדרות ביטוניות $\langle b_1, b_2, \ldots, b_{n/2} \rangle$ ו־$\langle b_n, b_{n-1}, \ldots, b_{n/2+1} \rangle$.

רשת המיזוג המתקבלת מוצגת באיור 28.11. רק השלב הראשון של MERGER[n] שונה מ־BITONIC-SORTER[n]. לכן, עומקה של רשת המיזוג MERGER[n] הוא $\lg n$, כמו זה של BITONIC-SORTER[n].



איור 28.11 רשת הממזגת שתי סדרות קלט ממוינות לסדרת פלט ממוינת אחת. את הרשת MERGER[n] ניתן לראות כ־BITONIC-SORTER[n] שבו מנקה־המחצית הראשון שוזה כך שישווה את הקלטים i ו־$i + 1$, עבור $i = 1, 2, \ldots, n/2$, כאן, $n = 8$. **(i)** הרשת מוצגת כשהיא מפורקת לשלב הראשון ואחריו שני עותקים מקבילים של BITONIC-SORTER[$n/2$]. **(ii)** אותה רשת, כאשר הרקורסיה פרושה. מעל לתילים רשומים ערכי 0-1 לדוגמה, והשלבים השונים תחומים על ידי מלבנים אפורים.

תרגילים

28.4-1

הוכח אנלוגיה של עקרון ה־0-1 עבור רשתות מיזוג. בפרט, הראה כי רשת השוואה שמסוגלת למזג כל שתי סדרות מונוטוניות עולות של 0־ים ו־1־ים מסוגלת למזג כל שתי סדרות מונוטוניות עולות של מספרים כלשהם.

28.4-2

על כמה סדרות קלט שונות של 0-1 יש להפעיל רשת השוואה כדי לוודא שהיא רשת מיזוג?

28.4-3

הראה שכל רשת המסוגלת למזג פריט אחד עם $n - 1$ פריטים ממוינים ליצירת סדרה ממוינת באורך n, חייבת להיות בעלת עומק של $\lg n$ לפחות.

* 28.4-4

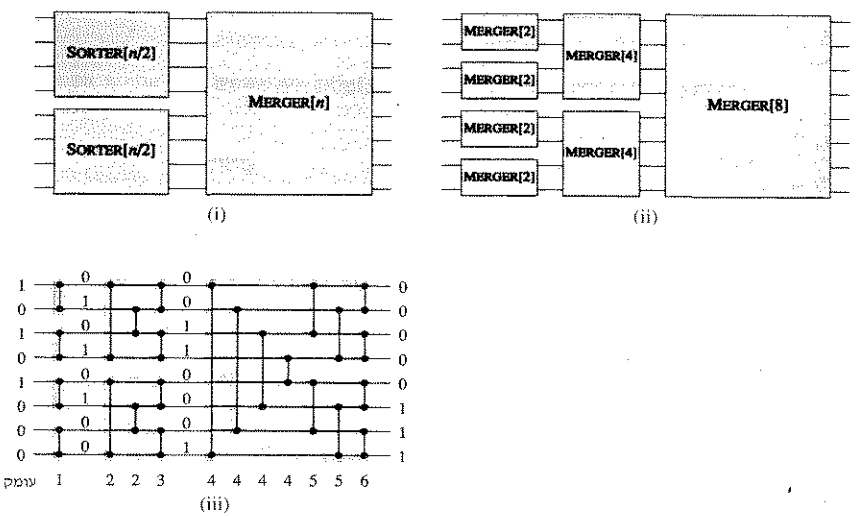
נתונה רשת מיזוג שהקלטים שלה הם $a_1, a_2, \ldots, a_n$, כאשר n הוא חזקה מדויקת של 2, ושתי הסדרות המונוטוניות שיש למזג הן $\langle a_1, a_3, \ldots, a_{n-1} \rangle$ ו־$\langle a_2, a_4, \ldots, a_n \rangle$. הוכח שמספר ההשוואות בסוג כזה של רשת מיזוג הוא $\Omega(n \lg n)$. מדוע זהו חסם תחתון מעניין? **(רמז:** חלק את המשוואים לשלוש קבוצות.)

* 28.4-5

הוכח שלכל רשת מיזוג דרושים $\Omega(n \lg n)$ משווים, בלי תלות בסדר הקלטים.

28.5 רשת מיון

עתה ברשותנו כל הכלים הנחוצים לבניית רשת שמסוגלת למיין כל סדרת קלט שהיא. רשת המיון SORTER[n] משתמשת ברשת המיזוג כדי לממש גרסה מקבילית של מיון־מיזוג מסעיף 1.3.1. המבנה של רשת המיון ואופן פעולתה מודגמים באיור 28.12.



איור 28.12 רשת המיון SORTER[n] הבנויה מצירוף רקורסיבי של רשתות מיזוג. **(i)** המבנה הרקורסיבי. **(ii)** פרישת הרקורסיה. **(iii)** החלפת ריבועי ה־MERGER ברשתות המיזוג הממשיות. באיור רשום העומק של כל משווה, ומעל לתילים רשומים ערכי 0-1 לדוגמה.

באיור 28.12(i) מוצג המבנה הרקורסיבי של $\text{SORTER}[n]$. איברי הקלט ממוינים רקורסיבית באמצעות שני עותקים של $\text{SORTER}[n/2]$, הממיינים (במקביל) שתי תת־סדרות שאורך כל אחת מהן $n/2$. לאחר מכן, שתי הסדרות המתקבלות ממוזגות באמצעות $\text{MERGER}[n]$. מקרה הגבול של הרקורסיה הוא כאשר $n = 1$, שבו משתמשים בתיל כדי למיין את הסדרה בת האיבר האחד, שכן סדרה בת איבר אחד כבר ממוינת. איור 28.12(ii) מראה את התוצאה של פרישת הרקורסיה, ובאיור 28.12(iii) מוצגת הרשת הממשית, לאחר החלפת ריבועי ה־MERGER מאיור 28.12(ii) ברשתות המיזוג הממשיות.

ברשת $\text{SORTER}[n]$ עוברים הנתונים דרך $\lg n$ שלבים. כל אחד מאיברי הקלט לרשת הוא למעשה סדרה ממוינת בת איבר אחד. השלב הראשון של $\text{SORTER}[n]$ מורכב מ־$n/2$ עותקים של $\text{MERGER}[2]$ הפועלים במקביל וממזגים זוגות של סדרות בנות איבר אחד לסדרות ממוינות באורך 2. השלב השני מורכב מ־$n/4$ עותקים של $\text{MERGER}[4]$ הממזגים זוגות של סדרות ממוינות אלה בנות 2 איברים לסדרות ממוינות באורך 4. במקרה הכללי, עבור $k = 1, 2, \ldots, \lg n$, השלב ה־k מורכב מ־$n/2^k$ עותקים של $\text{MERGER}[2^k]$ הממזגים זוגות של סדרות ממוינות בנות 2^{k-1} איברים לסדרות ממוינות באורך 2^k. בשלב הסופי מתקבלת סדרה ממוינת אחת המורכבת מכל ערכי הקלט. ניתן להראות באינדוקציה שרשת מיון זו ממיינת כהלכה סדרות של 0-1, ולכן, על פי עקרון ה־0-1 (משפט 28.2), היא מסוגלת למיין ערכים כלשהם.

ניתן לנתח רקורסיבית את עומקה של רשת המיון. העומק $D(n)$ של $\text{SORTER}[n]$ הוא העומק $D(n/2)$ של $\text{SORTER}[n/2]$ (הרשת מכילה שני עותקים של $\text{SORTER}[n/2]$ אולם הם פועלים במקביל) ועוד העומק $\lg n$ של $\text{MERGER}[n]$. לכן, עומקה של $\text{SORTER}[n]$ נתון על ידי נוסחת הנסיגה:

$$D(n) = \begin{cases} 0 & \text{אם } n = 1 \\ D(n/2) + \lg n & \text{אם } n = 2^k \text{ וגם } k \geq 1 \end{cases}$$

אשר פותרונה הוא $D(n) = \Theta(\lg^2 n)$. ניתן אפוא למיין במקביל n מספרים בזמן $O(\lg^2 n)$.

תרגילים

28.5-1
כמה משווים מכילה $\text{SORTER}[n]$?

28.5-2
הראה שעומקה של $\text{SORTER}[n]$ הוא בדיוק $(\lg n)(\lg n + 1)/2$.

28.5-3
נניח שאנו משנים משווה כך שיקבל כקלט שתי רשימות ממוינות באורך k, ימזג אותן וינתב את k הערכים הגדולים ביותר לפלט "max" ואת k הקטנים ביותר לפלט "min". הראה שכל רשת מיון על n קלטים עם משווים כאלה מסוגלת למיין nk מספרים, בהנחה שכל קלט לרשת הוא רשימה ממוינת באורך k.

28.5-4
נניח שנתונים לנו $2n$ איברים $\langle a_1, a_2, \ldots, a_{2n} \rangle$ וברצוננו לחלק אותם ל־n האיברים הקטנים ביותר ול־n האיברים הגדולים ביותר. הוכח שניתן לעשות זאת בתוספת קבועה של עומק לאחר מיון נפרד של $\langle a_1, a_2, \ldots, a_n \rangle$ ו־$\langle a_{n+1}, a_{n+2}, \ldots, a_{2n} \rangle$.

* 28.5-5
יהי $S(k)$ עומקה של רשת מיון בעלת k קלטים, ויהי $M(k)$ עומקה של רשת מיזוג בעלת $2k$ קלטים. נניח שעלינו למיין סדרה של n מספרים ואנו יודעים שמיקומו של כל מספר בסדרת הקלט מרוחק k מקומות לכל היותר ממיקומו הנכון בסדר הממוין. הראה שניתן למיין את n המספרים באמצעות רשת שעומקה $S(k) + 2M(k)$.

* 28.5-6
ניתן למיין את איבריה של מטריצה בגודל $m \times m$ על־ידי ביצוע התהליך הבא k פעמים:
1. מיין כל שורה אי־זוגית בסדר מונוטוני עולה.
2. מיין כל שורה זוגית בסדר מונוטוני יורד.
3. מיין כל טור בסדר מונוטוני עולה.

מהו המספר k של איטרציות הדרושות כדי למיין את המטריצה באמצעות תהליך זה ומהי תבניתו של הפלט הממוין?

בעיות

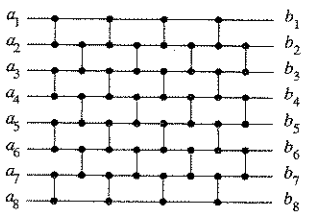
28-1 רשתות חילוף ממיינות
רשת השוואה היא **רשת חילוף** (transposition network) אם כל המשווים בה מחברים קווים סמוכים, כמו ברשת המוצגת באיור 28.3.

א. הראה שכל רשת חילוף הממיינת n קלטים מכילה $\Omega(n^2)$ משווים.

ב. הוכח שרשת חילוף עבור n קלטים היא רשת מיון אם ורק אם היא ממיינת את הסדרה $\langle n, n-1, \ldots, 1 \rangle$. (**רמז:** השתמש בטיעון אינדוקטיבי אלגוגי לזה ששימש בהוכחת למה 28.1.)

רשת מיון זוג־פרד (odd-even sorting network) עבור n קלטים $\langle a_1, a_2, \ldots, a_n \rangle$ היא רשת חילוף המורכבת מ־n רמות של משווים. באיור 28.13 מוצגת רשת כזאת עבור 8 קלטים. כפי שניתן לראות באיור, עבור $i = 1, 2, \ldots, n$ ו־$d = 1, 2, \ldots, n$, הקו ה־i מחובר על־ידי משווה בעומק d לקו $j = i + (-1)^{i+d}$, אם $1 \leq j \leq n$.

ג. הוכח שמשפחת רשתות המיון זוג־פרד היא אכן משפחה של רשתות מיון.



איור 28.13 רשת מיון זוג־פרד עבור 8 קלטים.

28-2 רשת המיזוג זוג־פרד של באטצ'ר (Batcher)

בסעיף 28.4 ראינו כיצד בונים רשת מיזוג המבוססת על מיון ביטוני. בבעיה זו נבנה **רשת מיזוג זוג־פרד** (odd-even merging network). אנו מניחים כי n הוא חזקה מדויקת של 2, וברצוננו למזג את סדרת האיברים הממוינת על הקווים $\langle a_1, a_2, \ldots, a_n \rangle$ עם סדרת האיברים הממוינת על הקווים $\langle a_{n+1}, a_{n+2}, \ldots, a_{2n} \rangle$. אנו בונים באופן רקורסיבי שתי רשתות מיזוג זוג־פרד אשר ממזגות במקביל תת־סדרות ממוינות. הראשונה ממזגת את הסדרה על הקווים $\langle a_1, a_3, \ldots, a_{n-1} \rangle$ עם הסדרה על הקווים $\langle a_{n+1}, a_{n+3}, \ldots, a_{2n-1} \rangle$ (האיברים האי־זוגיים). הרשת השנייה ממזגת את $\langle a_2, a_4, \ldots, a_n \rangle$ עם $\langle a_{n+2}, a_{n+4}, \ldots, a_{2n} \rangle$ (האיברים הזוגיים). כדי לחבר את שתי תת־סדרות הממוינות מציבים משווה בין a_{2i-1} ו־a_{2i}, עבור $i = 1, 2, \ldots, n$.

א. שרטט רשת מיזוג בעלת $2n$ קלטים עבור $n = 4$.

ב. השתמש בעקרון ה־0-1 כדי להוכיח שכל רשת מיזוג זוג־פרד בעלת $2n$ קלטים היא אכן רשת מיזוג.

ג. מהו עומקה של רשת מיזוג זוג־פרד בעלת $2n$ קלטים? מהו גודלה?

28-3 רשתות תמורה

רשת תמורה (permutation network) בת n קלטים ו־n פלטים מכילה מתגים המאפשרים לחבר את הקלטים שלה לפלטים שלה בהתאם לכל אחת מ־$n!$ התמורות האפשריות. באיור 28.14(i) מופיעה רשת תמורה P_2 בת 2 קלטים ו־2 פלטים, המורכבת ממתג יחיד, בעל שני מצבים: חיבור ישיר של הקלטים לפלטים או הצלבתם.

א. הראה שאם נחליף כל משווה ברשת מיון במתג שבאיור 28.14(i), נקבל רשת תמורה. כלומר, עבור כל תמורה π, תהיה דרך לכוון את המתגים ברשת כך שהקלט i יחובר לפלט $\pi(i)$.

באיור 28.14(ii) מוצג המבנה הרקורסיבי של רשת תמורה P_8 בת 8 קלטים ו־8 פלטים, המורכבת משני עותקים של P_4 ומ־8 מתגים. באיור, המתגים מכוונים למימוש התמורה $\langle 4, 7, 3, 5, 1, 6, 8, 2 \rangle = \langle \pi(1), \pi(2), \ldots, \pi(8) \rangle = \pi$. מימוש זה דורש כי P_4 העליונה תממש (רקורסיבית) את התמורה $\langle 4, 2, 3, 1 \rangle$, ו־$P_4$ התחתונה תממש (רקורסיבית) את התמורה $\langle 2, 3, 1, 4 \rangle$.

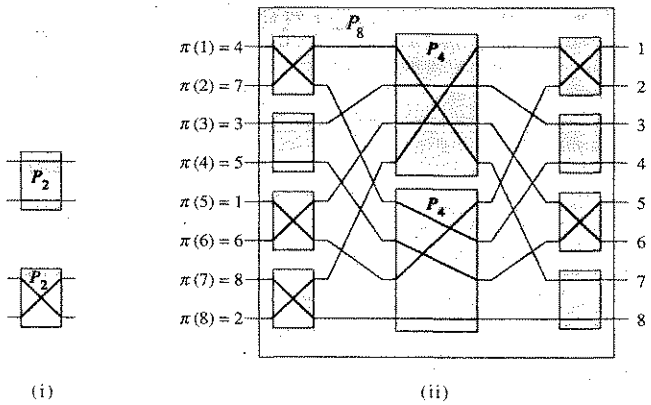
ב. הראה כיצד ניתן לממש את התמורה $\langle 5, 3, 4, 6, 1, 8, 2, 7 \rangle$ על P_8 על־ידי כך שתשרטט את מצב המתגים ואת התמורות המבוצעות על ידי שתי ה־P_4.

יהי n חזקה מדויקת של 2. נגדיר את P_n באופן רקורסיבי במונחי שתי $P_{n/2}$, בדומה לאופן שבו הגדרנו את P_8.

ג. תאר אלגוריתם (רגיל, עבור מכונת גישה אקראית) שזמן ריצתו $O(n)$, אשר מכוון את n המתגים המחוברים לקלטים ולפלטים של P_n ומציין את התמורות אשר שתי ה־$P_{n/2}$ חייבות לממש כדי לקבל כל תמורה נתונה של n איברים. הוכח את נכונותו של האלגוריתם שלך.

ד. מהם עומקה וגודלה של P_n? כמה זמן יידרש למכונת גישה אקראית רגילה לחשב את כל מצבי המתגים, כולל אלה שבתוך שתי ה־$P_{n/2}$?

ה. הראה שעבור $n > 2$, לכל רשת תמורה — לא רק ל־P_n — ישנה בהכרח תמורה אחת לפחות אשר מתממשת על־ידי שני צירופים שונים של מצבי מתגים.



איור 28.14 רשתות תמורה. **(i)** רשת התמורה P_2, המורכבת ממתג יחיד שניתן לכוון אותו באחת משתי הדרכים המוצגות. **(ii)** הבנייה הרקורסיבית של P_8 מ־8 מתגים ושני עותקים של P_4. המתגים ושתי ה־P_4 מכוונים למימוש התמורה $\pi = \langle 4, 7, 3, 5, 1, 6, 8, 2 \rangle$.

הערות ביבליוגרפיות לפרק 28

Knuth [123] מכיל דיון ברשתות מיון וסקירה של ההיסטוריה שלהן. ככל הנראה, הן נחקרו לראשונה ב־1954 על־ידי P. N. Armstrong, R. J. Nelson ו־D. J. O'Connor. בשנות ה־60 המוקדמות גילה K. E. Batcher את הרשת הראשונה שהיתה מסוגלת למזג שתי סדרות של n מספרים בזמן $O(\lg n)$. הוא השתמש במיזוג זוג־פרד (ראה בעיה 2-28), וגם הראה כיצד ניתן להשתמש בשיטה זו כדי למיין n מספרים בזמן $O(\lg^2 n)$. זמן קצר לאחר מכן גילה ממיין ביטוני בעל עומק $O(\lg n)$, הדומה לזה שהוצג בסעיף 28.3. Knuth מייחס את עקרון ה־0-1 ל־W. G. Bouricius (1954), שהוכיח אותו בהקשר של עצי החלטה.

במשך זמן רב נותרה פתוחה השאלה אם קיימת רשת מיון בעומק $O(\lg n)$. ב־1983 הראו חוקרים שהתשובה היא כן, אך כן שאינו ממש מספק. רשת המיון AKS (הקרויה על שם מפתחיה: Ajtai, Komlós ו־Szemerédi [8]) מסוגלת למיין n מספרים, עומקה $O(\lg n)$, והיא מכילה $O(n \lg n)$ משווים. לרוע המזל, הקבועים החבויים בסימון O גדולים למדי (הרבה הרבה אלפים), ולכן היא אינה נחשבת למעשית.

29 מעגלים אריתמטיים

מודל החישוב שעליו מבוסס מחשב רגיל מניח כי הפעולות האריתמטיות הבסיסיות — חיבור, חיסור, כפל וחילוק — ניתנות לביצוע בזמן קבוע. זוהי הפשטה סבירה, מאחר שמרבית הפעולות הבסיסיות על מכונת גישה־אקראית הן בעלות עלויות דומות. אולם, בבואנו לתכנן את המעגלים המממשים פעולות אלה, אנו מגלים מהר מאוד שהביצועים תלויים בגודלם של המספרים שעליהם מבצעים את הפעולות. לדוגמה, כולנו למדנו בבית־הספר היסודי לחבר שני מספרים טבעיים, המיוצגים כמספרים עשרוניים בני n ספרות, ב־$\Theta(n)$ צעדים (אם כי לרוב לא הדגישו המורים את מספר הצעדים הנדרש).

בפרק זה נציג מעגלים המבצעים פונקציות אריתמטיות. כאשר הביצוע הוא סדרתי, $\Theta(n)$ הוא חסם הזמן האסימפטוטי הטוב ביותר שניתן לקוות להשיגו עבור חיבור שני מספרים בני n ספרות. אולם, באמצעות מעגלים הפועלים במקביל, ניתן להשיג חסם זמן טוב יותר. בפרק זה נתכנן מעגלים שיכולים לבצע במהירות חיבור וכפל. (חיסור אינו שונה במהותו מחיבור, ואת הטיפול בחילוק אנו דוחים לבעיה 29-1.) אנו נניח שכל הקלטים הם מספרים טבעיים בני n סיביות המיוצגים בייצוג בינרי.

נפתח בסעיף 29.1 בהצגת מעגלים צירופיים. נראה את ההתאמה בין עומקו של מעגל לבין "זמן הריצה" שלו. **הסוכם המלא** (full adder), המהווה אבן בניין של רוב המעגלים המוצגים בפרק זה, ישמש כדוגמה הראשונה שלנו למעגל צירופי. בסעיף 29.2 נציג שני מעגלים צירופיים לביצוע חיבור: **סוכם מגלגל־נשא** (ripple-carry adder), המחבר שני מספרים בני n סיביות בזמן $\Theta(n)$, ו**סוכם צופה־נשא** (carry-lookahead adder), העושה זאת בזמן $O(\lg n)$ בלבד. כמו כן נציג **סוכם שומר־נשא** (carry-save adder) שיכול לצמצם את בעיית החיבור של שלושה מספרים לבעיית החיבור של שני מספרים, בזמן $\Theta(1)$. בסעיף 29.3 נציג שני כופלים צירופיים: **כופל מערכי** (array multiplier), שדרוש לו זמן של $\Theta(n)$, ו**כופל עץ־וואלאס** (Wallace-tree multiplier), העושה זאת בזמן $\Theta(\lg n)$ בלבד. לבסוף, בסעיף 29.4, נציג מעגלים המכילים רכיבי אחסון מבוקרי־שעון (אוגרים), ונראה כיצד ניתן לחסוך בחומרה באמצעות שימוש חוזר במעגלים צירופיים.

29.1 מעגלים צירופיים

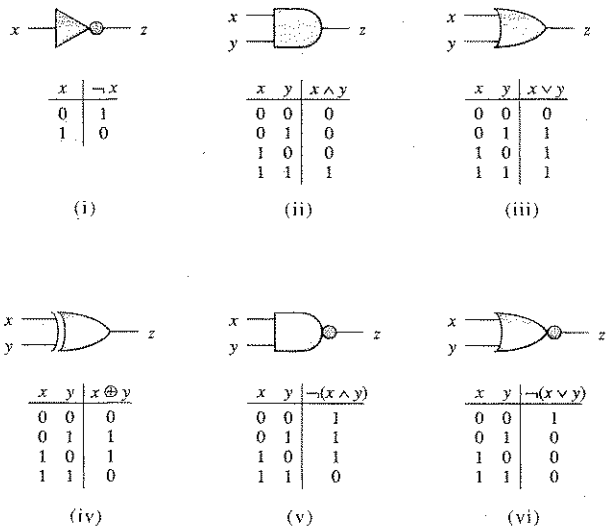
כמו רשתות ההשוואה מפרק 28, מעגלים צירופיים פועלים **במקביל**: רכיבים רבים יכולים לחשב ערכים בו־זמנית כצעד יחיד. בסעיף זה נגדיר מעגלים צירופיים ונחקור כיצד ניתן לבנות משערים בסיסיים מעגלים צירופיים גדולים יותר.

רכיבים צירופיים

מעגלים אריתמטיים במחשבים ממשיים בנויים מרכיבים צירופיים המחוברים ביניהם באמצעות תילים. **רכיב צירופי** (combinational element) הוא כל רכיב במעגל שיש לו מספר קבוע של קלטים ופלטים והוא מבצע פונקציה מוגדרת היטב. חלק מן הרכיבים שנעסוק בהם בפרק זה הם **רכיבים צירופיים בוליאניים** (boolean combinational elements) — הקלטים והפלטים שלהם לקוחים כולם מהקבוצה $\{0, 1\}$, כאשר 0 מייצג FALSE ו־1 מייצג TRUE.

רכיב צירופי בוליאני המחשב פונקציה בוליאנית פשוטה נקרא **שער לוגי** (logic gate). באיור 29.1 מוצגים ארבעת השערים הלוגיים הבסיסיים שישמשו כרכיבים צירופיים בפרק זה: **שער NOT** (או **מהפך**, inverter), **שער AND**, **שער OR** ו**שער XOR**. (כמו כן יוצגו שני שערים לוגיים נוספים — **שער NAND** ו**שער NOR** — הנדרשים באחדים מן התרגילים.) שער NOT מקבל **קלט** בינרי יחיד x, שערכו 0 או 1, ומפיק **פלט** בינרי z שערכו הפוך לערך הקלט. כל אחד משלושת השערים האחרים מקבל שני קלטים בינריים, x ו־y, ומפיק פלט בינרי יחיד z.

את פעולתו של כל שער, וגם של כל רכיב צירופי בוליאני אחר, ניתן לתאר באמצעות **טבלת אמת** (truth table). באיור 29.1 מוצגת טבלת האמת של כל שער מתחת לשער עצמו. טבלת אמת נותנת את הפלטים של הרכיב הצירופי עבור כל צירוף אפשרי של קלטים. לדוגמה, טבלת האמת של שער XOR מראה כי כאשר הקלטים הם $x = 0$ ו־$y = 1$, ערך הפלט הוא $z = 1$; שער זה מחשב את ה"או המוציא" (exclusive OR) של שני קלטיו. נסמן את הפונקציה NOT על־ידי $\neg$, את הפונקציה AND על־ידי $\wedge$, את הפונקציה OR על־ידי $\vee$ ואת הפונקציה XOR על־ידי $\oplus$. כך, למשל, $0 \oplus 1 = 1$.



איור 29.1 שישה שערים לוגיים בסיסיים עם קלטים ופלטים בינריים. מתחת לכל שער מוצגת טבלת האמת המתארת את פעולתו. **(i)** שער NOT. **(ii)** שער AND. **(iii)** שער OR. **(iv)** שער XOR (exclusive-OR). **(v)** שער NAND (NOT-AND). **(vi)** שער NOR (NOT-OR).

רכיבים צירופיים במעגלים ממשיים אינם פועלים בן רגע. לאחר שערכי הקלט הנכנסים אל הרכיב הצירופי **מתייצבים** (stabilize) — דהיינו נשארים יציבים וקבועים במשך זמן ארוך דיו — מובטח שהערך של פלט הרכיב יתייצב בערכו הנכון פרק זמן קבוע לאחר־מכן. הפרש זמנים זה נקרא **זמן ההשהיה** (propagation delay) של הרכיב. בפרק זה נניח שכל הרכיבים הצירופיים הם בעלי זמן השהיה קבוע.

מעגלים צירופיים

מעגל צירופי (combinational circuit) הוא מעגל המורכב מאחד או יותר רכיבים צירופיים המחוברים ביניהם בצורה לא־מעגלית (acyclic), כלומר, באופן שאינם מכילים מסלולים מעגליים. החיבורים נקראים **תילים** (wires). תיל יכול לחבר את הפלט של רכיב אחד לקלט של רכיב אחר, וכך להזין את ערך הפלט של הרכיב הראשון כערך קלט לרכיב השני. אף שתיל יחיד אינו יכול להיות מחובר ליותר מאשר פלט אחד של רכיב צירופי, הוא יכול להזין קלטים לכמה רכיבים. מספר הקלטים לרכיבים שמזין תיל נקרא **פרישת היציאה** (fan-out) של התיל. תיל שאינו מחובר לאף פלט של רכיב הוא **קלט למעגל** (circuit input), המקבל ערכי קלט ממקור חיצוני. תיל שאינו מחובר לאף קלט של רכיב הוא **פלט של המעגל** (circuit output), המעביר את תוצאות פעולתו של המעגל לעולם החיצוני. (תיל פנימי עשוי להתפרש גם החוצה לפלט של המעגל.) מעגלים צירופיים אינם מכילים מסלולים מעגליים ואינם מכילים רכיבי זיכרון (כגון האוגרים המתוארים בסעיף 29.4).

סוכמים מלאים

כדוגמה, מוצג באיור 29.2 מעגל צירופי הנקרא **סוכם מלא** (full adder). מעגל זה מקבל כקלט שלוש סיביות x, y ו־z, ומפיק כפלט שתי סיביות s ו־c, על פי טבלת האמת הבאה:

x	y	z	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

הפלט s הוא **הזוגיות** (parity) של סיביות הקלט:

$$s = \text{parity}(x, y, z) = x \oplus y \oplus z \tag{29.1}$$

והפלט c הוא **הרוב** (majority) של סיביות הקלט:

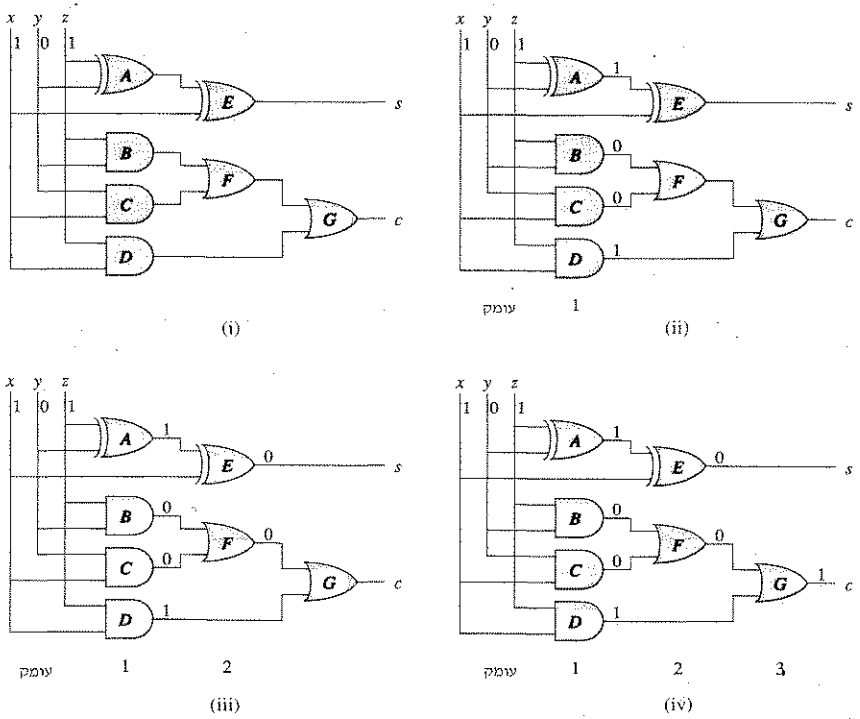
$$c = \text{majority}(x, y, z) = (x \wedge y) \vee (y \wedge z) \vee (x \wedge z) \tag{29.2}$$

(באופן כללי, פונקציות הזוגיות והרוב יכולות לקבל כקלט כל מספר של סיביות. הזוגיות היא 1 אם ורק אם מספר סיביות הקלט שערכן 1 הוא אי־זוגי. פונקציית הרוב מקבלת את הערך 1 אם ורק אם יותר ממחצית סיביות הקלט ערכן 1.) נשים לב כי אם מצרפים את הסיביות c ו־s מקבלים את הסכום של x, y ו־z. לדוגמה, אם $x = 1$, $y = 0$ ו־$z = 1$, אזי $\langle c, s \rangle = \langle 10 \rangle$,[1] שהוא הייצוג הבינרי של 2, הסכום של x, y ו־z.

1. מטעמי בהירות, נשמיט את הפסיקים בין איברי סדרה כאשר האיברים הם סיביות.

כל אחד מן הקלטים x, y ו־z לסוכם המלא הוא בעל פרישת יציאה של 3. כאשר הפעולה המבוצעת על ידי רכיב צירופי היא קומוטטיבית ואסוציאטיבית ביחס לקלטיו (כמו למשל הפונקציות AND, OR ו־XOR), אנו קוראים למספר הקלטים שלו **פרישת הכניסה** (fan-in) של הרכיב. אף כי פרישת הכניסה של כל שער באיור 29.2 היא 2, היינו יכולים לשרטט מחדש את הסוכם המלא כך ששערי ה־A XOR ו־E יוחלפו בשער XOR יחיד בעל 3 קלטים, ושערי ה־OR F ו־G יוחלפו בשער OR יחיד בעל 3 קלטים.

כדי לבחון כיצד פועל הסוכם המלא, נניח שכל שער מבצע את פעולתו ביחידת זמן אחת. באיור 29.2(i) מוצגת קבוצת קלטים ההופכים להיות יציבים בזמן 0. בזמן זה, ערכי כל הקלטים לשערים A–D, אך לא לשום שער אחר, הם יציבים, ולכן שערים אלה מפיקים את הערכים המוצגים באיור 29.2(ii) בזמן 1. נשים לב לכך שהשערים A–D פועלים במקביל. השערים E ו־F, אולם לא השער G, הם בעלי קלטים יציבים בזמן 1, והם מפיקים את הערכים המוצגים באיור 29.2(iii) בזמן 2. הפלט משער E הוא הסיבית s, כך שהפלט s של הסוכם המלא מוכן בזמן 2. אבל הפלט c עדיין אינו מוכן. לבסוף, שער G מקבל קלטים יציבים בזמן 2, ומפיק את הפלט c המוצג באיור 29.2(iv) בזמן 3.



איור 29.2 מעגל סוכם מלא. **(i)** בזמן 0, סיביות הקלט המוצגות מופיעות על שלושת תילי הקלט. **(ii)** בזמן 1, הערכים המוצגים מופיעים על הפלטים של השערים A-D הנמצאים בעומק 1. **(iii)** בזמן 2, הערכים המוצגים מופיעים על הפלטים של השערים E ו־F הנמצאים בעומק 2. **(iv)** בזמן 3, השער G מפיק את הפלט שלו, שהוא גם פלט המעגל.

עומק המעגל

כמו במקרה של רשתות ההשוואה שנדונו בפרק 28, אנו מודדים את זמן ההשהיה של מעגל צירופי במונחים של המספר הגדול ביותר של רכיבים צירופיים על מסלול כלשהו מן הקלטים אל הפלטים. בפרט, אנו מגדירים את **העומק** (depth) של מעגל, המתאים ל"זמן הריצה" שלו במקרה הגרוע ביותר, באופן אינדוקטיבי במונחי עומקיהם של התילים המרכיבים את המעגל. עומקו של תיל קלט הוא 0. אם רכיב צירופי הוא בעל הקלטים $x_1, x_2, \ldots, x_n$ שעומקיהם $d_1, d_2, \ldots, d_n$ בהתאמה, אזי עומקם של פלטי הרכיב הוא $\max\{d_1, d_2, \ldots, d_n\} + 1$. עומקו של רכיב צירופי הוא עומקם של פלטיו. עומקו של מעגל צירופי הוא העומק המקסימלי של רכיב צירופי כלשהו בו. מכיוון שאיננו מרשים שמעגלים צירופיים יכילו מסלולים מעגליים, המושגים השונים של עומק מוגדרים היטב.

אם לכל רכיב צירופי דרוש זמן קבוע כדי לחשב את ערכי הפלט שלו, אזי במקרה הגרוע, זמן ההשהיה של מעגל צירופי נמצא ביחס ישר לעומק המעגל. איור 29.2 מציג את עומקו של כל שער בסוכם המלא. מאחר שהשער בעל העומק הגדול ביותר הוא השער G, הרי שהסוכם המלא עצמו הוא בעל עומק 3, ועומק זה נמצא ביחס ישר לזמן הדרוש למעגל לבצע את פעולתו במקרה הגרוע ביותר.

מעגל צירופי יכול לפעמים לבצע חישוב מהר יותר מן הצפוי על פי עומקו. נניח שהפלט של תת־מעגל גדול מוזן לתוך אחד הקלטים של שער AND בעל 2 קלטים, אבל הקלט האחר של שער ה־AND הוא בעל ערך 0. אזי הפלט של השער יהיה 0, בלי תלות בערך הקלט המגיע מהתת־מעגל הגדול. אולם במקרה הכללי, איננו יכולים להסתמך על כך שיוזנו למעגל קלטים ספציפיים, ולכן ההפשטה של העומק כ"זמן הריצה" של המעגל סבירה למדי.

גודל המעגל

כאשר אנו מתכננים מעגלים, אנו מעוניינים בדרך כלל להקטין ככל האפשר לא רק את עומק המעגל אלא גם משאב נוסף — גודל המעגל. **הגודל** (size) של מעגל צירופי הוא מספר הרכיבים הצירופיים שמכיל המעגל. אינטואיטיבית, גודל המעגל מתאים למקום שתופס אלגוריתם בזיכרון המחשב. לדוגמה, גודלו של הסוכם המלא שבאיור 29.2 הוא 7, שכן הוא מכיל 7 שערים.

הגדרה זו של גודל המעגל אינה שימושית במיוחד עבור מעגלים קטנים. אחרי ככלות הכל, סוכם מלא הוא בעל מספר קבוע של קלטים ופלטים, והוא מחשב פונקציה מוגדרת היטב, ולכן הוא מקיים את ההגדרה של רכיב צירופי. סוכם מלא הבנוי מרכיב צירופי יחיד המהווה סוכם מלא, הוא אפוא בעל גודל 1. למעשה, על פי הגדרה זו, גודלו של כל רכיב צירופי הוא 1.

על ההגדרה של גודל מעגל להתאים למשפחות של מעגלים המחשבים פונקציות דומות. לדוגמה, מיד נראה מעגל חיבור המקבל שני קלטים בני n סיביות. למעשה, אין מדובר כאן במעגל יחיד, אלא במשפחה של מעגלים — אחד עבור כל גודל של קלט. בהקשר זה יש טעם בהגדרת גודל המעגל שהובאה לעיל, היא מאפשרת לנו להגדיר רכיבי מעגל נוחים בלי שגודלו של מימוש כלשהו של המעגל יושפע יותר מאשר פי גורם קבוע. כמובן, ביישומים מעשיים, חישובי הגודל מסובכים הרבה יותר, שכן יש להביא בחשבון לא רק את הרכיבים הצירופיים שנבחרו אלא גם שיקולים כגון השטח שדורש המעגל כאשר בונים אותו לתוך שבב צורן.

תרגילים

29.1-1
באיור 29.2, שנה את הקלט y ל־1. הצג את הערכים המתקבלים על התילים.

29.1-2
הראה כיצד ניתן לבנות מ־$n-1$ שערי XOR מעגל זוגיות בן n קלטים שעומקו $\lceil \lg n \rceil$.

29.1-3
הראה שניתן לבנות כל רכיב צירופי בוליאני ממספר קבוע של שערי AND, OR ו־NOT. (**רמז:** ממש את טבלת האמת של הרכיב.)

29.1-4
הראה שניתן לבנות כל פונקציה בוליאנית משערי NAND בלבד.

29.1-5
בנה מעגל צירופי המבצע את פונקציית האו־המוציא (XOR) באמצעות ארבעה שערי NAND בני 2 קלטים בלבד.

29.1-6
יהי C מעגל צירופי בן n קלטים ו־n פלטים שעומקו d. אם מחברים שני עותקים של C, כך שהפלטים של עותק אחד מוזנים ישירות כקלטים לעותק השני, מהו העומק המקסימלי האפשרי של המעגל שנוצר? מהו העומק המינימלי האפשרי?

29.2 מעגלי חיבור

עתה נחקור את הבעיה של חיבור מספרים המיוצגים בכתיב בינרי. נציג שלושה מעגלים צירופיים עבור בעיה זו. תחילה נבחן חיבור מגלגל־נשא, שבאמצעותו ניתן לחבר שני מספרים בני n סיביות בזמן $\Theta(n)$ באמצעות מעגל שגודלו $\Theta(n)$. ניתן לשפר את חסם הזמן הזה ל־$O(\lg n)$ באמצעות סוכם צופה־נשא, שגם גודלו $\Theta(n)$. לבסוף נציג חיבור שומר־נשא, אשר בזמן $O(1)$ עושה רדוקציה של חיבור שלושה מספרים בני n סיביות לחיבור מספר בן n סיביות ומספר בן $(n+1)$ סיביות. המעגל גודלו $\Theta(n)$.

29.2.1 חיבור מגלגל־נשא

נפתח בשיטה הרגילה לחיבור מספרים בינריים. אנו מניחים שמספר שלם אי־שלילי a מיוצג בייצוג בינרי על־ידי סדרה של n סיביות $\langle a_{n-1}, a_{n-2}, \ldots, a_0 \rangle$, כאשר $n \geq \lceil \lg(a+1) \rceil$ וערכו של a הוא:

$$\sum_{i=0}^{n-1} a_i 2^i$$

בהינתן שני מספרים שכל אחד מהם בן n סיביות, $a = \langle a_{n-1}, a_{n-2}, \ldots, a_0 \rangle$ ו־$b = \langle b_{n-1}, b_{n-2}, \ldots, b_0 \rangle$, ברצוננו לחשב את הסכום בן $(n+1)$ הסיביות $s = \langle s_n, s_{n-1}, \ldots, s_0 \rangle$. באיור 29.3 מודגם חיבורם של שני מספרים בני 8 סיביות. אנו מסכמים את העמודות מימין לשמאל, ומעבירים נשא מן העמודה ה־i לעמודה ה־$(i+1)$, עבור $i = 0, 1, \ldots, n-1$. בעמודה ה־i אנו לוקחים כקלט את הסיביות a_i, b_i ו**סיבית נשא־נכנס** c_i (carry-in bit), ומפיקים **סיבית סכום** s_i (sum bit) ו**סיבית נשא־יוצא** c_{i+1} (carry-out bit). סיבית הנשא־היוצא c_{i+1} המופקת מן העמודה ה־i היא סיבית הנשא־הנכנס הנקלטת על־ידי העמודה ה־$(i+1)$. מכיוון שאל העמודה ה־0 לא נכנס שום נשא, אנו מניחים כי $c_0 = 0$. הנשא היוצא c_n הוא הסיבית s_n של הסכום.

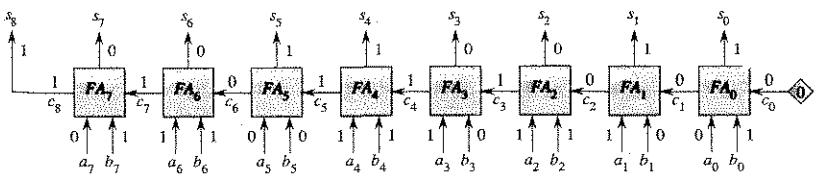
8	7	6	5	4	3	2	1	0		i
1	1	0	1	1	1	0	0	0	=	c
	0	1	0	1	1	1	1	0	=	a
	1	1	0	1	0	1	0	1	=	b
1	0	0	1	1	0	0	1	1	=	s

איור 29.3 חיבור שני מספרים בני 8 סיביות $\langle 01011110 \rangle = a$ ו־$\langle 11010101 \rangle = b$, לקבלת הסכום בן 9 הסיביות $\langle 100110011 \rangle = s$. כל סיבית c_i היא סיבית נשא. כל עמודת סיביות מייצגת, מלמעלה למטה, את c_i, a_i, b_i ו־s_i, עבור אינדקס i. סיבית הנשא הנכנס c_0 היא תמיד 0.

נשים לב כי כל סיבית סכום s_i היא הזוגיות של הסיביות a_i, b_i ו־c_i (ראה משוואה (29.1)). יתר על כן, סיבית הנשא־היוצא c_{i+1} היא הרוב של a_i, b_i ו־c_i (ראה משוואה (29.2)). לכן, כל שלב בחיבור ניתן לביצוע באמצעות סוכם מלא.

סוכם מגלגל־נשא (ripple-carry adder) בן n סיביות נוצר על־ידי חיבור שרשרת של n סוכמים מלאים, $FA_0, FA_1, \ldots, FA_{n-1}$, שבה הנשא היוצא c_{i+1} של FA_i מוזן ישירות כקלט הנשא הנכנס אל FA_{i+1}. באיור 29.4 מוצג סוכם מגלגל־נשא בן 8 סיביות. סיביות הנשא "מתגלגלות" מימין לשמאל. הנשא הנכנס c_0 לסוכם המלא FA_0 נקבע בחומרה ל־0, דהיינו, הוא תמיד 0, בלי תלות בערכים שמקבלים הקלטים האחרים. הפלט הוא המספר בן $(n+1)$ הסיביות $s = \langle s_n, s_{n-1}, \ldots, s_0 \rangle$, כאשר s_n שווה ל־c_n, סיבית הנשא היוצא מן הסוכם המלא FA_{n-1}.

מכיוון שסיביות הנשא "מתגלגלות" דרך כל n הסוכמים המלאים, הזמן הדרוש לסוכם מגלגל־נשא בן n סיביות הוא $\Theta(n)$. ביתר דיוק, הסוכם המלא FA_i נמצא בעומק $i+1$ במעגל. מכיוון ש־FA_{n-1} נמצא בעומק הרב ביותר מבין כל הסוכמים המלאים במעגל, עומקו של הסוכם מגלגל־הנשא כולו הוא n. גודל המעגל הוא $\Theta(n)$, משום שהוא מכיל n רכיבים צירופיים.



איור 29.4 סוכם מגלגל־נשא בן 8 סיביות מבצע את פעולת החיבור מאיור 29.3. ערכה של סיבית הנשא c_0 נקבע בחומרה ל־0 (באיור מצוין הדבר על־ידי המעוין), וסיביות הנשא "מתגלגלות" מימין לשמאל.

29.2.2 חיבור צופה־נשא

חיבור מגלגל־נשא דורש זמן של $\Theta(n)$ עקב גלגולן של סיביות הנשא דרך המעגל. בחיבור צופה־נשא מונעים את זמן ההשהיה הזה של $\Theta(n)$ על־ידי כך שמאיצים את חישובי סיביות הנשא, באמצעות מעגל דמוי עץ. סוכם צופה־נשא יכול לחבר שני מספרים בני n סיביות בזמן $O(\lg n)$.

המפתח הוא ההבחנה שבחיבור מגלגל־נשא, עבור $i \geq 1$, שניים מערכי הקלט של הסוכם המלא FA_i, קרי a_i ו־b_i, מוכנים זמן רב לפני שהנשא הנכנס c_i מוכן. הרעיון שביסודו של הסוכם צופה־הנשא הוא לנצל את המידע החלקי הזה.

כדוגמה, נניח כי $a_{i-1} = b_{i-1}$. מאחר שהנשא היוצא c_i הוא פונקציית הרוב, אנו מקבלים $c_i = a_{i-1} = b_{i-1}$ בלי *תלות בנשא הנכנס* c_{i-1}. אם $a_{i-1} = b_{i-1} = 0$, ניתן **להרוג** את הנשא היוצא c_i, כלומר, לאפס אותו, בלי לחכות לחישוב ערכו של c_{i-1}. באופן דומה, אם $a_{i-1} = b_{i-1} = 1$, ניתן **ליצור** את הנשא היוצא $c_i = 1$, בלי להתחשב בערכו של c_{i-1}.

אולם, אם $a_{i-1} \neq b_{i-1}$, אזי ערכו של c_i תלוי בערכו של c_{i-1}. ובפירוט, $c_i = c_{i-1}$, מכיוון ש"קולו" של הנשא הנכנס c_{i-1} הוא הקול המכריע בבחירות הרוב הקובעות את ערכו של c_i. במקרה זה אנו **מעבירים קדימה** (propagate) את הנשא כְּמות שהוא, שכן הנשא היוצא הוא הנשא הנכנס.

איור 29.5 מסכם קשרים אלה בין a_{i-1}, b_{i-1}, c_{i-1} ו־c_i במונחי **מצבי נשא** (carry statuses), כאשר k מציין "איפוס נשא" (kill), g מציין "יצירת נשא" (generate) ו־p מציין "קידום נשא" (propagate).

a_{i-1}	b_{i-1}	c_i	מצב נשא
0	0	0	k
0	1	c_{i-1}	p
1	0	c_{i-1}	p
1	1	1	k_g

איור 29.5 סיבית הנשא היוצא c_i ומצב הנשא המתאים לקלטים a_{i-1}, b_{i-1} ו־c_{i-1} של הסוכם המלא FA_{i-1} בחיבור מגלגל־נשא.

נתבונן ביחידה המורכבת משני סוכמים מלאים עוקבים FA_{i-1} ו־FA_i. הנשא הנכנס ליחידה הוא c_{i-1}, והנשא היוצא ממנה הוא c_{i+1}. ניתן לראות את היחידה המשולבת כמאפסת, יוצרת או מעבירה קדימה נשאים, כפי שעושה זאת סוכם מלא יחיד. היחידה המשולבת מאפסת את הנשא שלה אם FA_i מאפס את הנשא שלו, או אם FA_{i-1} מאפס את הנשא שלו ו־FA_i מעביר קדימה את שלו. באופן דומה, היחידה המשולבת יוצרת נשא אם FA_i יוצר נשא, או אם FA_{i-1} יוצר נשא ו־FA_i מעביר קדימה נשא. היחידה המשולבת מעבירה קדימה את הנשא, דהיינו, מציבה $c_{i+1} = c_{i-1}$, אם שני הסוכמים המלאים מעבירים קדימה את הנשאים שלהם. הטבלה באיור 29.6 מסכמת את הצירופים של מצבי הנשא המתקבלים ביחידה משולבת המורכבת משני סוכמים מלאים עוקבים. ניתן לראות טבלה זו כהגדרה של **אופרטור מצבי נשא** (carry status operator) $\otimes$ מעל התחום $\{k, p, g\}$. תכונה חשובה של אופרטור זה היא האסוציאטיביות, שאת הוכחת קיומה השארנו כתרגיל 29.2-2.

		FA_i		
	$\otimes$	k	p	g
	k	k	k	g
FA_{i-1}	p	k	p	g
	g	k	g	g

איור 29.6 מצבי הנשא של יחידה המורכבת מצירוף של שני סוכמים מלאים FA_{i-1} ו־FA_i, במונחי מצבי הנשא של כל אחד מהם, כפי שהם מתקבלים על ידי אופרטור מצבי הנשא $\otimes$ מעל התחום $\{\mathrm{k}, \mathrm{p}, \mathrm{g}\}$.

באמצעות אופרטור מצבי הנשא ניתן לבטא כל סיבית נשא c_i במונחי הקלטים. נתחיל בכך שנגדיר $x_0 = \mathrm{k}$, ועבור $i = 1, 2, \ldots, n$:

$$x_i = \begin{cases} \mathrm{k} & \text{אם } a_{i-1} = b_{i-1} = 0 \\ \mathrm{p} & \text{אם } a_{i-1} \neq b_{i-1} \\ \mathrm{g} & \text{אם } a_{i-1} = b_{i-1} = 1 \end{cases} \tag{29.3}$$

באופן זה, ערכו של x_i, עבור $i = 1, 2, \ldots, n$, הוא מצב הנשא הנתון באיור 29.5.

ערכו של הנשא c_i היוצא מסוכם מלא נתון עשוי להיות תלוי במצבי הנשא של כל הסוכמים המלאים FA_j עבור $j = 0, 1, \ldots, i-1$. הבה נגדיר $y_0 = x_0 = \mathrm{k}$, ועבור $i = 1, 2, \ldots, n$:

$$\begin{aligned} y_i &= y_{i-1} \otimes x_i \\ &= x_0 \otimes x_1 \otimes \cdots \otimes x_i \end{aligned} \tag{29.4}$$

נוכל לחשוב על y_i כעל "הרישא" של "המכפלה" $x_0 \otimes x_1 \otimes \cdots \otimes x_n$, את תהליך חישובם של הערכים $y_0, y_1, \ldots, y_n$ אנו מכנים **חישוב רישות** (prefix computation). (בפרק 30 נדון בחישובי רישות בהקשר מקבילי כללי יותר.) באיור 29.7 מוצגים הערכים x_i ו־y_i המתאימים לחיבור הבינרי שבאיור 29.3. הלמה הבאה מבהירה את חשיבותם של הערכים y_i בחיבור צופה־נשא.

i	8	7	6	5	4	3	2	1	0
a_i		0	1	0	1	1	1	1	0
b_i		1	1	0	1	0	1	0	1
x_i	p	g	k	g	p	g	p	p	k
y_i	g	g	k	g	g	g	k	k	k
c_i	1	1	0	1	1	1	0	0	0

איור 29.7 הערכים של x_i ו־y_i, עבור $i = 0, 1, \ldots, 8$, המתאימים לערכים של a_i, b_i ו־c_i בבעיית החיבור הבינרי שבאיור 29.3. כל ערך של x_i מוצל יחד עם הערכים של a_{i-1} ו־b_{i-1} שבהם הוא תלוי.

למה 29.1

נגדיר $x_0, x_1, \ldots, x_n$ ו־$y_0, y_1, \ldots, y_n$ על פי משוואות (29.3) ו־(29.4), בהתאמה. עבור $i = 1, 2, \ldots, n$, מתקיימים התנאים הבאים:

1. $y_i = \mathrm{k}$ גורר $c_i = 0$;
2. $y_i = \mathrm{g}$ גורר $c_i = 1$;
3. $y_i = \mathrm{p}$ אינו מתרחש אף פעם.

הוכחה ההוכחה היא באינדוקציה על i. עבור מקרה הבסיס, $i = 0$. לפי ההגדרה אנו מקבלים $y_0 = x_0 = \mathrm{k}$, וכן מתקיים $c_0 = 0$. להוכחת צעד האינדוקציה, נניח שהלמה מתקיימת עבור $i-1$. קיימים שלושה מקרים, בתלות בערכו של y_i:

1. אם $y_i = \mathrm{k}$, אזי מכיוון ש־$y_i = y_{i-1} \otimes x_i$, הרי שעל פי הגדרת אופרטור מצבי הנשא $\otimes$ באיור 29.6, מתקיים אחד מהשניים: או $x_i = \mathrm{k}$ או $x_i = \mathrm{p}$ וגם $y_{i-1} = \mathrm{k}$. אם $x_i = \mathrm{k}$, אזי משוואה (29.3) גוררת כי $a_{i-1} = b_{i-1} = 0$, ולכן $c_i = \mathrm{majority}(a_{i-1}, b_{i-1}, c_{i-1}) = 0$. אם $x_i = \mathrm{p}$ וגם $y_{i-1} = \mathrm{k}$, אזי $a_{i-1} \neq b_{i-1}$, וגם, על פי הנחת האינדוקציה, $c_{i-1} = 0$. לכן, $c_i = \mathrm{majority}(a_{i-1}, b_{i-1}, c_{i-1}) = 0$, כלומר, $c_i = 0$.
2. אם $y_i = \mathrm{g}$, אזי או שמתקיים $x_i = \mathrm{g}$ או שמתקיים $x_i = \mathrm{p}$ וגם $y_{i-1} = \mathrm{g}$. אם $x_i = \mathrm{g}$, אזי $a_{i-1} = b_{i-1} = 1$, ומכאן $c_i = 1$. אם $x_i = \mathrm{p}$ וגם $y_{i-1} = \mathrm{g}$, אזי $a_{i-1} \neq b_{i-1}$, וגם, על פי הנחת האינדוקציה, $c_{i-1} = 1$, ומכאן $c_i = 1$.
3. אם $y_i = \mathrm{p}$, אזי איור 29.6 גורר כי $y_{i-1} = \mathrm{p}$, בסתירה להנחת האינדוקציה. ■

מלמה 29.1 נובע שניתן לחשב כל סיבית נשא c_i על־ידי חישוב מצב הנשא y_i. ברגע שחישבנו את כל סיביות הנשא, אנו יכולים לחשב את הסכום כולו בזמן $\Theta(1)$ על־ידי חישוב מקביל של סיביות הסכום $s_i = \mathrm{parity}(a_i, b_i, c_i)$ עבור $i = 0, 1, \ldots, n$ (כאשר $a_n = b_n = 0$). בעיית החיבור המהיר של שני מספרים מצטמצמת אפוא לבעיית חישוב הרישות של מצבי הנשא $y_0, y_1, \ldots, y_n$.

חישוב מצבי נשא באמצעות מעגל רישות מקבילי

באמצעות מעגל רישות הפועל במקביל, בניגוד למעגל מגלגל־נשא המפיק את פלטיו בזה אחר זה, ניתן לחשב את כל n מצבי הנשא $y_0, y_1, \ldots, y_n$ במהירות רבה יותר. ובפירוט, נבנה מעגל רישות מקבילי שעומקו $O(\lg n)$. גודלו של המעגל יהיה $\Theta(n)$ — אסימפטוטית אותה כמות של חומרה כמו סוכם מגלגל־נשא.

לפני שנתחיל בבניית מעגל הרישות המקבילי, נציג סימון שיעזור לנו להבין את פעולתו של המעגל. עבור שני שלמים i ו־j בתחום $0 \leq i \leq j \leq n$, נגדיר:

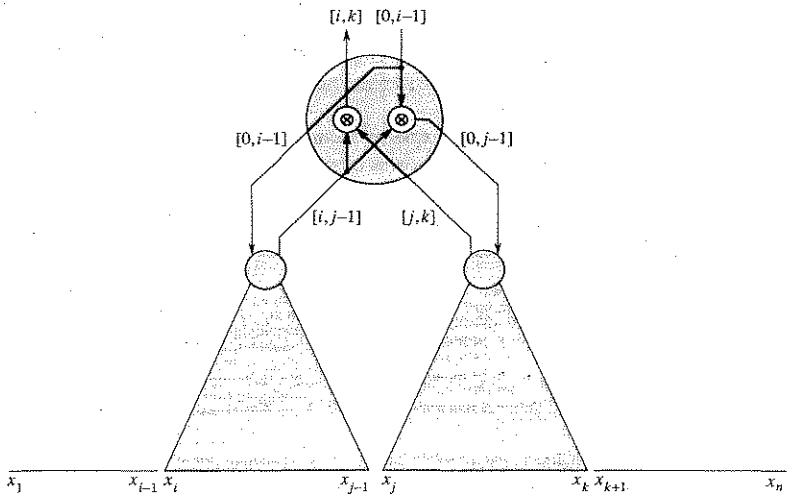
$$[i, j] = x_i \otimes x_{i+1} \otimes \cdots \otimes x_j$$

אם כך, $[i, i] = x_i$ עבור $i = 0, 1, \ldots, n$, שכן ההרכבה של מצב נשא אחד x_i מותירה אותו כמות שהוא. כמו כן, עבור i, j ו־k בתחום $0 \leq i < j \leq k \leq n$, מתקיימת הזהות:

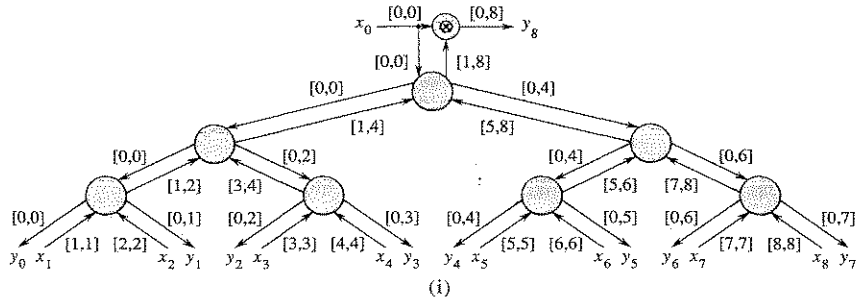
$$[i, k] = [i, j-1] \otimes [j, k] \tag{29.5}$$

שכן אופרטור מצבי הנשא הוא אסוציאטיבי. במונחי סימונים אלה, מטרת חישוב הרישות היא לחשב את $y_i = [0, i]$ עבור $i = 0, 1, \ldots, n$.

הרכיב הצירופי היחיד שמשתמשים בו במעגל הרישות המקבילי הוא מעגל המחשב את האופרטור $\otimes$. איור 29.8 מראה כיצד זוגות של רכיבי $\otimes$ מאורגנים ליצירת הצמתים הפנימיים של עץ בינרי שלם, ואיור 29.9 מציג את מעגל הרישות המקבילי עבור $n = 8$. נשים לב כי התילים במעגל יוצרים מבנה עץ, אולם המעגל עצמו אינו עץ, למרות שהוא מעגל צירופי טהור. הקלטים $x_1, x_2, \ldots, x_n$ מוזנים בעלים, והקלט x_0 מוזן בשורש. הפלטים $y_0, y_1, \ldots, y_{n-1}$ נוצרים בעלים, והפלט y_n — בשורש. (כדי להקל את הבנת חישוב הרישות, האינדקסים של המשתנים באיורים 29.8 ו־29.9 עולים משמאל לימין, ולא מימין לשמאל כמו בשאר האיורים בספר זה.)



איור 29.8 הארגון של מעגל רישות מקבילי. הצומת המוצג הוא שורשו של תת־עץ שהעלים שלו מקבלים את הקלטים x_i עד x_k. התת־עץ השמאלי של צומת זה מטפל בקלטים x_i עד x_{j-1}, והתת־העץ הימני שלו מטפל בקלטים x_j עד x_k. הצומת מורכב משני רכיבי $\otimes$, הפועלים בזמנים שונים במהלך פעולתו של המעגל. רכיב אחד מחשב את $[i, j-1] \otimes [j, k] \leftarrow [i, k]$, והרכיב האחר מחשב את $[0, i-1] \otimes [i, j-1] \leftarrow [0, j-1]$. הערכים המחושבים רשומים ליד התילים.



איור 29.9 מעגל רישות מקבילי עבור $n = 8$. **(i)** המבנה הכללי של המעגל והערכים הנישאים על כל תיל. **(ii)** אותו מעגל עם ערכים המתאימים לאיורים 29.3 ו־29.7.

שני רכיבי האופרטור $\otimes$ שבכל צומת פועלים בדרך כלל בזמנים שונים והם בעלי עומקים שונים במעגל. כפי שניתן לראות באיור 29.8, אם התת־עץ המושרש בצומת נתון מטפל בתחום מסוים, $x_i, x_{i+1}, \ldots, x_k$, של קלטים, התת־עץ השמאלי שלו מטפל בתחום $x_i, x_{i+1}, \ldots, x_{j-1}$, והתת־עץ הימני שלו מטפל בתחום $x_j, x_{j+1}, \ldots, x_k$, אזי הצומת חייב להפיק עבור אביו את המכפלה $[i, k]$ של כל הקלטים המטופלים על־ידי התת־עץ שלו. מאחר שנוכל להניח באינדוקציה כי בנו השמאלי ובנו הימני של הצומת מפיקים את המכפלות $[i, j-1]$ ו־$[j, k]$, הרי שהצומת פשוט משתמש באחד משני רכיביו לחישוב $[i, j-1] \otimes [j, k] \leftarrow [i, k]$.

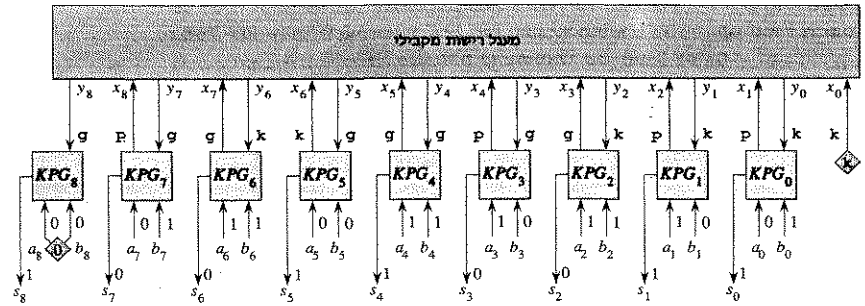
זמן מה לאחר שלב זה של חישוב כלפי מעלה, הצומת מקבל מאביו את המכפלה $[0, i-1]$ של כל הקלטים שהופיעו לפני הקלט השמאלי ביותר, x_i, שבו מטפל תת־עץ של צומת זה. עתה הצומת מחשב באותו אופן ערכים עבור בניו. הקלט השמאלי ביותר המטופל על־ידי הבן השמאלי של הצומת גם הוא x_i, ולכן הצומת מעביר לבן השמאלי את הערך $[0, i-1]$ ללא שינוי. הקלט השמאלי ביותר המטופל על־ידי הבן הימני של הצומת הוא x_j, ולכן הצומת חייב לחשב את $[0, j-1]$. מכיוון שהצומת מקבל את הערך $[0, i-1]$ מאביו ואת הערך $[i, j-1]$ מבנו השמאלי, הוא פשוט מחשב את $[i, j-1] \otimes [0, i-1] \leftarrow [0, j-1]$, ושולח את הערך הזה לבנו הימני.

באיור 29.9 מוצג המעגל המתקבל, כולל מקרה הגבול המתרחש בשורש. אל השורש מוזן כקלט הערך $x_0 = [0, 0]$, ורכיב $\otimes$ אחד נוסף משמש לחישוב הערך $y_n = [0, n] = [0, 0] \otimes [1, n]$ (במקרה הכללי).

אם n הוא חזקה מדויקת של 2, אזי מעגל הרישות המקבילי מכיל $2n - 1$ רכיבי $\otimes$. חישוב כל $n + 1$ הרישות מתבצע בזמן $O(\lg n)$ בלבד, שכן החישוב מתקדם במעלה העץ ואז בחזרה במורדו. תרגיל 29.2-5 חוקר ביתר פירוט את עומקו של המעגל.

השלמת בנייתו של הסוכם צופה־הנשא

עתה, כשיש בידינו מעגל רישות מקבילי, אנו יכולים להשלים את תיאורו של הסוכם צופה־הנשא. איור 29.10 מתאר את מבנהו. **סוכם צופה־נשא** (carry-lookahead adder) בן n סיביות מורכב מ־$(n + 1)$ **קופסאות KPG**. כל אחת בגודל $\Theta(1)$, וממעגל רישות מקבילי שקלטיו הם $x_0, x_1, \ldots, x_n$ (ערכו של x_0 נקבע בחומרה כ־k), ופלטיו הם $y_0, y_1, \ldots, y_n$. קופסת ה־KPG ה־i, כלומר, KPG_i, מקבלת את הקלטים החיצוניים a_i ו־b_i ומפיקה את סיבית הסכום s_i. (ערכיהן של סיביות הקלט a_n ו־b_n נקבעים בחומרה כ־0.) בהינתן a_{i-1} ו־b_{i-1}, הקופסה KPG_{i-1} מחשבת את $x_i \in \{\mathrm{k}, \mathrm{p}, \mathrm{g}\}$ על פי משוואה (29.3) ושולחת את התוצאה כקלט החיצוני x_i של מעגל הרישות המקבילי. (המעגל מתעלם מן הערך של x_{n+1}.) חישוב כל הערכים x_i מתבצע בזמן $\Theta(1)$. לאחר השהיה של $O(\lg n)$, מעגל הרישות המקבילי מפיק את $y_0, y_1, \ldots, y_n$. על פי למה 29.1, ערכו של y_i הוא k או g; הוא אינו יכול לקבל את הערך p. כל ערך y_i מציין איזה נשא נכנס לסוכם המלא FA_i בסוכם מגלגל־הנשא: $y_i = \mathrm{k}$ גורר $c_i = 0$, ו־$y_i = \mathrm{g}$ גורר $c_i = 1$. לכן, ערכו של y_i מוזן ל־KPG_i לקביעת ערכו של הנשא הנכנס c_i, וסיבית הסכום $s_i = \mathrm{parity}(a_i, b_i, c_i)$ מחושבת בזמן קבוע. הסוכם צופה־הנשא פועל אפוא בזמן $O(\lg n)$ וגודלו $\Theta(n)$.



איור 29.10 בנייתו של סוכם צופה־נשא בן n סיביות, עבור $n = 8$. הסוכם מורכב מ־$n + 1$ קופסאות KPG. כל קופסה KPG_i, עבור $i = 0, 1, \ldots, n$, מקבלת את הקלטים החיצוניים a_i ו־b_i (כאשר ערכיהם של a_n ו־b_n נקבעים על־ידי החומרה כ־0, כפי שמציין המעוין שבאיור) ומחשבת את מצב הנשא x_{i+1}. ערכים אלה מוזנים למעגל רישות מקבילי, המחזיר את התוצאות y_i של חישובי הרישות. עתה, כל קופסה KPG_i מקבלת את y_i כקלט, מפרשת אותו כסיבית הנשא הנכנס c_i, ואז מפיקה את סיבית הסכום $s_i = \text{parity}(a_i, b_i, c_i)$. הערכים המופיעים באיור מקבילים לאלה שבאיורים 29.3 ו־29.7.

29.2.3 חיבור שומר־נשא

סוכם צופה־נשא יכול לחבר שני מספרים בני n סיביות בזמן $O(\lg n)$. מפתיע אולי שחיבור שלושה מספרים בני n סיביות צורך תוספת זמן קבועה בלבד. "הטריק" הוא לעשות רדוקציה של בעיית החיבור של שלושה מספרים לבעיית החיבור של שני מספרים בלבד.

בהינתן שלושה מספרים בני n סיביות:

$$x = \langle x_{n-1}, x_{n-2}, \ldots, x_0 \rangle, \quad y = \langle y_{n-1}, y_{n-2}, \ldots, y_0 \rangle,$$
$$z = \langle z_{n-1}, z_{n-2}, \ldots, z_0 \rangle$$

סוכם שומר־נשא (carry-save adder) בן n סיביות מפיק מספר בן n סיביות $u = \langle u_{n-1}, u_{n-2}, \ldots, u_0 \rangle$, ומספר בן $(n+1)$ סיביות $v = \langle v_n, v_{n-1}, \ldots, v_0 \rangle$ המקיימים:

$$u + v = x + y + z$$

כפי שניתן לראות באיור 29.11(i), הוא עושה זאת על־ידי כך שהוא מחשב עבור $i = 0, 1, \ldots, n-1$ את הערכים הבאים:

$$u_i = \text{parity}(x_i, y_i, z_i)$$
$$v_{i+1} = \text{majority}(x_i, y_i, z_i)$$

ערכה של הסיבית v_0 הוא תמיד 0.

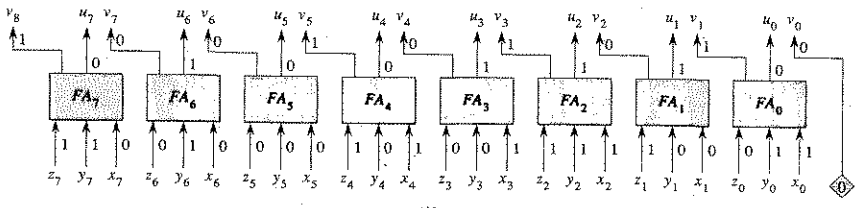
הסוכם שומר־הנשא בן n הסיביות המוצג באיור 29.11(ii) מורכב מ־n סוכמים מלאים $FA_0, FA_1, \ldots, FA_{n-1}$. עבור $i = 0, 1, \ldots, n-1$, הסוכם המלא FA_i מקבל את הקלטים x_i, y_i ו־z_i. סיבית הסכום המתקבלת כפלט מ־FA_i נלקחת כ־u_i, וסיבית הנשא היוצא מ־FA_i נלקחת כ־v_{i+1}. ערכה של הסיבית v_0 נקבע על־ידי החומרה כ־0.

מכיוון שהחישובים של כל $2n + 1$ סיביות הפלט הם בלתי־תלויים, ניתן לבצעם במקביל. לכן סוכם שומר־נשא פועל בזמן $\Theta(1)$ וגודלו $\Theta(n)$. כדי לסכם שלושה מספרים בני n סיביות אנו

צריכים אפוא רק לבצע חיבור שומר־נשא, האורך זמן $\Theta(1)$, ואחר־כך לבצע חיבור צופה־נשא, האורך זמן $O(\lg n)$. אף כי שיטה זו אינה טובה יותר אסימפטוטית מן השימוש בשני חיבורים צופי־נשא, היא מהירה בהרבה ביישומים מעשיים. יתירה מזו, בסעיף 29.3 נראה כי חיבור שומר־נשא ממלא תפקיד מרכזי באלגוריתמי כפל מהירים.

8	7	6	5	4	3	2	1	0	i	
	0	0	0	1	1	1	0	1	=	x
	1	1	0	0	0	1	0	1	=	y
	1	0	0	1	0	1	1	0	=	z
0	1	0	0	1	1	1	1	0	=	u
1	0	0	1	0	1	0	1	0	=	v

(i)



(ii)

איור 29.11 (i) חיבור שומר־נשא. בהינתן שלושה מספרים בני n סיביות x, y ו־z, אנו יוצרים מספר u בן n סיביות ומספר v בן $(n+1)$ סיביות המקיימים $x + y + z = u + v$. הזוג ה־i של סיביות מוצללות הוא פונקציה של x_i y_i ו־z_i. **(ii)** סוכם שומר־נשא בן 8 סיביות. כל סוכם מלא FA_i מקבל כקלט את הסיביות x_i, y_i ו־z_i ומפיק את סיבית הסכום u_i ואת סיבית הנשא היוצא v_{i+1}. ערכה של הסיבית v_0 נקבע בחומרה כ־0.

תרגילים

29.2-1

יהיו $a = \langle 01111111 \rangle$, $b = \langle 00000001 \rangle$, ו־$n = 8$. הצג את סיביות הסכום והנשא המתקבלות כפלט מן הסוכמים המלאים כאשר מבצעים חיבור מגלגל־נשא של שתי סדרות אלה. הצג את מצבי הנשא $x_8, x_7, \ldots, x_0$ המתאימים ל־a ו־b, רשום ליד כל תיל במעגל הרישות המקבילי מאיור 29.9 את הערך שהוא מסר לקלטים x_i אלה, והצג את הפלטים המתקבלים $y_8, y_7, \ldots, y_0$.

29.2-2

הוכח שאופרטור מצבי הנשא $\otimes$, הנתון על־ידי איור 29.5, הוא אסוציאטיבי.

29.2-3

הדגם את בנייתו של מעגל רישות מקבילי הפועל בזמן $O(\lg n)$ עבור ערכי n שאינם חזקות מדויקות של 2, על־ידי כך שתשרטט מעגל כזה עבור $n = 11$. אפיין את ביצועיהם של מעגלי רישות מקבילים הבנויים בצורת עצים בינריים שרירותיים.

29.2-4

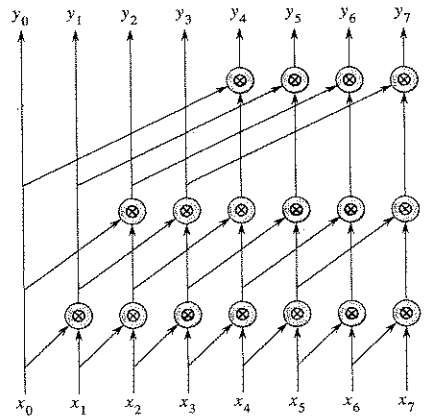
הצג את המבנה של קופסת KPG_i ברמת שערים. הנח שכל פלט x_i מיוצג על־ידי $\langle 00 \rangle$ אם $x_i = \text{k}$, על־ידי $\langle 11 \rangle$ אם $x_i = \text{g}$, ועל־ידי $\langle 01 \rangle$ או $\langle 10 \rangle$ אם $x_i = \text{p}$. כמו כן הנח שכל קלט y_i מיוצג על־ידי 0 אם $y_i = \text{k}$ ועל־ידי 1 אם $y_i = \text{g}$.

29.2-5

ציין ליד כל תיל במעגל הרישות המקבילי מאיור 29.9(i) את עומקו. **מסלול קריטי** (critical path) במעגל הוא מסלול בעל המספר הגדול ביותר של רכיבים צירופיים מבין כל המסלולים מן הקלטים אל הפלטים. זהה את המסלול הקריטי באיור 29.9(i), והראה שאורכו $O(\lg n)$. הראה שקיים צומת אשר רכיבי $\otimes$ שלו פועלים בהפרש זמן של $\Theta(\lg n)$. האם קיים צומת אשר רכיבי $\otimes$ שלו פועלים בו־זמנית?

29.2-6

הצג תרשים מלבנים רקורסיבי של המעגל מאיור 29.12 עבור מספר קלטים כלשהו n שהוא חזקה מדויקת של 2. התבסס על תרשים המלבנים שלך כדי להראות שהמעגל אכן מבצע חישוב רישות. הראה שעומקו של המעגל הוא $\Theta(\lg n)$ וגודלו $\Theta(n \lg n)$.



איור 29.12 מעגל רישות מקבילי עבור תרגיל 29.2-6.

29.2-7

מהי פרישת היציאה המקסימלית של תיל כלשהו בסוכם צופה־נשאז הראה שגם אם מגבילים את פרישת היציאה של שערים ל־$O(1)$, עדיין ניתן לבצע חיבור באמצעות מעגל בגודל $\Theta(n)$ בזמן $O(\lg n)$.

29.2-8

מעגל מנייה (tally circuit) הוא מעגל בעל n קלטים ביניריים ו־$m = \lceil \lg(n+1) \rceil$ פלטים. אם מפרשים את הפלטים כמספר בינרי, הם מייצגים את מספר ה־1־ים בקלטים. לדוגמה, אם הקלט הוא $\langle 10011110 \rangle$, הפלט הוא $\langle 101 \rangle$ והוא מציין שהקלט מכיל חמישה 1־ים. תאר מעגל מנייה בעל עומק $O(\lg n)$ שגודלו $\Theta(n)$.

* 29.2-9

הראה שניתן לבצע חיבור מספרים בני n סיביות באמצעות מעגל צירופי בעל עומק 4 וגודל פולינומיאלי ב־n אם מתירים בשערי AND ו־OR פרישת כניסה גבוהה כרצוננו. (**רשות:** הראה שניתן לעשות זאת באמצעות מעגל בעומק 3.)

* 29.2-10

נניח שמחברים באמצעות סוכם מגלגל־נשא שני מספרים אקראיים בני n סיביות, כאשר כל סיבית מקבלת את הערך 0 או 1 באופן בלתי תלוי בהסתברות שווה. הראה שהסתברות של $1 - 1/n$ לפחות, לא יהיה נשא שיקודם יותר מ־$O(\lg n)$ שלבים עוקבים. במילים אחרות, למרות שעומקו של הסוכם מגלגל־הנשא הוא $\Theta(n)$, עבור שני מספרים אקראיים, הפלטים מתייצבים כמעט תמיד בתוך זמן $O(\lg n)$.

29.3 מעגלי כפל

אלגוריתם הכפל "של בית הספר היסודי" המוצג באיור 29.13 יכול לחשב את המכפלה בת $2n$ הסיביות $p = \langle p_{2n-1}, p_{2n-2}, \ldots, p_0 \rangle$ של שני המספרים בני n הסיביות $a = \langle a_{n-1}, a_{n-2}, \ldots, a_0 \rangle$ ו־$b = \langle b_{n-1}, b_{n-2}, \ldots, b_0 \rangle$. בוחנים את הסיביות של b מ־b_0 עד b_{n-1}. עבור כל סיבית b_i שערכה 1, מוסיפים את a למכפלה, אבל כשהוא מוזז i סיביות שמאלה. עבור כל סיבית b_i שערכה 0, מוסיפים למכפלה 0. אם נסמן $m^{(i)} = a \cdot b_i \cdot 2^i$, אנו מחשבים את:

$$p = a \cdot b = \sum_{i=0}^{n-1} m^{(i)}$$

כל איבר $m^{(i)}$ נקרא **מכפלה חלקית** (partial product). בחישוב המכפלה יש לסכם n מכפלות חלקיות, אשר סיביותיהן נמצאות במקומות 0 עד $2n-2$. הנשא היוצא מהסיבית הגבוהה ביותר הוא הסיבית הסופית הנמצאת במקום ה־$2n-1$.

בסעיף זה נבחן שני מעגלים לכפל שני מספרים בני n סיביות. כופלים מערכיים פועלים בזמן $\Theta(n)$ וגודלם $\Theta(n^2)$. כופלי עץ־וואלאס גם הם גודלם $\Theta(n^2)$, אולם הם פועלים בזמן $\Theta(\lg n)$. שני המעגלים מבוססים על אלגוריתם הכפל הנלמד בבית הספר היסודי.

				1	1	1	0	=	a
				1	1	0	1	=	b
				1	1	1	0	=	$m^{(0)}$
			0	0	0	0		=	$m^{(1)}$
		1	1	1	0			=	$m^{(2)}$
	1	1	1	0				=	$m^{(3)}$
1	0	1	1	0	1	1	0	=	p

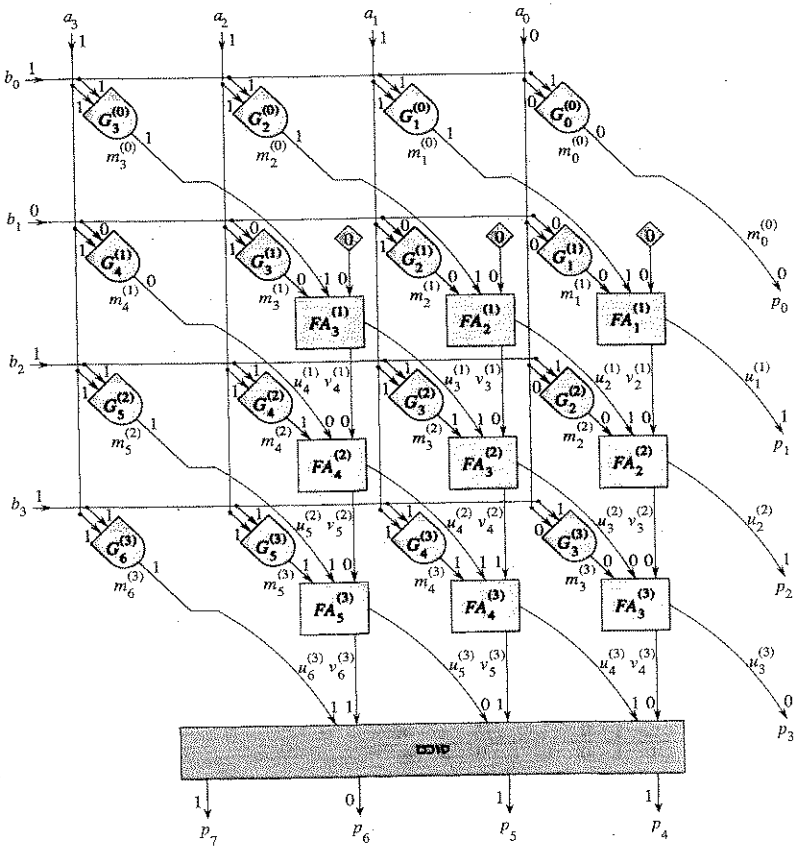
איור 29.13 שיטת הכפל "של בית הספר היסודי". באיור מוצג כפל של $a = \langle 1110 \rangle$ ו־$b = \langle 1101 \rangle$ לקבלת המכפלה $p = \langle 10110110 \rangle$. אנו מסכמים $\sum_{i=0}^{n-1} m^{(i)}$, כאשר $m^{(i)} = a \cdot b_i \cdot 2^i$. כאן, $n = 4$. כל איבר $m^{(i)}$ נוצר על־ידי הזזה של a (אם $b_i = 1$) או 0 (אם $b_i = 0$) i סיביות שמאלה. הסיביות שאינן מופיעות באיור ערכן 0, בלי תלות בערכיהם של a ושל b.

29.3.1 כופלים מערכיים

באופן עקרוני, כופל מערכי מורכב משלושה חלקים. החלק הראשון יוצר את המכפלות החלקיות. החלק השני מסכם את המכפלות החלקיות באמצעות סוכמים שומרי־נשא. לבסוף, החלק השלישי

מסכם את שני המספרים המתקבלים מן החיבורים שומרי־הנשא באמצעות סוכם מגלגל־נשא או צופה־נשא.

באיור 29.14 מוצג **כופל מערכי** (array multiplier) עבור שני מספרי קלט $a = \langle a_{n-1}, a_{n-2}, \ldots, a_0 \rangle$ ו־$b = \langle b_{n-1}, b_{n-2}, \ldots, b_0 \rangle$. הערכים a_j מתקדמים בכופל אנכית, מלמעלה למטה, והערכים b_i — אופקית, משמאל לימין. כל סיבית קלט נפרשת ל־n שערי AND ליצירת המכפלות החלקיות. סוכמים מלאים, המאורגנים כסוכמים שומרי־נשא, מסכמים את המכפלות החלקיות. הסיביות מסדר נמוך של המכפלה הסופית מוצאות לפלט מימין. הסיביות מסדר גבוה נוצרות על־ידי חיבור שני המספרים שפולט הסוכם שומר־הנשא האחרון.



איור 29.14 כופל מערכי המחשב את המכפלה $p = \langle p_{2n-1}, p_{2n-2}, \ldots, p_0 \rangle$ של שני המספרים בני n הסיביות $a = \langle a_{n-1}, a_{n-2}, \ldots, a_0 \rangle$ ו־$b = \langle b_{n-1}, b_{n-2}, \ldots, b_0 \rangle$. באיור מוצג כופל מערכי עבור $n = 4$. כל שער AND $G_j^{(i)}$ מחשב את סיבית המכפלה החלקית $m_j^{(i)}$ כל שורה של סוכמים מלאים מרכיבה סוכם שומר־נשא. n הסיביות הנמוכות של המכפלה הסופית הן $m_0^{(0)}$ וסיביות ה־u הנפלטות מהטור הימני ביותר של סוכמים מלאים. n הסיביות העליונות של המכפלה הסופית מתקבלות על־ידי סיכום סיביות ה־u וה־v הנפלטות משורת הסוכמים המלאים התחתונה. באיור מוצגים ערכי הסיביות עבור הקלטים $a = \langle 1110 \rangle$ ו־$b = \langle 1101 \rangle$ והמכפלה $p = \langle 10110110 \rangle$, המופיעים גם באיורים 29.13 ו־29.15.

הבה נבחן ביתר פירוט את מבנהו של הכופל המערכי. בהינתן שני מספרי הקלט $a = \langle a_{n-1}, a_{n-2}, \ldots, a_0 \rangle$ ו־$b = \langle b_{n-1}, b_{n-2}, \ldots, b_0 \rangle$, קל לחשב את סיביות המכפלות החלקיות. ובאופן מפורט, עבור $i, j = 0, 1, \ldots, n-1$, אנו מקבלים:

$$m_{j+i}^{(i)} = a_j \cdot b_i$$

מאחר שאת המכפלה של ערכים בני סיבית אחת ניתן לחשב ישירות באמצעות שער AND, ניתן להפיק את כל הסיביות של המכפלות החלקיות (פרט לאלה שידוע כי הן 0, שאותן אין צורך לחשב במפורש) בצעד אחד, תוך שימוש ב־n^2 שערי AND.

איור 29.15 מדגים כיצד הכופל המערכי מבצע את החיבורים שומרי־הנשא כאשר הוא מסכם את המכפלות החלקיות שבאיור 29.13. הכופל מתחיל בכך שהוא מבצע חיבור שומר־נשא של $m^{(0)}$, $m^{(1)}$ ו־0, לקבלת המספרים בני $(n+1)$ הסיביות $u^{(1)}$ ו־$v^{(1)}$. (המספר $v^{(1)}$ הוא בן $(n+1)$ סיביות בלבד, ולא $(n+2)$, מכיוון שהסיבית ה־$(n+1)$ של 0 וגם הסיבית ה־$(n+1)$ של $m^{(0)}$ הן 0.) לכן, $m^{(0)} + m^{(1)} = u^{(1)} + v^{(1)}$. לאחר־מכן, הכופל מבצע חיבור שומר־נשא של $u^{(1)}$, $v^{(1)}$ ו־$m^{(2)}$, ומקבל את המספרים בני $(n+2)$ הסיביות $u^{(2)}$ ו־$v^{(2)}$. (שוב, $v^{(2)}$ הוא בן $(n+2)$ סיביות בלבד מאחר שהסיביות $u_{n+2}^{(1)}$ ו־$v_{n+2}^{(1)}$ שתיהן 0.) אנו מקבלים אפוא $m^{(0)} + m^{(1)} + m^{(2)} = u^{(2)} + v^{(2)}$. הכופל ממשיך לבצע חיבורים שומרי־נשא של $u^{(i-1)}$, $v^{(i-1)}$ ו־$m^{(i)}$ עבור $i = 2, 3, \ldots, n-1$. התוצאה היא המספר בן $(2n-1)$ הסיביות $u^{(n-1)}$ והמספר בן $(2n-1)$ הסיביות $v^{(n-1)}$, המקיימים:

$$\begin{aligned} u^{(n-1)} + v^{(n-1)} &= \sum_{i=0}^{n-1} m^{(i)} \\ &= p \end{aligned}$$

למעשה, החיבורים שומרי־הנשא שבאיור 29.15 מתבצעים על יותר סיביות מהנדרש. נשים לב כי עבור $i = 0, 1, \ldots, n-1$ ו־$j = 0, 1, \ldots, i-1$, מתקיים $m_j^{(i)} = 0$, בשל האופן שבו אנו מזיזים את המכפלות החלקיות. עוד נשים לב כי עבור $i = 0, 1, \ldots, n-1$ ו־$j = 0, 1, \ldots, i, i+n, i+n+1, \ldots, 2n-1$, מתקיים $v_j^{(i)} = 0$ (ראה תרגיל 29.3-1). כל חיבור שומר־נשא צריך אפוא להתבצע על $n-1$ סיביות בלבד.

הבה נבחן עתה את הקשר בין הכופל המערכי לבין שיטת הביצוע החוזר של חיבורים שומרי־נשא. כל שער AND מסומן על־ידי $G_j^{(i)}$ עבור איזושהם i ו־j בתחומים $0 \le i \le n-1$ ו־$0 \le j \le 2n-2$. השער $G_j^{(i)}$ מפיק את $m_j^{(i)}$, הסיבית ה־j של המכפלה החלקית ה־i. עבור $i = 0, 1, \ldots, n-1$, השורה ה־i של שערי AND מחשבת את n הסיביות המשמעותיות של המכפלה החלקית $m^{(i)}$, דהיינו, את $\langle m_{n+i-1}^{(i)}, m_{n+i-2}^{(i)}, \ldots, m_i^{(i)} \rangle$.

פרט לסוכמים המלאים בשורה העליונה (דהיינו, עבור $i = 2, 3, \ldots, n-1$), כל סוכם מלא $FA_j^{(i)}$ מקבל שלוש סיביות קלט — $m_j^{(i)}$, $u_j^{(i-1)}$ ו־$v_j^{(i-1)}$ — ומפיק שתי סיביות פלט — $u_j^{(i)}$ ו־$v_{j+1}^{(i)}$. ניש לשים לב כי בטור השמאלי ביותר של הסוכמים המלאים מתקיים $u_{i+n-1}^{(i-1)} = m_{i+n-1}^{(i)}$, כל סוכם מלא $FA_j^{(1)}$ בשורה העליונה מקבל את הקלטים $m_j^{(1)}$, $m_j^{(0)}$ ו־0, ומפיק את הסיביות $u_j^{(1)}$ ו־$v_{j+1}^{(i)}$.

לבסוף, הבה נבחן את הפלט שמפיק הכופל המערכי. כאמור, $v_j^{(n-1)} = 0$ עבור $j = 0, 1, \ldots, n-1$. לכן, $p_j = u_j^{(n-1)}$ עבור $j = 0, 1, \ldots, n-1$. יתר על כן, מכיוון ש־$m_0^{(1)} = 0$, אזי $u_0^{(1)} = m_0^{(0)}$, ומכיוון ש־i הסיביות הפחות משמעותיות של כל $m^{(i)}$ ושל כל $v^{(i-1)}$ הן 0, אזי $u_j^{(i)} = u_j^{(i-1)}$ עבור $i = 2, 3, \ldots, n-1$ ו־$j = 0, 1, \ldots, i-1$. לכן, $p_0 = m_0^{(0)}$, ובאינדוקציה, $p_i = u_i^{(i)}$ עבור $i = 1, 2, \ldots, n-1$. את סיביות המכפלה

$\langle p_{2n-1}, p_{2n-2}, \ldots, p_n \rangle$ מפיק סוכם בן n סיביות אשר מסכם את הפלטים המתקבלים מהשורה האחרונה של הסוכמים המלאים:

$$\langle p_{2n-1}, p_{2n-2}, \ldots, p_n \rangle = \langle u_{2n-2}^{(n-1)}, u_{2n-3}^{(n-1)}, \ldots, u_n^{(n-1)} \rangle + \langle v_{2n-2}^{(n-1)}, v_{2n-3}^{(n-1)}, \ldots, v_n^{(n-1)} \rangle$$

$$\begin{array}{cccccccccc}
 & & & 0 & 0 & 0 & 0 & = & 0 & \\
 & & & 1 & 1 & 1 & 0 & = & m^{(0)} & \\
 & & 0 & 0 & 0 & 0 & & = & m^{(1)} & \\
\hline
 & & 0 & 1 & 1 & 1 & 0 & = & u^{(1)} & \\
 & & 0 & 0 & 0 & & & = & v^{(1)} & \\
 & 1 & 1 & 1 & 0 & & & = & m^{(2)} & \\
\hline
 & 1 & 1 & 0 & 1 & 1 & 0 & = & u^{(2)} & \\
 & 0 & 1 & 0 & & & & = & v^{(2)} & \\
1 & 1 & 1 & 0 & & & & = & m^{(3)} & \\
\hline
1 & 0 & 1 & 0 & 1 & 1 & 0 & = & u^{(3)} & \\
1 & 1 & 0 & & & & & = & v^{(3)} & \\
\hline
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & = & p
\end{array}$$

איור 29.15 סיכום המכפלות החלקיות באמצעות חיבורים שומרי־נשא חוזרים. בדוגמה זו, $a = \langle 1110 \rangle$ ו־$b = \langle 1101 \rangle$. הסיביות במקומות הריקים הן תמיד 0, בלי תלות בערכי a ו־b. תחילה מוצאים את $m^{(0)} + m^{(1)} + 0 = u^{(1)} + v^{(1)}$, לאחר־מכן את $u^{(1)} + v^{(1)} + m^{(2)} = u^{(2)} + v^{(2)}$, אחר־כך את $u^{(2)} + v^{(2)} + m^{(3)} = u^{(3)} + v^{(3)}$ ולבסוף את $p = m^{(0)} + m^{(1)} + m^{(2)} + m^{(3)} = u^{(3)} + v^{(3)}$. יש לשים לב כי $p_0 = m_0^{(0)}$ ו־$p_i = u_i^{(i)}$ עבור $i = 1, 2, \ldots, n-1$.

ניתוח

נתונים מתקדמים דרך הכופל המערכי מהפינה השמאלית העליונה עד לפינה הימנית התחתונה. ליצירת הסיביות הפחות משמעותיות $\langle p_{n-1}, p_{n-2}, \ldots, p_0 \rangle$ של המכפלה נחוץ זמן של $\Theta(n)$, ונחוץ זמן של $\Theta(n)$ עד שהפלטים לסוכם מוכנים. אם הסוכם הוא סוכם מגלגל־נשא, נחוץ זמן נוסף של $\Theta(n)$ עד לקבלת הסיביות היותר משמעותיות $\langle p_{2n-1}, p_{2n-2}, \ldots, p_n \rangle$ של המכפלה. אם הסוכם הוא סוכם צופה־נשא, אזי נחוץ לשם כך זמן של $\Theta(\lg n)$ בלבד, אולם הזמן הכולל נותר $\Theta(n)$.

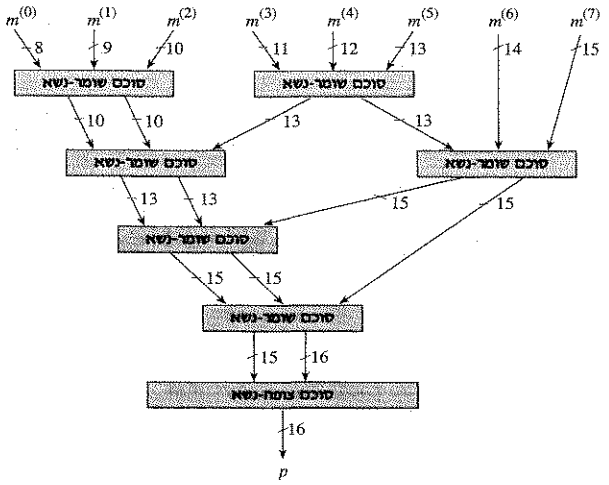
בכופל המערכי ישנם n^2 שערי AND ו־$n^2 - n$ סוכמים מלאים. הסוכם המחשב את הסיביות היותר משמעותיות של הפלט מוסיף עוד $\Theta(n)$ שערים בלבד. לכן, גודלו של הכופל המערכי הוא $\Theta(n^2)$.

29.3.2 כופלי עץ־וואלאס

עץ־וואלאס (Wallace tree) הוא מעגל שעושה רדוקציה של בעיית החיבור של n מספרים בני n סיביות לבעיית החיבור של שני מספרים בני $\Theta(n)$ סיביות. הוא עושה זאת על־ידי שימוש ב־$\lfloor n/3 \rfloor$ סוכמים שומרי־נשא הפועלים במקביל להמרת סכום n המספרים לסכום של $\lceil 2n/3 \rceil$ מספרים. לאחר מכן, הוא בונה באופן רקורסיבי עץ וואלאס על $\lceil 2n/3 \rceil$ המספרים שהתקבלו. בדרך זו, קבוצת המספרים קטנה בהדרגה עד שנותרים רק שני מספרים. על־ידי ביצוע במקביל של

חיבורים שומרי־נשא רבים, עצי וואלאס מאפשרים לכפול שני מספרים בני n סיביות בזמן $\Theta(\lg n)$, באמצעות מעגל בגודל $\Theta(n^2)$.

באיור 29.16 מוצג עץ־וואלאס[2] המחבר 8 מכפלות חלקיות $m^{(0)}, m^{(1)}, \ldots, m^{(7)}$. המכפלה החלקית $m^{(i)}$ מורכבת מ־$n + i$ סיביות. כל שורה מייצגת מספר בשלמותו, לא סיבית יחידה בלבד; ליד כל שורה רשום מספר הסיביות שהשורה מייצגת (ראה תרגיל 29.3-3). הסוכם צופה־הנשא התחתון מחבר מספר בן $(2n - 1)$ סיביות למספר בן $2n$ סיביות, לקבלת המכפלה בת $2n$ הסיביות.



איור 29.16 עץ וואלאס המחבר $n = 8$ מכפלות חלקיות $m^{(0)}, m^{(1)}, \ldots, m^{(7)}$. כל שורה מייצגת מספר שמספר סיביותיו רשום באיור. הפלט השמאלי של כל סוכם שומר־נשא מייצג את סיביות הסכום והפלט הימני מייצג את סיביות הנשא.

ניתוח

הזמן הדרוש לפעולתו של עץ וואלאס בן n קלטים תלוי בעומקם של הסוכמים שומרי־הנשא. בכל רמה של העץ, כל קבוצה של 3 מספרים מצטמצמת ל־2 מספרים, ונותרים לכל היותר שני מספרים (כמו במקרה של $m^{(6)}$ ו־$m^{(7)}$ ברמה העליונה באיור 29.16). לכן, ניתן לבטא את עומקו המקסימלי $D(n)$ של סוכם שומר־נשא בעץ־וואלאס בן n קלטים באמצעות נוסחת הנסיגה:

$$D(n) = \begin{cases} 0 & \text{אם } n \le 2 \\ 1 & \text{אם } n = 3 \\ D(\lceil 2n/3 \rceil) + 1 & \text{אם } n \ge 4 \end{cases}$$

שפתרונה $D(n) = \Theta(\lg n)$ על פי מקרה 2 של משפט האב (משפט 4.1). כל סוכם שומר־נשא מבצע את פעולתו בזמן $\Theta(1)$. את כל n המכפלות החלקיות ניתן ליצור במקביל בזמן $\Theta(1)$.

2. קפי שניתן לראות באיור, עץ וואלאס אינו עץ אמיתי אלא גרף מכוון ללא מעגלים. הוא מכונה "עץ" מסיבות היסטוריות.

(וערכן של $i-1$ הסיביות הפחות משמעותיות של $m^{(i)}$, עבור $i = 1, 2, \ldots, n-1$, נקבע בחומרה ל־0.) הסוכם צופה־הנשא מבצע את פעולתו בזמן $O(\lg n)$. לכן, הכפל של שני מספרים בני n סיביות מתבצע כולו בזמן $\Theta(\lg n)$.

גודלו של כופל עץ־וואלאס עבור שני מספרים בני n סיביות הוא $\Theta(n^2)$, כפי שמראה הניתוח שלהלן. תחילה אנו חוסמים את גודל המעגל של הסוכמים שומרי־הנשא. ניתן לקבל בנקל חסם מלמטה של $\Omega(n^2)$, שכן בעומק 1 ישנם $\lfloor n/3 \rfloor$ סוכמים שומרי־נשא, וכל אחד מהם מורכב לפחות מ־n סוכמים מלאים. כדי לקבל חסם עליון של $O(n^2)$, נשים לב כי מאחר שהמכפלה הסופית היא בת $2n$ סיביות, כל סוכם שומר־נשא בעץ־וואלאס מכיל לכל היותר $2n$ סוכמים מלאים. עלינו להראות שמספר הכולל של הסוכמים שומרי־הנשא הוא $O(n)$. יהי $C(n)$ המספר הכולל של סוכמים שומרי־נשא בעץ־וואלאס בן n מספרי קלט. אנו מקבלים את נוסחת הנסיגה:

$$C(n) \le \begin{cases} 1 & \text{אם } n = 3 \\ C(\lceil 2n/3 \rceil) + \lfloor n/3 \rfloor & \text{אם } n \ge 4 \end{cases}$$

שפתרונה $C(n) = \Theta(n)$ על פי מקרה 3 של משפט האב. קיבלנו אפוא חסם הדוק אסימפטוטית של $\Theta(n^2)$ על גודלו של מעגל הסוכמים שומרי־הנשא בכופל עץ־וואלאס. גודלו של המעגל המחשב את n המכפלות החלקיות הוא $\Theta(n^2)$, וגודלו של הסוכם צופה־הנשא הדרוש בסוף הוא $\Theta(n)$. גודל הכופל כולו הוא אפוא $\Theta(n^2)$.

אף כי כופל המבוסס על עץ־וואלאס מהיר יותר אסימפטוטית מאשר הכופל המערכי ובעל אותו גודל אסימפטוטי, מבנהו של כופל עץ־וואלאס, כאשר מממשים אותו, אינו סדיר כמו זה של הכופל המערכי, ואיננו "צפוף" כמותו (במובן של המקום המבוזבז בין רכיבי המעגל). ביישומים מעשיים, משתמשים לעתים קרובות במעין פשרה בין שני המבנים. הרעיון הוא להשתמש בשני מערכים במקביל, שכל אחד מהם מסכם מחצית מן המכפלות החלקיות. זמן ההשהיה הוא מחצית בלבד מזה שמתקבל במערך יחיד המסכם את כל n המכפלות החלקיות. שני חיבורים שומרי־נשא נוספים מצמצמים את ארבעת המספרים שמפיקים המערכים לשני מספרים, ואז סוכם צופה־נשא מסכם את שני המספרים כדי לקבל את המכפלה. זמן ההשהיה הכולל עולה במעט על מחצית זמן ההשהיה בכופל מערכי מלא, ועוד איבר נוסף מהצורה $O(\lg n)$.

תרגילים

29.3-1

הוכח שבכופל מערכי מתקיים $v_j^{(i)} = 0$ עבור $i = 1, 2, \ldots, n-1$ ו־$j = 0, 1, \ldots, i, i+n, i+n+1, \ldots, 2n-1$.

29.3-2

הראה שבכופל המערכי שבאיור 29.14, כל הסוכמים המלאים בשורה העליונה, פרט לאחד, מיותרים למעשה. יהיה עליך לחבר מחדש חלק מן התילים.

29.3-3

נניח שסוכם שומר־נשא מקבל כקלט את x, y ו־z ומפיק את הפלטים s ו־c, שהם בני n_x, n_y, n_z, n_s ו־n_c סיביות, בהתאמה. עוד נניח, בלי הגבלת הכלליות, כי $n_z \le n_y \le n_x$. הראה כי $n_s = n_z$, וכן:

$$n_c = \begin{cases} n_z & \text{אם } n_y < n_z \\ n_z + 1 & \text{אם } n_y = n_z \end{cases}$$

29.3-4

הראה שגם אם נגביל את פרישת היציאה של השערים ל־$O(1)$, עדיין נוכל לבצע כפל בזמן $O(\lg n)$ באמצעות מעגל בגודל $O(n^2)$.

29.3-5

תאר מעגל יעיל לחישוב המנה כאשר מחלקים מספר בינרי x ב־3. (**רמז**: שים לב לכך שמספרים ביניריים מקיימים $\cdots \times 1.0001 \times 1.01 \times .01 = .010101\ldots$.)

29.3-6

מעגל הזזה מעגלית (cyclic shifter), או **מעגל הזזה גלילית** (barrel shifter), הוא מעגל המקבל שני קלטים $x = \langle x_{n-1}, x_{n-2}, \ldots, x_0 \rangle$ ו־$s = \langle s_{m-1}, s_{m-2}, \ldots, s_0 \rangle$, כאשר $m = \lceil \lg n \rceil$. פלט המעגל $y = \langle y_{n-1}, y_{n-2}, \ldots, y_0 \rangle$ מוגדר על־ידי $y_i = x_{i+s \bmod n}$ עבור $i = 0, 1, \ldots, n-1$. כלומר, המעגל מזיז בצורה מעגלית את הסיביות של x במספר המקומות שמציין s. תאר מעגל הזזה מעגלית יעיל. במונחים של כפל מודולרי, איזו פונקציה מממש מעגל הזזה מעגלית?

29.4 מעגלים מבוקרי־שעון

במהלך ביצועו של חישוב משתמשים בכל אחד מן הרכיבים של מעגל צירופי רק פעם אחת. אם נשלב במעגל רכיבי זיכרון מבוקרי־שעון, נוכל לשוב ולהשתמש ברכיבים הצירופיים. מכיוון שהם יכולים להשתמש בחומרה יותר מפעם אחת, מעגלים מבוקרי־שעון יכולים להיות לעתים קרובות קטנים בהרבה ממעגלים צירופיים המבצעים אותה פונקציה.

בסעיף זה נחקור מעגלים מבוקרי־שעון עבור פעולות חיבור וכפל. נתחיל במעגל מבוקר־שעון בגודל $\Theta(1)$, הנקרא סוכם סיביות סדרתי, אשר יכול לחבר שני מספרים בני n סיביות בזמן $\Theta(n)$. לאחר מכן נחקור כופלי מערך־לינארי. נציג כופל מערך־לינארי בגודל $\Theta(n)$ שיכול לכפול שני מספרים בני n סיביות בזמן $\Theta(n)$.

29.4.1 חיבור סיביות סדרתי

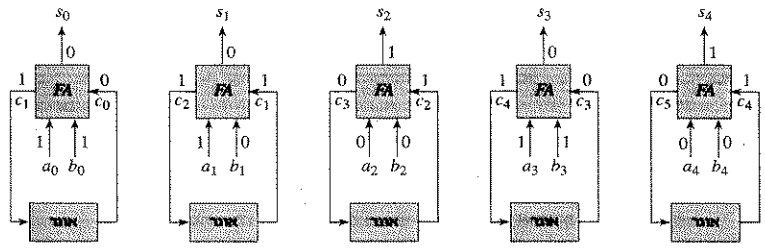
כדי להציג את המושג מעגל מבוקר־שעון, נחזור לבעיית החיבור של שני מספרים בני n סיביות. איור 29.17 מדגים כיצד ניתן להשתמש בסוכם מלא יחיד לקבלת הסכום $s = \langle s_n, s_{n-1}, \ldots, s_0 \rangle$ בן $(n+1)$ הסיביות של שני המספרים בני n הסיביות $a = \langle a_{n-1}, a_{n-2}, \ldots, a_0 \rangle$ ו־$b = \langle b_{n-1}, b_{n-2}, \ldots, b_0 \rangle$. העולם החיצון מגיש לנו את סיביות הקלט זוג אחר זוג: תחילה a_0 ו־b_0, לאחר מכן a_1 ו־b_1, וכן הלאה. למרות שאנו רוצים כי הנשא היוצא מחיבור הסיביות במיקום מסוים יהיה הנשא הנכנס עבור מיקום הסיביות הבא, איננו יכולים פשוט להזין את הפלט c מן הסוכם המלא ישירות כקלט. יש כאן עניין של תזמון: הנשא

הנכנס c_i לסוכם המלא חייב להתאים לקלטים הנכונים a_i ו־b_i. אם סיביות קלט אלה לא יגיעו בדיוק באותו רגע שבו מוזן הנשא בחזרה לסוכם, הפלט עלול להיות שגוי.

כפי שניתן לראות באיור 29.17, הפתרון הוא להשתמש **במעגל מבוקר־שעון** (clocked circuit), או **מעגל סדרתי** (sequential circuit), המורכב מרכיבים צירופיים ו**מאוגר** (register), או רכיב זיכרון מבוקר־שעון) אחד או יותר. הקלטים לרכיבים הצירופיים מגיעים מן העולם החיצון או מן הפלט של האוגרים. הפלטים שלהם נפלטים אל העולם החיצון או מוזנים כקלט לאוגרים. כמו במעגלים צירופיים, איננו מרשים שהרכיבים הצירופיים במעגל מבוקר־שעון יכילו מסלולים מעגליים.

כל אוגר במעגל מבוקר־שעון מבוקר על ידי אות מחזורי, או **שעון** (clock). בכל פעם שהשעון שולח דופק או **פועם**, ערך הקלט נטען באוגר ומאוחסן שם. פרק הזמן בין שתי פעימות עוקבות של השעון הוא **זמן המחזור של השעון** (clock period). במעגל **מבוקר־שעון באופן גלובלי** (gióbally clocked), כל האוגרים פועלים על פי אותו שעון.

הבה נבחן ביתר פירוט את פעולתו של אוגר. אנו מתייחסים לכל פעימה של השעון כאל דופק רגעי. בעת פעימה נתונה, האוגר קורא את ערך הקלט המוצג לו *באותו רגע* ומאחסן אותו. ערך מאוחסן זה מופיע אז כפלט של האוגר, וניתן להשתמש בו לחישוב ערכים המועברים לאוגרים אחרים בעת פעימת השעון הבאה. במילים אחרות, הערך המופיע כקלט האוגר במהלך זמן מחזור אחד של השעון מופיע כפלט האוגר במהלך זמן המחזור הבא של השעון.



איור 29.17 פעולתו של סוכם סיביות סדרתי. במהלך מחזור השעון ה־i, עבור $i = 0, 1, \ldots, n$, הסוכם המלא FA מקבל את סיביות הקלט a_i ו־b_i מן העולם החיצון ואת סיבית הנשא c_i מן האוגר. הסוכם המלא מוציא אז לפלט את סיבית הסכום s_i המופנית החוצה, ואת סיבית הנשא c_{i+1} המאוחסנת בחזרה באוגר כדי שניתן יהיה להשתמש בה במהלך מחזור השעון הבא. האוגר מאותחל לערך $c_0 = 0$. **(i)-(v)** מצב המעגל בכל אחד מחמשת מחזורי השעון במהלך החיבור של $a = \langle 1011 \rangle$ ו־$b = \langle 1001 \rangle$ לקבלת $s = \langle 10100 \rangle$.

הבה נבחן עתה את המעגל שבאיור 29.17, שאנו קוראים לו **סוכם סיביות סדרתי** (bit-serial adder). כדי שהפלטים של הסוכם המלא יהיו נכונים, אנו דורשים שאורכו של זמן המחזור של השעון יהיה לפחות זמן ההשהיה של הסוכם המלא, כך שהערכים ברכיבים הצירופיים יוכלו להתייצב בין שתי פעימות שעון. במהלך מחזור השעון 0, המוצג באיור 29.17(i), העולם החיצון מזין את סיביות הקלט a_0 ו־b_0 לשניים מן הקלטים של הסוכם המלא. אנו מניחים שהאוגר מאותחל לערך 0; לכן, סיבית הנשא הנכנס ההתחלתית, שהיא פלט של האוגר, היא $c_0 = 0$. מאוחר יותר במהלך זמן מחזור זה של השעון, סיבית הסכום s_0 וסיבית הנשא היוצא c_1 יופיעו מתוך הסוכם המלא. סיבית הסכום תועבר אל העולם החיצון, וההנחה היא כי שם תישמר כחלק

מן הסכום s כולו. התיל מן הנשא היוצא של הסוכם המלא מוזן אל האוגר, כך שעם פעימת השעון הבאה, c_1 נקרא אל תוך האוגר. בתחילת מחזור השעון 1, האוגר מכיל אפוא את c_1. במהלך מחזור השעון 1, המוצג באיור 29.17(ii), העולם החיצון מזין את הסיביות a_1 ו־b_1 לתוך הסוכם המלא, אשר קורא גם את c_1 מן האוגר ומפיק את הפלטים s_1 ו־c_2. סיבית הסכום s_1 יוצאת אל העולם החיצון ו־c_2 מועברת אל האוגר. פעולה מחזורית זו נמשכת עד למחזור השעון n, המוצג באיור 29.17(v), שבו האוגר מכיל את c_n. אז מזין העולם החיצון $a_n = b_n = 0$, כך שאנו מקבלים $s_n = c_n$.

ניתוח

כדי לחשב את הזמן הכולל t שאורכת פעולתו של מעגל מבוקר־שעון באופן גלובלי, עלינו לדעת את המספר p של מחזורי השעון ואת משך הזמן d של כל מחזור: $t = pd$. מחזור השעון d צריך להיות ארוך דיו כדי שכל הערכים במעגל הצירופי יספיקו להתייצב בין פעימות עוקבות. אף כי עבור קלטים מסוימים הערכים עשויים להתייצב מוקדם יותר, הרי שאם ברצוננו להבטיח שהמעגל יפעל כהלכה עבור כל הקלטים, גודלו של d חייב להיות לפחות ביחס ישר לעומקו של המעגל הצירופי.

הבה נראה כמה זמן אורך חיבורם של שני מספרים בני n סיביות באמצעות סוכם סיביות סדרתי. זמן המחזור של השעון הוא $\Theta(1)$, שכן עומקו של הסוכם המלא הוא $\Theta(1)$. מכיוון שלהפקת כל הפלטים נדרשות $(n+1)$ פעימות שעון, הרי שהזמן הכולל לביצוע חיבור סיביות סדרתי הוא $(n+1)\Theta(1) = \Theta(n)$.

גודלו של סוכם סיביות סדרתי (מספר הרכיבים הצירופיים ועוד מספר האוגרים) הוא $\Theta(1)$.

חיבור מגלגל־נשא לעומת חיבור סיביות סדרתי

נשים לב כי סוכם מגלגל־נשא דומה לסוכם סיביות סדרתי שהוכפל כמה פעמים, כאשר האוגרים מוחלפים בחיבורים ישירים בין הרכיבים הצירופיים. כלומר, סוכם מגלגל־נשא מתאים ל"פרישה מרחבית" של החישוב שמבצע סוכם הסיביות הסדרתי. הסוכם המלא ה־i בסוכם מגלגל־הנשא מממש את מחזור השעון ה־i של סוכם הסיביות הסדרתי.

כלל, ניתן להחליף כל מעגל מבוקר־שעון במעגל צירופי שקול בעל אותו זמן השהיה אסימפטוטי — אם אנו יודעים מראש כמה מחזורי שעון פועל המעגל מבוקר־השעון. כמובן, כרוך בכך עניין של "תן־וקח". המעגל מבוקר־השעון מכיל פחות רכיבים (פי $\Theta(n)$ פחות בסוכם הסיביות הסדרתי לעומת הסוכם מגלגל־הנשא), אבל יתרונו של המעגל הצירופי בכך שיש בו פחות מעגלי בקרה — אין צורך בשעון או במעגל חיצוני מסונכרן להזנת סיביות הקלט ולאחסון סיביות הסכום. יתר על כן, אף כי שני המעגלים הם בעלי אותו זמן השהיה אסימפטוטי, הרי שהלכה למעשה, המעגל הצירופי פועל בדרך כלל קצת יותר מהר. המהירות הגבוהה יותר מתאפשרת משום שהמעגל הצירופי אינו צריך לחכות להתייצבותם של ערכים במהלך כל מחזור של השעון. אם כל הקלטים מתייצבים מיד, ערכים פשוט מתגלגלים דרך המעגל במהירות המקסימלית האפשרית, בלי לחכות לשעון.

29.4.2 כופלים מבוססי מערך לינארי

הכופלים הצירופיים שהוצגו בסעיף 29.3 דורשים גודל של $\Theta(n^2)$ כדי לכפול שני מספרים בני n סיביות. עתה נציג שני כופלים הבנויים ממערכים לינאריים, ולא דו־ממדיים, של רכיבים. המהיר מבין שני הכופלים האלה רץ בזמן $\Theta(n)$, כמו הכופל המערכי.

הכופלים המבוססים על מערכים לינאריים מממשים את **אלגוריתם האיכר הרוסי** (Russian peasant's algorithm) (הנקרא כך מכיוון שאנשי המערב שביקרו ברוסיה במאה התשע־עשרה מצאו שהיה שם בשימוש נרחב). האלגוריתם מודגם באיור 29.18(i). בהינתן שני מספרי קלט a ו־b, אנו עורכים שני טורי מספרים שבראשם a ו־b. בכל שורה, הערך המופיע בטור של a הוא הערך השלם של מחצית הערך בשורה הקודמת בטור של a; הערך בטור של b הוא כפליים הערך בשורה הקודמת בטור של b. השורה האחרונה היא השורה שבה מופיע בטור של a הערך 1. אנו מתבוננים בכל השורות שבהן מופיעים בטור של a ערכים אי־זוגיים, ומסכמים את הערכים המופיעים בטור של b באותן שורות. סכום זה הוא המכפלה $a \cdot b$.

אף כי אלגוריתם האיכר הרוסי נראה יוצא דופן במבט ראשון, איור 29.18(ii) מראה שאינו אלא מימוש שיטת הכפל של בית הספר היסודי במערכת מספרים בינריים; אלא שהמספרים כתובים בכתיב עשרוני ולא בינרי. שורות שבהן הערך בטור של a הוא אי־זוגי תורמות למכפלה את b כפול החזקה המתאימה של 2.

a	b
19	**29**
9	**58**
4	116
2	232
1	**464**
	551

(i)

a	b
10011	**11101**
1001	**111010**
100	1110100
10	11101000
1	**111010000**
	1000100111

(ii)

איור 29.18 כפל 19 ב־29 באמצעות אלגוריתם האיכר הרוסי. בטור של a, הערך בכל שורה הוא מחצית הערך בשורה הקודמת בטור זה, כשמתעלמים משברים. בטור של b, הערכים מוכפלים משורה לשורה. אנו מסכמים את אותם ערכים בטור של b המופיעים בכל השורות שבהן הערכים בטור של a הם אי־זוגיים (שורות אלה מוצללות באיור). סכום זה הוא המכפלה המבוקשת. **(i)** המספרים בכתיב עשרוני. **(ii)** אותם מספרים בכתיב בינרי.

מימוש איטי של כופל מבוסס מערך־לינארי

באיור 29.19(i) מוצגת דרך אחת למימוש אלגוריתם האיכר הרוסי עבור שני מספרים בני n סיביות. אנו משתמשים במעגל מבוקר־שעון המורכב ממערך לינארי של $2n$ תאים. כל תא מכיל שלושה אוגרים: אוגר אחד מאחסן סיבית מאחד הערכים בטור של a, השני מאחסן סיבית מאחד הערכים בטור של b, והשלישי מאחסן סיבית של המכפלה p. אנו מסמנים באינדקסים עליונים ערכי תאים לפני כל צעד של האלגוריתם. לדוגמה, ערכה של הסיבית a_i לפני הצעד ה־j הוא $a_i^{(j)}$, ואנו מגדירים $a^{(j)} = \langle a_{2n-1}^{(j)}, a_{2n-2}^{(j)}, \ldots, a_0^{(j)} \rangle$.

האלגוריתם מבצע סדרה של n צעדים, הממוספרים על־ידי $0, 1, \ldots, n-1$, כאשר כל צעד נמשך מחזור אחד של השעון. האלגוריתם משמר את האינווריאנטה שעל פיה לפני הצעד ה־j מתקיים:

$$a^{(j)} \cdot b^{(j)} + p^{(j)} = a \cdot b \qquad (29.6)$$

(ראה תרגיל 29.4-2). במצב ההתחלתי, $a^{(0)} = a$, $b^{(0)} = b$ וכן $p^{(0)} = 0$. בצעד ה־j מתבצעים החישובים הבאים:

1. אם $a^{(j)}$ אי־זוגי (כלומר, $a_0^{(j)} = 1$), אזי מוסיפים את b ל־p: $p^{(j+1)} \leftarrow b^{(j)} + p^{(j)}$. (החיבור מתבצע על־ידי סוכם מגלגל־נשא המוצב לאורכו של המערך; סיביות נשא מתגלגלות מימין לשמאל.) אם $a^{(j)}$ זוגי, אזי מעבירים את p הלאה לצעד הבא: $p^{(j+1)} \leftarrow p^{(j)}$.
2. מזיזים את a מקום אחד ימינה:

$$a_i^{(j+1)} \leftarrow \begin{cases} a_{i+1}^{(j)} & 0 \le i \le 2n-2 \quad \text{אם} \\ 0 & i = 2n-1 \quad \text{אם} \end{cases}$$

3. מזיזים את b מקום אחד שמאלה:

$$b_i^{(j+1)} \leftarrow \begin{cases} b_{i-1}^{(j)} & 1 \le i \le 2n-1 \quad \text{אם} \\ 0 & i = 0 \quad \text{אם} \end{cases}$$

לאחר ביצוע n צעדים, כל הסיביות של a כבר הוזזו החוצה, ולפיכך $a^{(n)} = 0$. מאינווריאנטה (29.6) נובע כי $p(n) = a \cdot b$.

עתה ננתח את האלגוריתם. האלגוריתם מבצע n צעדים, בהנחה שמידע הבקרה משודר לכל התאים בו־זמנית. כל צעד מתבצע במקרה הגרוע בזמן $\Theta(n)$, משום שעומקו של הסוכם מגלגל־הנשא הוא $\Theta(n)$, ולכן משך מחזור השעון חייב להיות לפחות $\Theta(n)$. כל הזזה נמשכת זמן $\Theta(1)$ בלבד. זמן הריצה הכולל של האלגוריתם הוא אפוא $\Theta(n^2)$. מאחר שכל תא הוא בעל גודל קבוע, גודלו של המערך הלינארי כולו הוא $\Theta(n)$.

מימוש מהיר של כופל מבוסס מערך־לינארי

על ידי שימוש בחיבור שומר־נשא במקום חיבור מגלגל־נשא, ניתן לקצר את הזמן עבור כל צעד ל־$\Theta(1)$, ולשפר בכך את הזמן הכולל ל־$\Theta(n)$. כפי שמראה איור 29.19(ii), גם הפעם כל תא מכיל סיבית אחת מאחד הערכים בטור של a וסיבית אחת מאחד הערכים בטור של b. כל תא מכיל גם שתי סיביות נוספות, u ו־v, שהם הפלטים מהחיבור שומר־הנשא. תוך שימוש בייצוג שומר־נשא לצבירת המכפלה, אנו משמרים את האינווריאנטה שעל פיה לפני הצעד ה־j מתקיים:

$$a^{(j)} \cdot b^{(j)} + u^{(j)} + v^{(j)} = a \cdot b \qquad (29.7)$$

(ראה שוב תרגיל 29.4-2). בכל צעד מתבצעת הזזה על a ועל b באותה דרך כמו במימוש האיטי, כך שניתן לצרף את המשוואות (29.6) ו־(29.7) ולקבל $u^{(j)} + v^{(j)} = p^{(j)}$. לכן, הסיביות של u ו־v מכילות אותו מידע כמו הסיביות של p במימוש האיטי.

בצעד ה־j של המימוש המהיר מבוצע חיבור שומר־נשא של u ו־v, כאשר האופרנדים תלויים בשאלה אם a זוגי או אי־זוגי. אם $a_0^{(j)} = 1$, אנו מחשבים:

$$u_i^{(j+1)} \leftarrow \text{parity}(b_i^{(j)}, u_i^{(j)}, v_i^{(j)}) \qquad \text{עבור } i = 0, 1, \ldots, 2n-1$$

וכן:

$$v_i^{(j+1)} \leftarrow \begin{cases} \text{majority}(b_{i-1}^{(j)}, u_{i-1}^{(j)}, v_{i-1}^{(j)}) & 1 \le i \le 2n-1 \quad \text{אם} \\ 0 & i = 0 \quad \text{אם} \end{cases}$$

אחרת, $a_0^{(j)} = 0$, ואנו מחשבים:

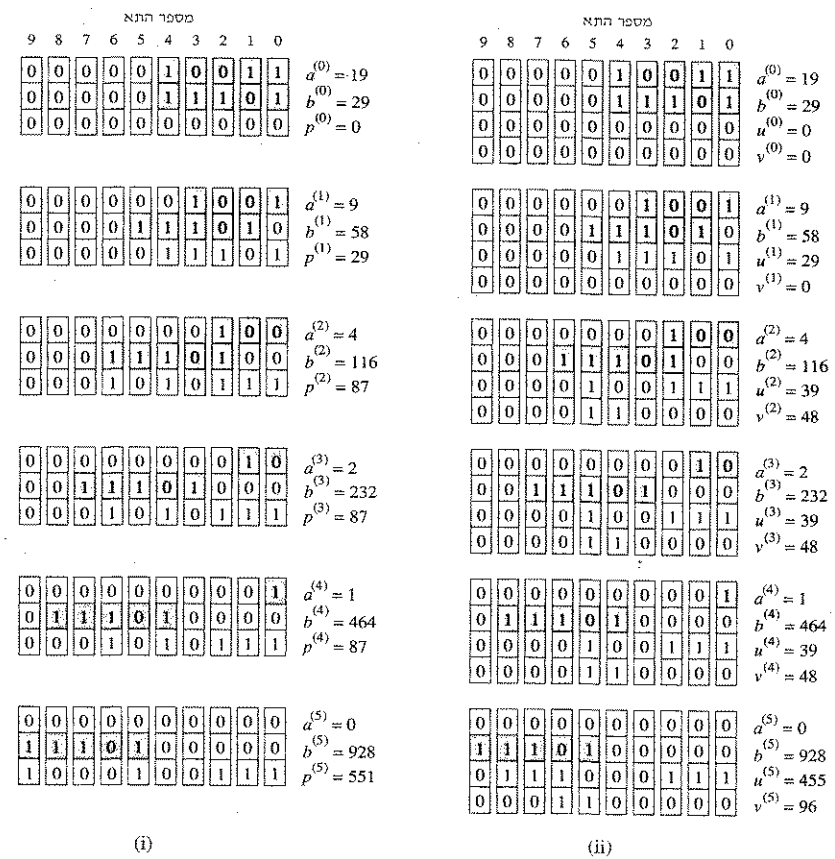
עבור $i = 0, 1, \ldots, 2n-1$

$$u_i^{(j+1)} \leftarrow \text{parity}(0, u_i^{(j)}, v_i^{(j)})$$

וכן:

$$v_i^{(j+1)} \leftarrow \begin{cases} \text{majority}(0, u_{i-1}^{(j)}, v_{i-1}^{(j)}) & \text{אם } 1 \le i \le 2n-1 \\ 0 & \text{אם } i = 0 \end{cases}$$

לאחר עדכון u ו־v, הצעד ה־j מזיז את a ימינה ואת b שמאלה באותו אופן כמו במימוש האיטי.



איור 29.19 שני מימושים מבוססי מערך־לינארי של אלגוריתם האיכר הרוסי תמציגים כפל של $\langle 10011 \rangle = a = 19$ ב־$\langle 11101 \rangle = b = 29$, עם $n = 5$. באיור מוצג המצב בתחילת כל צעד j, כאשר הסיביות המשמעותיות הנותרות של $a^{(j)}$ ושל $b^{(j)}$ מוצללות. **(i)** מימוש איטי שרץ בזמן $\Theta(n^2)$. מכיוון ש־$a^{(5)} = 0$, אנו מקבלים $p^{(5)} = a \cdot b$. מתבצעים n צעדים, וכל צעד משתמש בחיבור מגלגל־נשא. זמן המחזור של השעון נמצא אפוא ביחס ישר לאורך המערך, כלומר, ל־$\Theta(n)$, ומכך נובע כי זמן הריצה הכולל הוא $\Theta(n^2)$. **(ii)** מימוש מהיר שרץ בזמן $\Theta(n)$ משום שבכל צעד משתמשים בחיבור שומר־נשא (ולא בחיבור מגלגל נשא). המתבצע בזמן $\Theta(1)$ בלבד. מספר הצעדים הכולל הוא $2n - 1 = 9$; לאחר הצעד האחרון המוצג באיור, חיבור שומר־נשא חוזר של u ו־v מניב $u^{(9)} = a \cdot b$.

המימוש המהיר מבצע בסך הכל $2n - 1$ צעדים. עבור $j \ge n$, אנו מקבלים $a^{(j)} = 0$, ולכן מאינווריאנטה (29.7) נובע כי $u^{(j)} + v^{(j)} = a \cdot b$. מרגע ש־$a^{(j)} = 0$, כל הצעדים הבאים משמשים רק לביצוע חיבור שומר־נשא של u ו־v. בתרגיל 29.4-3 תתבקש להראות כי $v^{(2n-1)} = 0$, כך ש־$u^{(2n-1)} = a \cdot b$.

זמן הריצה הכולל במקרה הגרוע הוא $\Theta(n)$, שכן כל אחד מ־$2n-1$ הצעדים מתבצע בזמן $\Theta(1)$. מכיוון שגם כאן גודלו של כל תא קבוע, הגודל הכולל נותר $\Theta(n)$.

תרגילים

29.4-1

יהיו $a = \langle 101101 \rangle$, $b = \langle 011110 \rangle$, ו־$n = 6$. הראה בכתיב עשרוני ובכתיב בינרי כיצד פועל אלגוריתם האיכר הרוסי על הקלטים a ו־b.

29.4-2

הוכח את האינווריאנטות (29.6) ו־(29.7) עבור כופלים מבוססי מערך לינארי.

29.4-3

הוכח שבמימוש המהיר של כופל מבוסס מערך לינארי מתקיים $v^{(2n-1)} = 0$.

29.4-4

תאר כיצד מייצג הכופל המערכי מסעיף 29.3.1 "פרישה" של החישוב שמבצע כופל המערך־הלינארי המהיר.

29.4-5

נתון זרם נתונים $\langle x_1, x_2, \ldots \rangle$ המגיע למעגל מבוקר־שעון בקצב של ערך אחד בכל פעימת שעון. עבור ערך קבוע n, המעגל צריך לחשב את הערך:

$$y_t = \max_{t-n+1 \le i \le t} x_i$$

עבור $t = n, n+1, \ldots$. כלומר, y_t הוא המקסימום מבין n הערכים האחרונים שהגיעו אל המעגל. תכנן מעגל בגודל $O(n)$ אשר בכל פעימת שעון קולט את הערך x_t ומחשב את ערך הפלט y_t בזמן $O(1)$. ניתן להשתמש באוגרים וברכיבים צירופיים המחשבים את המקסימום מבין שני קלטים.

* 29.4-6

חזור על תרגיל 29.4-5 תוך שימוש ב־$O(\lg n)$ רכיבי "מקסימום" בלבד.

בעיות

29-1 מעגלי חילוק

ניתן לבנות מעגל חילוק ממעגלי חיסור וכפל באמצעות שיטה המכונה **איטרציית ניוטון** (Newton iteration). אנו נתמקד בביעה הקרובה של חישוב הופכי, שכן בהינתן מעגל לחישוב ההופכי, ניתן לקבל ממנו מעגל חילוק על־ידי הוספת פעולת כפל אחת.

הרעיון הוא לחשב סדרה $y_0, y_1, y_2, \ldots$ של קירובים להופכי של מספר x באמצעות הנוסחה הבאה:

$$y_{i+1} \leftarrow 2y_i - xy_i^2$$

נניח ש־x נתון כשבר בינרי בן n סיביות בתחום $1 \le x \le 1/2$. מכיוון שההופכי יכול להיות שבר חוזר אינסופי, אנו נתמקד בחישובו של קירוב בן n סיביות, מדויק עד לסיבית הכי פחות משמעותית שלו.

א. נניח כי $|y_i - 1/x| \le \epsilon$ עבור איזשהו קבוע $\epsilon > 0$. הוכח כי $|y_{i+1} - 1/x| \le \epsilon^2$.

ב. מצא קירוב התחלתי y_0 כך ש־y_k מקיים $|y_k - 1/x| \le 2^{-2^k}$ עבור כל $k \ge 0$. כמה גדול צריך k להיות כדי שהקירוב y_k יהיה מדויק עד הסיבית הכי פחות משמעותית שלו?

ג. תאר מעגל צירופי אשר, בהינתן קלט x בן n סיביות, מחשב קירוב בן n סיביות של $1/x$ בזמן $O(\lg^2 n)$. מהו גודל המעגל שלך? **(רמז:** בעזרת תחכום מסוים ניתן "לגבור" על החסם $\Theta(n^2 \lg n)$ על הגודל.)

29-2 נוסחאות בוליאניות עבור פונקציות סימטריות

פונקציה בת n קלטים $f(x_1, x_2, \ldots, x_n)$ נקראת **סימטרית** (symmetric) אם:

$$f(x_1, x_2, \ldots, x_n) = f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$$

עבור כל תמורה π של $\{1, 2, \ldots, n\}$. בבעיה זו נראה שקיימת נוסחה בוליאנית המייצגת את f, ואשר גודלה פולינומיאלי ב־n. (למטרותינו, נוסחה בוליאנית היא מחרוזת המורכבת מן המשתנים $x_1, x_2, \ldots, x_n$, מסוגריים, ומן האופרטורים הבוליאניים $\vee$, $\wedge$ ו־$\neg$.) גישתנו תהיה להמיר מעגל בוליאני בעומק לוגריתמי בנוסחה בוליאנית שקולה בגודל פולינומיאלי. נניח שכל המעגלים מורכבים משערי AND בעלי 2 קלטים, משערי OR בעלי 2 קלטים ומשערי NOT.

א. נתחיל בכך שנתבונן בפונקציה סימטרית פשוטה. **פונקציית הרוב** (majority function) המוכללת על n קלטים בוליאניים מוגדרת על־ידי:

$$\text{majority}_n(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{אם } x_1 + x_2 + \cdots + x_n > n/2 \\ 0 & \text{אחרת} \end{cases}$$

תאר מעגל צירופי שעומקו $O(\lg n)$ עבור majority_n. **(רמז:** בנה עץ של סוכמים.)

ב. נניח כי f היא פונקציה בוליאנית כלשהי של n המשתנים הבוליאניים $x_1, x_2, \ldots, x_n$. עוד נניח כי קיים מעגל C בעל עומק d המחשב את f. הראה כיצד לבנות מ־C נוסחה בוליאנית באורך $O(2^d)$ עבור f. הסק כי קיימת נוסחה בגודל פולינומיאלי עבור majority_n.

ג. הראה שניתן לייצג כל פונקציה בוליאנית סימטרית $f(x_1, x_2, \ldots, x_n)$ כפונקציה של $\sum_{i=1}^{n} x_i$.

ד. הראה שכל פונקציה על n קלטים בוליאניים ניתנת לחישוב באמצעות מעגל צירופי בעל עומק $O(\lg n)$.

ה. הראה שכל פונקציה בוליאנית סימטרית על n משתנים בוליאניים ניתנת לייצוג על־ידי נוסחה בוליאנית שאורכה פולינומיאלי ב־n.

הערות ביבליוגרפיות לפרק 29

מרבית הספרים העוסקים בפעולות אריתמטיות במחשבים מתמקדים יותר במימושים מעשיים של מעגלים מאשר בתיאוריה אלגוריתמית. Savage [173] הוא אחד הבודדים החוקרים היבטים אלגוריתמיים של הנושא. מבין הספרים המתמקדים יותר בחומרה של אריתמטיקת מחשבים, Cavanagh [39] ו־Hwang [108] קריאים במיוחד. בין הספרים הטובים על תכנון מעגלים צירופיים ומעגלים לוגיים סדרתיים נמנים Hill & Peterson [96] וכן, עם פזילה לעבר תורת השפות הפורמליות, Kohavi [126].

Aiken & Hopper [7] סוקרים את ההיסטוריה המוקדמת של אלגוריתמים אריתמטיים. חיבור מגלגל־נשא מופלג בגילו לפחות כמו החשבונייה, הידועה כבר למעלה מ־5000 שנה. את המחשבון המכני הראשון שביצע חיבור מגלגל־נשא בנה פסקאל (B. Pascal) בשנת 1642. הרעיון לבניית מכונת חישוב שתותאם לביצוע כפל באמצעות חיבורים חוזרים הועלה על־ידי מורלאנד (S. Morland) ב־1666, ובאופן בלתי תלוי, על־ידי לייבניץ (G. W. Leibnitz) ב־1671. אלגוריתם האיכר הרוסי עבור כפל הוא ככל הנראה עתיק הרבה יותר מאשר השימוש בו ברוסיה במאה התשע־עשרה. על פי Knuth [122], מתמטיקאים מצרים השתמשו בו כבר בסביבות שנת 1800 לפנה"ס.

מצבי הנשא — איפוס, יצירה וקידום — נוצלו במחשבון ממסר שנבנה בהרווארד באמצע שנות ה־40' [180]. אחד המימושים הראשונים של חיבור צופה־נשא תואר על־ידי Weinberger & Smith [199], אולם השיטה שלהם לצפיית נשא דורשת שערים גדולים. Ofman [152] הוכיח שניתן לחבר מספרים בני n סיביות בזמן $O(\lg n)$ באמצעות חיבור צופה־נשא עם שערים בגודל קבוע.

הרעיון להשתמש בחיבור שומר־נשא כדי להאיץ פעולות כפל הועלה על־ידי Estrin, Gilchrist & Pomerene [64]. Atrubin [13] מתאר כופל מבוסס מערך לינארי באורך אינסופי שבאמצעותו ניתן לכפול מספרים בינריים באורך כלשהו. הכופל מפיק את הסיבית ה־n של המכפלה מיד עם קבלת הסיביות ה־n של הקלטים. כופל עץ־וואלאס מיוחס ל־Waliace [197], אולם הרעיון התגלה באופן בלתי תלוי גם על־ידי Ofman [152].

אלגוריתמי חילוק הוצגו לראשונה על־ידי ניוטון (I. Newton), אשר בשנת 1665 בערך המציא את השיטה הידוע כאיטרציית ניוטון. בבעיה 29-1 משתמשת באיטרציית ניוטון לבניית מעגל חילוק בעל עומק $\Theta(\lg^2 n)$. שיטה זו שופרה על־ידי Beame, Cook & Hoover [19] אשר הראו כי למעשה ניתן לבצע חילוק של מספרים בני n סיביות באמצעות מעגל שעומקו $\Theta(\lg n)$.

30 אלגוריתמים עבור מחשבים מקביליים

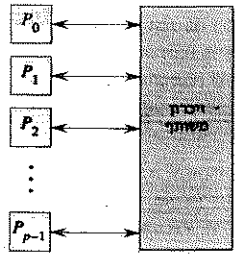
ככל שמחשבים מקביליים הפכו נפוצים יותר, גבר העניין **באלגוריתמים מקביליים** (parallel algorithms). אלגוריתמים המבוצעים בעת ובעונה אחת יותר מאשר פעולה אחת. חקר האלגוריתמים המקביליים התפתח היום לכדי תחום מחקר העומד בזכות עצמו. ואמנם, אלגוריתמים מקביליים פותחו עבור רבות מן הבעיות שפתרנו בספר זה באמצעות אלגוריתמים סדרתיים רגילים. בפרק זה נתאר כמה אלגוריתמים מקביליים פשוטים המדגימים בעיות ושיטות בסיסיות בתחום.

כדי ללמוד אלגוריתמים מקביליים, עלינו לבחור מודל מתאים של חישוב מקבילי. מכונת הגישה האקראית (RAM) — המודל שהשתמשנו בו ברובו של ספר זה — היא כמובן מודל סדרתי ולא מקבילי. המודלים המקביליים שנלמדו — רשתות מיון (פרק 28) ומעגלים (פרק 29) — מוגבלים מכדי לחקור, למשל, אלגוריתמים על מבני נתונים.

האלגוריתמים המקביליים המופיעים בפרק זה מוצגים במונחי אחד המודלים התיאורטיים המקובלים: **מכונת גישה־אקראית מקבילית** (parallel random-access machine, או בקיצור PRAM, המבוטא "פִּי־רַאם"). במודל זה ניתן לתאר בקלות אלגוריתמים מקביליים רבים עבור מערכים, רשימות, עצים וגרפים. אף כי PRAM מתעלם מהיבטים חשובים רבים של מכונות מקביליות ממשיות, המאפיינים המהותיים של אלגוריתמים מקביליים נוטים להתקיים מעבר למודלים שעבורם מתוכננים האלגוריתמים. אם ביצועיו של אלגוריתם PRAM אחד טובים יותר מאלה של אלגוריתם PRAM אחר, הביצועים היחסיים אינם משתנים בדרך כלל במידה משמעותית כאשר מממשים את שני האלגוריתמים על מחשב מקבילי ממשי.

מודל ה־PRAM

באיור 30.1 מוצגת הארכיטקטורה הבסיסית של **מכונת הגישה־האקראית המקבילית** (PRAM, parallel random-access machine). המכונה מורכבת מ־p מעבדים רגילים (סדרתיים), $P_0, P_1, \ldots, P_{p-1}$, שמקום האחסון המשמש אותם הוא זיכרון גלובלי משותף. כל המעבדים יכולים לקרוא מן הזיכרון הגלובלי או לכתוב אליו "במקביל" (בו־זמנית). המעבדים יכולים גם לבצע במקביל פעולות אריתמטיות ולוגיות שונות.



איור 30.1 הארכיטקטורה הבסיסית של מודל PRAM. ישנם p מעבדים $P_0, P_1, \ldots, P_{p-1}$, המחוברים לזיכרון משותף. כל מעבד יכול לבצע גישה למילה כלשהי בזיכרון המשותף ביחידת זמן.

הנחת המפתח הנוגעת לביצועים האלגוריתמיים על מודל PRAM היא, שזמן הריצה יכול להימדד על־ידי מספר הגישות המקביליות לזיכרון שאלגוריתם מבצע. הנחה זו אינה אלא הכללה ישירה של מודל RAM רגיל, שבו מספר הגישות לזיכרון הוא מדד אסימפטוטי שאינו נופל מכל מדד אחר של זמן הריצה. הנחה פשוטה זו ותשרת אותנו היטב בעת שנבחן אלגוריתמים מקביליים, אף על פי שמחשבים מקביליים ממשיים אינם יכולים לבצע גישות מקביליות לזיכרון המשותף ביחידת זמן אחת: זמן הגישה לזיכרון גדל עם מספר המעבדים במחשב המקבילי.

אף על פי כן, עבור אלגוריתמים מקביליים הניגשים לנתונים בצורה אקראית, ההנחה שפעולות זיכרון מתבצעות ביחידת זמן אחת ניתנת להצדקה. מכונות מקביליות ממשיות יש להן בדרך כלל רשת תקשורת שיכולה לתמוך בהפשטה של זיכרון גלובלי. גישה לנתונים דרך הרשת היא פעולה איטית יחסית בהשוואה לפעולות אריתמטיות ואחרות. לכן, מניית מספר הגישות המקביליות לזיכרון שמבצעים שני אלגוריתמים מקביליים אכן מניבה למעשה אומדן מדויק למדי של ביצועיהם היחסיים. האופן העיקרי שבו מפרות מכונות ממשיות את ההפשטה של ה־PRAM המניחה גישה לזיכרון ביחידת זמן אחת, הוא בכך שקיימות תבניות גישה לזיכרון שהן מהירות יותר מאחרות. אולם, כקירוב ראשון, ההנחה של יחידת הזמן במודל PRAM היא הנחה סבירה למדי.

זמן הריצה של אלגוריתם מקבילי תלוי במספר המעבדים המבצעים את האלגוריתם כמו גם בגודל הקלט של הבעיה. לכן, ככלל, כאשר מנתחים אלגוריתמי PRAM, יש להביא בחשבון את הזמן וגם את מספר המעבדים — בניגוד לאלגוריתמים סדרתיים, שבהם הניתוח התמקד בעיקר בזמן. בדרך כלל קיימים יחסי "קח־ותן" בין מספר המעבדים שהאלגוריתם משתמש בהם לבין זמן הריצה שלו. בסעיף 30.3 נדון ביחסי "קח־ותן" אלה.

גישות בו־זמניות לעומת גישות בלעדיות לזיכרון

אלגוריתם **קריאה־בו־זמנית** (concurrent-read) הוא אלגוריתם PRAM שבמהלך ביצועו יכולים מעבדים רבים לקרוא מאותו מיקום בזיכרון המשותף בעת ובעונה אחת. אלגוריתם **קריאה־בלעדית** (exclusive-read) הוא אלגוריתם PRAM שבו שני מעבדים לעולם אינם יכולים לקרוא בו בזמן מאותו מיקום בזיכרון. אנו עושים הבחנה דומה בדבר האפשרות שכמה מעבדים יכתבו בו־זמנית באותו מיקום בזיכרון, ומחלקים אלגוריתמי PRAM לאלגוריתמי **כתיבה־בו־זמנית** (concurrent-write) ולאלגוריתמי **כתיבה־בלעדית** (exclusive-write). שמות מקוצרים מקובלים לסוגי האלגוריתמים שמנינו הם:

- **EREW**: קריאה בלעדית וכתיבה בלעדית,
- **CREW**: קריאה בו־זמנית וכתיבה בלעדית,
- **ERCW**: קריאה בלעדית וכתיבה בו־זמנית,
- **CRCW**: קריאה בו־זמנית וכתיבה בו־זמנית.

(את הקיצורים האלה מבטאים בדרך כלל לא כמילים אלא כמחרוזות של אותיות.)

מבין טיפוסי אלגוריתמים אלה, המקורים הקיצוניים — EREW ו־CRCW — הם הפופולריים ביותר. מודל PRAM התומך באלגוריתמי EREW בלבד נקרא **EREW PRAM** וכזה התומך באלגוריתמי CRCW נקרא **CRCW PRAM**. ברור כי CRCW PRAM מסוגל לבצע אלגוריתמי EREW, אבל EREW PRAM אינו מסוגל לתמוך ישירות בגישות הבו־זמניות לזיכרון הנדרשות באלגוריתמי CRCW. החומרה שעליה בנוי EREW PRAM פשוטה יחסית, ולכן מהירה, משום שאינה צריכה לטפל בהתנגשויות אפשריות של פעולות קריאה וכתיבה מן הזיכרון ואליו. מודל CRCW PRAM דורש תמיכה גדולה יותר של החומרה אם ברצוננו שההנחה של גישה ביחידת זמן אכן תספק מדד מדויק דיו לביצועים האלגוריתמיים, אבל הוא מספק מודל תכנות שהוא אולי ישיר יותר מאשר EREW PRAM.

מבין שני טיפוסי האלגוריתמים הנותרים — CREW ו־ERCW — הוקדשה בספרות תשומת לב רבה יותר למודל CREW. אולם מנקודת מבט מעשית, התמיכה בפעולות כתיבה בו־זמניות אינה קשה יותר מאשר התמיכה בפעולות קריאה בו־זמניות. בפרק זה נתייחס בדרך כלל לאלגוריתם כאל אלגוריתם CRCW אם הוא מכיל קריאות בו־זמניות או כתיבות בו־זמניות, בלי הבחנות נוספות. בהיבטים העדינים של הבחנה זו נדון בסעיף 30.2.

כאשר מעבדים רבים כותבים אל אותו מיקום בזיכרון באלגוריתם CRCW, תוצאת הכתיבות המקבילות אינה מוגדרת היטב ללא פירוט נוסף. בפרק זה נשתמש במודל **CRCW־משותף** (common-CRCW): כאשר מעבדים אחדים כותבים אל אותו מיקום בזיכרון, כולם חייבים לכתוב ערך משותף (אותו ערך). קיימים בספרות כמה טיפוסים חלופיים של PRAM, המטפלים בבעיה זו מתוך הנחות אחרות. האפשרויות האחרות כוללות:

- **ערך שרירותי**: הערך המאוחסן נבחר באופן שרירותי מבין הערכים הנכתבים;
- **קדימות**: הערך המאוחסן הוא זה שנכתב על־ידי המעבד בעל האינדקס הנמוך ביותר;
- **צירוף**: הערך המאוחסן הינו צירוף מסוים של הערכים הנכתבים.

במקרה האחרון, הצירוף הנבחר הוא בדרך כלל פונקציה אסוציאטיבית וקומוטטיבית כלשהי, כגון חיבור (הערך המאוחסן הוא סכום הערכים הנכתבים) או מקסימום (הערך המאוחסן הוא המקסימלי מבין הערכים הנכתבים).

סינכרון ובקרה

כדי שאלגוריתמי PRAM יפעלו כהלכה הם חייבים להיות מסונכרנים היטב. כיצד מושג סיכרון כזה? כמו כן, המעבדים באלגוריתמי PRAM חייבים לזהות לעתים קרובות תנאי עצירה של לולאה התלויים במצבם של כל המעבדים. כיצד ממומשת פונקציית בקרה זו?

אנו לא נדון בהרחבה בנושאים אלה. מחשבים מקביליים ממשיים רבים משתמשים ברשת בקרה המחברת את המעבדים ומסייעת בסינכרון ובזיהוי תנאי עצירה. בדרך כלל, רשת בקרה יכולה לממש פונקציות אלה באותה מהירות שבה יכולה רשת ניתוב לממש גישות לזיכרון גלובלי.

למטרותינו, די להניח קיומו של סיכרון פנימי הדוק בין המעבדים. כל המעבדים מבצעים באותו זמן אותם משפטים, ואף מעבד אינו רץ קדימה בעוד האחרים נותרים מאחוריו בקוד. כאשר נבחן

את האלגוריתם המקבילי הראשון שלנו, נצביע על המקומות שבהם אנו מניחים כי המעבדים מסונכרנים ביניהם.

כדי לגלות תנאי עצירה של לולאה מקבילית התלויה במצבם של כל המעבדים, נניח שניתן לבדוק תנאי עצירה מקבילי באמצעות רשת הבקרה בזמן $O(1)$. חלק מהמודלים מטיפוס EREW PRAM המתוארים בספרות אינם מניחים הנחה זו, ואז יש לכלול את הזמן (הלוגריתמי) לבדיקת תנאי הלולאה בזמן הריצה הכולל (ראה תרגיל 30.1.8). כפי שנראה בסעיף 30.2, אלגוריתמי CRCW PRAM אינם זקוקים לרשת בקרה כדי לבדוק עצירה: הם יכולים לזהות עצירה של לולאה מקבילית בזמן $O(1)$ על־ידי שימוש בכתיבות בו־זמניות.

תמצית הפרק

סעיף 30.1 מציג את הטכניקה של הקפצת מצביעים, המספקת דרך מהירה לטיפול מקבילי ברשימות. נראה כיצד ניתן להשתמש בהקפצת מצביעים לחישובי רישות ברשימות, וכיצד ניתן להתאים אלגוריתמים מהירים עבור רשימות לטיפול בעצים. סעיף 30.2 דן בכוח החישוב היחסי של אלגוריתמי CRCW ו־EREW ומראה שגישה בו־זמנית לזיכרון מגדילה את כוח החישוב.

בסעיף 30.3 נציג את משפט ברנט (Brent), המראה כיצד ניתן לדמות ביעילות מעגלים צירופיים באמצעות PRAM. כמו כן נדון בסעיף זה בנושא החישוב של יעילות העבודה, ונציג את התנאים שבהם ניתן לתרגם ביעילות אלגוריתם PRAM על p מעבדים לאלגוריתם PRAM על p' מעבדים, עבור כל $p' < p$. בסעיף 30.4 נחזור לבעיה של ביצוע חישוב רישות על רשימה מקושרת ונראה כיצד אלגוריתם אקראי יכול לבצע את החישוב בצורה יעילה. לבסוף, בסעיף 30.5 נראה כיצד ניתן לשבור סימטריה בצורה מקבילית בזמן קצר בהרבה מזמן לוגריתמי באמצעות אלגוריתם דטרמיניסטי.

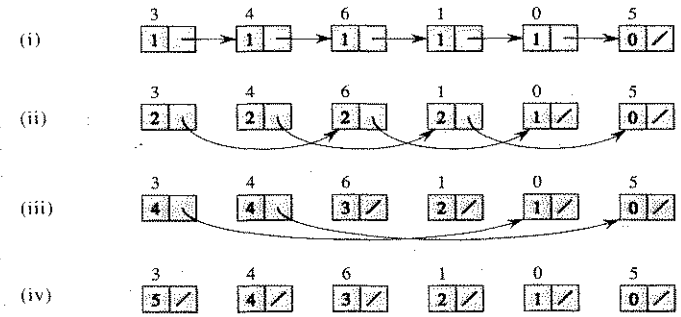
האלגוריתמים המקביליים שבפרק זה לקוחים רובם מתחום תורת הגרפים. אין הם מייצגים אלא מבחר מצומצם ממגוון האלגוריתמים המקביליים הקיימים היום. אולם הטכניקות המוצגות בפרק זה מיטיבות למדי לייצג את הטכניקות המשמשות אלגוריתמים מקביליים בתחומים אחרים של מדעי המחשב.

30.1 הקפצת מצביעים

אלגוריתמים העוסקים במצביעים נמנים על אלגוריתמי PRAM המעניינים ביותר. בסעיף זה נחקור שיטה חזקה שנקראת "הקפצת מצביעים", המניבה אלגוריתמים מהירים לטיפול ברשימות. בפרט, נציג אלגוריתם שזמן ריצתו $O(\lg n)$ אשר מחשב את מרחקו של כל עצם מסוף הרשימה ברשימה בת n עצמים. לאחר מכן נשנה אלגוריתם זה כך שיבצע "חישוב רישות מקבילי" על רשימה בת n עצמים בזמן $O(\lg n)$. לבסוף, נחקור שיטה המאפשרת להפוך בעיות רבות על עצים לבעיות על רשימות, שאותן ניתן לפתור באמצעות הקפצת מצביעים. כל האלגוריתמים בסעיף זה הם אלגוריתמי EREW: לא נדרשות בהם גישות בו־זמניות לזיכרון גלובלי.

30.1.1 מיקום איברים ברשימה

האלגוריתם המקבילי הראשון שלנו פועל על רשימות. ניתן לאחסן רשימה על PRAM בדרך דומה למדי לדרך שבה מאחסנים רשימות על RAM רגיל. אולם כדי לבצע פעולות במקביל על רשימת עצמים, נוח לקבוע עבור כל עצם מעבד "אחראי". אנו נניח כי מספר המעבדים שווה למספר העצמים ברשימה, וכי המעבד ה-i אחראי לעצם ה-i. לדוגמה, באיור 30.2(i) מופיעה רשימה מקושרת המורכבת מסדרת העצמים $\langle 3, 4, 6, 1, 0, 5 \rangle$. מכיוון שישנו מעבד אחד לכל עצם ברשימה, ניתן לבצע פעולות על כל עצם ברשימה על-ידי המעבד האחראי שלו בזמן $O(1)$.



איור 30.2 מציאת המרחק של כל עצם מסוף הרשימה ברשימה בת n עצמים בזמן $O(\lg n)$ באמצעות הקפצת מצביעים. **(i)** רשימה מקושרת המיוצגת ב-PRAM, עם ערכי d ההתחלתיים. עם סיומו של האלגוריתם, ערכו של כל d הוא מרחקו של העצם שלו מסוף הרשימה. מעל לכל עצם מופיע מספרו של המעבד האחראי עליו. **(ii)-(iv)** המצביעים וערכי d לאחר כל איטרציה של לולאת ה-**while** באלגוריתם LIST-RANK.

נניח שנתונה לנו רשימה חד-מקושרת L בת n עצמים, וברצוננו לחשב, עבור כל עצם ב-L, את מרחקו מסוף הרשימה. בניסוח פורמלי יותר, אם $next$ הוא שדה המצביע, ברצוננו לחשב עבור כל עצם i ברשימה ערך $d[i]$ המקיים:

$$d(i) = \begin{cases} 0 & \text{אם } next[i] = \text{NIL} \\ d[next[i]] + 1 & \text{אם } next[i] \neq \text{NIL} \end{cases}$$

אנו מכנים את בעיית חישובם של ערכי d בשם **בעיית המיקום ברשימה** (list-ranking problem).

אחת הדרכים לפתרון בעיית המיקום ברשימה היא פשוט להתחיל בסוף הרשימה ולחשב מרחקים אחורה. שיטה זו מתבצעת בזמן $\Theta(n)$, שכן העצם ה-k מהסוף חייב להמתין עד שייקבעו מרחקיהם מסוף הרשימה של $k-1$ העצמים הנמצאים אחריו ברשימה, קודם שייקבע מרחקו שלו מסוף הרשימה. פתרון זה הוא במהותו פתרון סדרתי.

פתרון מקבילי יעיל, הדורש זמן של $O(\lg n)$ בלבד, נתון על-ידי הפסאודו-קוד המקבילי הבא:


```
LIST-RANK(L)
1  for each processor i, in parallel
2      do if next[i] = NIL
3           then d[i] ← 0
4           else d[i] ← 1
5  while there exists an object i such that next[i] ≠ NIL
6      do for each processor i, in parallel
7           do if next[i] ≠ NIL
8                then d[i] ← d[i] + d[next[i]]
9                     next[i] ← next[next[i]]
```

איור 30.2 מראה כיצד האלגוריתם מחשב את המרחקים. בכל חלק באיור מופיע מצב הרשימה לפני איטרציה של לולאת ה-**while** שבשורות 5-9. בחלק (i) מוצגת הרשימה מיד לאחר האתחול. באיטרציה הראשונה, לחמישה עצמים ברשימה יש מצביעים השונים מ-NIL, ולפיכך מבוצעות שורות 8-9 על-ידי המעבדים האחראיים עליהם. התוצאה מוצגת בחלק (ii) של האיור. באיטרציה השנייה, רק ארבעת העצמים הראשונים הם בעלי מצביעים שונים מ-NIL, תוצאת איטרציה זו מוצגת בחלק (iii). האיטרציה השלישית פועלת על שני העצמים הראשונים בלבד, והתוצאה הסופית, שבה לכל העצמים יש מצביעים NIL, מוצגת בחלק (iv).

הרעיון המממומש בשורה 9, שבה מתבצעת ההשמה $next[i] \leftarrow next[next[i]]$, בכל המצביעים $next[i]$ השונים מ-NIL, נקראת **הקפצת מצביעים** (pointer jumping). יש לשים לב כי הקפצת מצביעים משנה את שדות המצביעים והורסת בכך את מבנה הרשימה. אם יש לשמור על מבנה הרשימה, אזי מכינים עותקים של המצביעים $next$ ומשתמשים בהם לחישוב המרחקים.

נכונות

LIST-RANK מקיימת את האינווריאנטה שבתחילתה של כל איטרציה של לולאת ה-**while** שבשורות 5-9, עבור כל עצם i, אם מסכמים את ערכי d בתת-רשימה שראשה i, מקבלים את המרחק הנכון מ-i לסוף הרשימה המקורית L. באיור 30.2(ii) לדוגמה, התת-רשימה שבראשה העצם 3 היא הסדרה $\langle 3, 6, 0 \rangle$ שערכי d שלה, 2, 2 ו-1, מסתכמים ל-5 — מרחקו של 3 מסופה של הרשימה המקורית. הסיבה לשימורה של האינווריאנטה זו היא, שכאשר עצם "מדלג" על העוקב לו ברשימה, הוא מוסיף את ערך d של העוקב לערך d שלו עצמו.

יש לשים לב כי כדי שאלגוריתם הקפצת מצביעים זה יפעל כהלכה, הגישות המקבילות לזיכרון חייבות להיות מסונכרנות. כל ביצוע של שורה 9 עשוי לעדכן כמה מצביעי $next$. אנו סומכים על כך שכל פעולות הקריאה מן הזיכרון באגף הימני של משפט ההשמה (קריאת $next[next[i]]$) יתבצעו לפני כל פעולות הכתיבה בזיכרון (כתיבת $next[i]$) שבאגף השמאלי.

הבה נראה עתה מדוע LIST-RANK הוא אלגוריתם EREW. מכיוון שכל מעבד אחראי לכל היותר לעצם אחד, כל פעולת קריאה וכתיבה בשורות 2-7 היא בלעדית, וכך גם כל פעולות הכתיבה בשורות 8-9. נשים לב כי הקפצת מצביעים משמרת את האינווריאנטה שעבור כל שני עצמים שונים i ו-j מתקיים אחד מהשניים: או $next[i] \neq next[j]$ או $next[i] = next[j] = \text{NIL}$. אינווריאנטה זו בוודאי נכונה עבור הרשימה ההתחלתית, ושורה 9 משמרת אותה. מכיוון שכל הערכים $next$ שאינם NIL שונים זה מזה, כל הקריאות שבשורה 9 הן בלעדיות.

אולם עלינו להניח שמידה מסוימת של סינכרון אכן מתבצעת בשורה 8, אם כל הקריאות אמורות להיות בלעדיות. בפרט, אנו דורשים שכל המעבדים i יקראו קודם את $d[i]$ ולאחר מכן את $d[next[i]]$. עם סינכרון כזה, אם עצם i הוא בעל $next[i] \neq \text{NIL}$ וקיים עצם אחר j המצביע

על i (כלומר, $next[j] = i$), אזי הקריאה הראשונה שולפת את $d[i]$ עבור המעבד i והקריאה השנייה שולפת את $d[i]$ עבור המעבד j. לכן, LIST-RANK הוא אלגוריתם EREW.

מכאן ואילך, נתעלם מפרטי סינכרון כאלה ונניח שאלגוריתם PRAM וסביבת התכנות שלו בפסאודו־קוד פועלים באופן עקבי ומסונכרן, כאשר כל המעבדים מבצעים קריאות וכתיבות באותו זמן.

ניתוח

עתה נראה שאם רשימה L מכילה n עצמים, אזי השגרה LIST-RANK מתבצעת בזמן $O(\lg n)$. מכיוון שהאתחול מתבצע בזמן $O(1)$ וכל איטרציה של לולאת ה־**while** מתבצעת בזמן $O(1)$, אזי די להראות שמתבצעות בדיוק $\lceil \lg n \rceil$ איטרציות. הבחנת המפתח היא שכל שלב של הקפצת מצביעים הופך כל רשימה לשתי רשימות משולבות זו בזו: האחת מורכבת מהעצמים במקומות הזוגיים והשנייה מורכבת מהעצמים במקומות האי־זוגיים. לכן, כל שלב של הקפצת מצביעים מכפיל את מספר הרשימות ומקטין למחצית את האורכים שלהן. לפיכך, לאחר $\lceil \lg n \rceil$ איטרציות, כל אחת מהרשימות מכילה איבר אחד בלבד.

אנו מניחים שבדיקת תנאי העצירה של האלגוריתם בשורה 5 אורכת זמן של $O(1)$, בשל רשת הבקרה ב־EREW PRAM. בתרגיל 30.1.8 תתבקש לתאר מימוש EREW של LIST-RANK שרץ בזמן $O(\lg n)$ ואשר מבצע את בדיקת תנאי העצירה במפורש בפסאודו־קוד.

לבד מזמן הריצה המקבילי, קיים מדד מעניין נוסף לביצועיהם של אלגוריתמים מקביליים. אנו מגדירים את ה**עבודה** המבוצעת על־ידי אלגוריתם מקבילי כמכפלת זמן הריצה שלו במספר המעבדים הנדרשים לביצועו. אינטואיטיבית, העבודה היא כמות החישוב המבוצעת על־ידי מכונת גישה אקראית סדרתית כאשר היא מדמה את האלגוריתם המקבילי.

השגרה LIST-RANK מבצעת עבודה של $\Theta(n \lg n)$, שכן דרושים לה n מעבדים והיא רצה בזמן $\Theta(\lg n)$. האלגוריתם הסדרתי הישיר לבעיה של מיקום איברים ברשימה רץ בזמן $\Theta(n)$, והדבר מצביע על כך שהשגרה LIST-RANK מבצעת עבודה רבה יותר ממה שנדרש בהכרח, אולם התוספת מתבטאת בגורם לוגריתמי בלבד.

אנו מגדירים אלגוריתם PRAM A כאלגוריתם **יעיל בעבודה** (work-efficient) ביחס לאלגוריתם (סדרתי או מקבילי) אחר B, עבור אותה בעיה, אם העבודה שמבצע A אינה עולה על העבודה שמבצע B כפול גורם קבוע. וביתר פשטות נאמר שאלגוריתם PRAM A הוא **יעיל בעבודה** אם הוא יעיל בעבודה ביחס לאלגוריתם הטוב ביותר שרץ על מעבד RAM סדרתי. מאחר שהאלגוריתם הסדרתי הטוב ביותר עבור מיקום איברים ברשימה רץ על RAM סדרתי בזמן $\Theta(n)$, הרי שהשגרה LIST-RANK אינה יעילה בעבודה. בסעיף 30.4 נציג אלגוריתם מקבילי יעיל בעבודה עבור מיקום איברים ברשימה.

30.1.2 חישוב מקבילי של רישות של רשימות

אפשרויות היישום של שיטת הקפצת המצביעים חורגות הרבה מעבר למיקום איברים ברשימה. סעיף 29.2.2 מראה כיצד, בהקשר של מעגלים אריתמטיים, ניתן להשתמש בחישוב "רישות" לביצוע מהיר של חיבור בינרי. עתה נחקור כיצד ניתן להשתמש בהקפצת מצביעים לביצוע חישובי רישות. אלגוריתם EREW שלנו עבור בעיית הרישות רץ בזמן $O(\lg n)$ על רשימות בנות n עצמים.

חישוב רישות (prefix computation) מוגדר במונחי אופרטור בינרי אסוציאטיבי $\otimes$. הקלט לחישוב זה הוא סדרה $\langle x_1, x_2, \ldots, x_n \rangle$ והפלט הוא סדרה $\langle y_1, y_2, \ldots, y_n \rangle$ כך ש־$y_1 = x_1$ ועבור $k = 2, 3, \ldots, n$, מתקיים:

$$\begin{aligned} y_k &= y_{k-1} \otimes x_k \\ &= x_1 \otimes x_2 \otimes \cdots \otimes x_k \end{aligned}$$

במילים אחרות, כל y_k מתקבל על־ידי "מכפלת" k האיברים הראשונים בסדרת ה־x_k — מכאן המונח "רישא". (בהגדרה שבפרק 29 האינדקסים של איברי הסדרות מתחילים מ־0, ואילו כאן הם מתחילים מ־1 — הבדל בלתי משמעותי.)

כדוגמה לחישוב רישות, נניח כי כל איבר ברשימה בת n עצמים מכיל את הערך 1, ויהי $\otimes$ החיבור הרגיל. מכיוון שעבור $k = 1, 2, \ldots, n$, האיבר ה־k ברשימה מכיל את הערך $x_k = 1$, חישוב רישות נותן $y_k = k$, האינדקס של האיבר ה־k. לכן, דרך אחרת למיקום איברים ברשימה היא להפוך את סדר האיברים ברשימה (פעולה הניתנת לביצוע בזמן $O(1)$), לבצע חישוב רישות כזה, ולחסר 1 מכל ערך שחושב.

עתה נראה כיצד אלגוריתם EREW מסוגל לחשב במקביל רישות ברשימה בת n עצמים בזמן $O(\lg n)$. מטעמי נוחות, נגדיר את הסימון הבא:

$$[i, j] = x_i \otimes x_{i+1} \otimes \cdots \otimes x_j$$

עבור שלמים i ו־j בתחום $1 \le i \le j \le n$. אזי, $[k, k] = x_k$ עבור $k = 1, 2, \ldots, n$, וכן:

$$[i, k] = [i, j] \otimes [j + 1, k]$$

עבור $1 \le i \le j < k \le n$. במונחי סימון זה, מטרת חישוב הרישות היא לחשב את $y_k = [1, k]$ עבור $k = 1, 2, \ldots, n$.

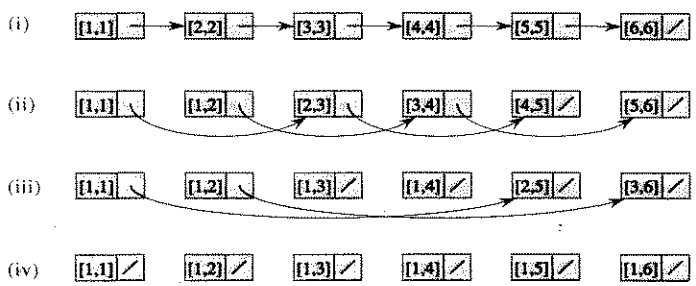
כאשר אנו מבצעים חישוב רישות על רשימה, אנו רוצים שסדר האיברים בסדרת הקלט $\langle x_1, x_2, \ldots, x_n \rangle$ ייקבע על־ידי האופן שבו מקושרים העצמים ברשימה, ולא על־ידי האינדקסים של העצמים במערך שבו הם מאוחסנים בזיכרון. (בתרגיל 30.1-2 תתבקש להציג אלגוריתם רישות עבור מערכים.) אלגוריתם EREW שלהלן מתחיל עם ערך $x[i]$ בכל עצם i ברשימה L. אם עצם i הוא העצם ה־k מתחילת הרשימה, אזי $x[i] = x_k$ הוא האיבר ה־k בסדרת הקלט. לפיכך, חישוב הרישות המקבילי נותן $y[i] = y_k = [1, k]$.

```
LIST-PREFIX(L)
1  for each processor i, in parallel
2      do y[i] ← x[i]
3  while there exists an object i such that next[i] ≠ NIL
4      do for each processor i, in parallel
5          do if next[i] ≠ NIL
6              then y[next[i]] ← y[i] ⊗ y[next[i]]
7                   next[i] ← next[next[i]]
```

הפסאודו־קוד ואיור 30.3 מצביעים על הדמיון בין אלגוריתם זה לבין LIST-RANK. ההבדלים היחידים הם האתחול והעדכונים של ערכי d או y. ב־LIST-RANK, המעבד ה־i מעדכן את $d[i]$ — ערך d שלו עצמו, ואילו ב־LIST-PREFIX, המעבד ה־i מעדכן את $y[next[i]]$ — ערך y של מעבד אחר. נשים לב כי השגרה LIST-PREFIX היא אלגוריתם EREW מאותה סיבה שהשגרה LIST-RANK היא כזאת: הקפצת מצביעים משמרת את האינווריאנטה שעל פיה, עבור עצמים i ו־j שונים זה מזה, מתקיים אחד מהשניים: $next[i] \ne next[j]$ או $next[i] = next[j] = \text{NIL}$.

באיור 30.3 מוצג מצב הרשימה לפני כל איטרציה של לולאת ה־**while**. השגרה משמרת את האינווארינטה שעל פיה, בסוף האיטרציה ה־i של לולאת ה־**while**, המעבד ה־k מאחסן את $[max(1, k - 2^i + 1), k]$ עבור $k = 1, 2, \ldots, n$. באיטרציה הראשונה, האיבר ה־k ברשימה מצביע בתחילה על העצם ה־$(k + 1)$, פרט לכך שהעצם האחרון הוא בעל מצביע NIL. שורה 6 גורמת לכך שהעצם ה־k, עבור $k = 1, 2, \ldots, n - 1$, ישלוף את הערך $[k + 1, k + 1]$ מתוך העוקב שלו. לאחר מכן הוא מבצע את הפעולה $[k, k] \otimes [k + 1, k + 1]$, המניבה $[k, k + 1]$, ואת התוצאה הוא מאחסן בחזרה בעוקב שלו. לאחר מכן מוקפצים המצביעים *next* כמו ב־List-Rank. תוצאת האיטרציה הראשונה מוצגת באיור 30.3(ii). את האיטרציה השנייה ניתן לראות באופן דומה. עבור $k = 1, 2, \ldots, n - 2$, העצם ה־k שולף את הערך $[k + 1, k + 2]$ מתוך העוקב שלו (כפי שהוא מוגדר עתה על־ידי הערך החדש של שדה *next* שלו), ואז מאחסן בו בחזרה את $[k - 1, k] \otimes [k + 1, k + 2] = [k - 1, k + 2]$. התוצאה מוצגת באיור 30.3(iii). באיטרציה השלישית והאחרונה, רק שני העצמים הראשונים ברשימה הם בעלי מצביעים שונים מ־NIL, וכל אחד מהם שולף ערכים מן העוקב שלו ברשימה שלו. התוצאה הסופית מוצגת באיור 30.3(iv). הבחנת המפתח להבנת פעולתה של List-Prefix היא שבכל שלב, אם נבצע חישוב רישות על כל אחת מבין הרשימות הקיימות, כל עצם יקבל את הערך הנכון שלו.

מכיוון ששני האלגוריתמים משתמשים באותו מנגנון הקפצת מצביעים, ניתוח List-Prefix דומה לזה של List-Rank: זמן הריצה על EREW PRAM הוא $O(\lg n)$, והעבודה הכוללת המבוצעת היא $\Theta(n \lg n)$.



איור 30.3 האלגוריתם המקבילי List-Prefix לחישוב רישות ברשימה מקושרת. **(i)** ערך y ההתחלתי של העצם ה־k ברשימה הוא $[k, k]$. המצביע *next* של העצם ה־k מצביע על העצם ה־$(k + 1)$, או שהוא NIL עבור העצם האחרון. **(ii)-(iv)** ערכי y ו־*next* לפני כל אחת מן הבדיקות בשורה 3. בחלק **(iv)** מופיעה התוצאה הסופית, שבה עבור כל k, ערך y של העצם ה־k הוא $[k, 1]$.

30.1.3 שיטת מסלול־אוילר

בסעיף זה נציג את שיטת מסלול־אוילר ונראה כיצד ניתן ליישמה בבעיית חישוב עומקו של כל צומת בעץ בינרי n־צומתי. שלב מרכזי באלגוריתם EREW זה, שרץ בזמן $O(\lg n)$, הוא חישוב רישות מקבילי.

כדי לאחסן עצים בינריים ב־PRAM, אנו משתמשים בייצוג פשוט של עץ בינרי מן הסוג שהוצג בסעיף 11.4. כל צומת i מכיל את השדות $parent[i]$, $left[i]$ ו־$right[i]$, אשר מצביעים על האב, הבן השמאלי, והבן הימני של הצומת i, בהתאמה. הבה ננית שכל צומת מזוהה על־ידי מספר שלם אי־שלילי. מסיבות שיובהרו מיד, אנו מקשרים לכל צומת לא מעבד אחד אלא שלושה; נקרא להם המעבדים A, B ו־C של הצומת. עלינו להיות מסוגלים למפות בקלות בין צומת לשלושת מעבדיו; לדוגמה, צומת i עשוי להיות מקושר למעבדים $3i$, $3i + 1$ ו־$3i + 2$.

חישוב העומק של כל הצמתים בעץ n־צומתי מתבצע בזמן $O(n)$ על RAM סדרתי. אלגוריתם מקבילי פשוט לחישוב עומקים יוצר "גל" המתקדם מן השורש במורד העץ. הגל מגיע בו־זמנית לכל הצמתים הנמצאים באותו עומק, וכך, על־ידי הוספת 1 למונה הנישא עם הגל, ניתן לחשב את עומקו של כל צומת. אלגוריתם מקבילי זה פועל היטב על עץ בינרי שלם, שכן זמן ריצתו נמצא ביחס ישר לגובה העץ. אולם גובהו של העץ יכול להגיע עד $n - 1$, ובמקרה זה האלגוריתם ירוץ בזמן $\Theta(n)$ — זמן שאינו טוב מזה של האלגוריתם הסדרתי. אבל באמצעות שיטת מסלול־אוילר ניתן לחשב את עומקי הצמתים בזמן $O(\lg n)$ על EREW PRAM, ויהיה אשר יהיה גובהו של העץ.

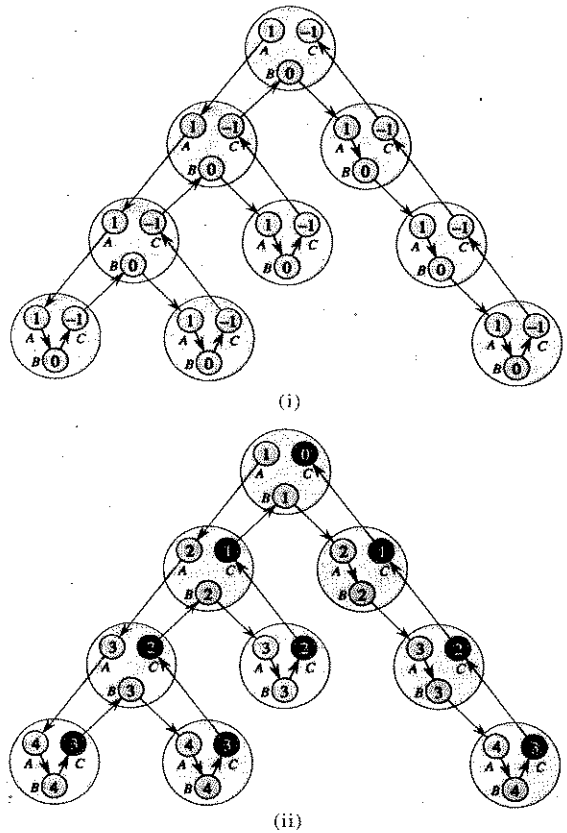
מסלול אוילר (Euler tour) של גרף הוא מעגל העובר לאורך כל קשת בדיוק פעם אחת, אף כי הוא עשוי לבקר בקדקוד יותר מפעם אחת. על פי בעיה 23-3, גרף מכוון קשיר מכיל מסלול אוילר אם ורק אם עבור כל הקדקודים v, דרגת הכניסה של v שווה לדרגת היציאה שלו. מאחר שכל קשת בלתי־מכוונת (u, v) בגרף בלתי־מכוון ממופה לשתי קשתות מכוונות (u, v) ו־(v, u) בגרסה המכוונת, הרי שהגרסה המכוונת של כל גרף בלתי־מכוון קשיר — ולכן של כל עץ בלתי־מכוון — מכילה מסלול אוילר.

כדי לחשב את עומקי הצומתים בעץ בינרי T, אנו יוצרים תחילה מסלול אוילר של הגרסה המכוונת של T (כשמתייחסים אליו כאל גרף בלתי־מכוון). המסלול מתאים לטריקה של העץ והוא מיוצג באיור 30.4(i) על־ידי רשימה מקושרת העוברת דרך צומתי העץ. מבנה המסלול הוא כדלקמן:

- המעבד A של צומת מצביע על המעבד A של בנו השמאלי של הצומת, אם קיים כזה, ואחרת, על המעבד B של הצומת עצמו.
- המעבד B של צומת מצביע על המעבד A של בנו הימני של הצומת, אם קיים כזה, ואחרת, על המעבד C של הצומת עצמו.
- המעבד C של צומת מצביע על המעבד B של אביו של הצומת אם הצומת הוא בן שמאלי, ועל המעבד C של אביו של הצומת אם הצומת הוא בן ימני. המעבד C של השורש מצביע על NIL.

אם כן, ראש הרשימה המקושרת שיוצר מסלול אוילר הוא המעבד A של השורש, וזנבה הוא המעבד C של השורש. בהינתן המצביעים המרכיבים את העץ המקורי, ניתן לבנות מסלול אוילר בזמן $O(1)$.

עם השלמת בנייתה של הרשימה המקושרת המייצגת את מסלול אוילר של T, אנו מציבים 1 בכל מעבד A, 0 בכל מעבד B, ו־-1 בכל מעבד C, כמודגם באיור 30.4(i). לאחר מכן אנו מבצעים חישוב רישות מקבילי, כשהפעולה האסוציאטיבית היא חיבור רגיל, כפי שעשינו בסעיף 30.1.2. באיור 30.4(ii) מוצגת תוצאת חישוב הרישות המקבילי.



איור 30.4 שימוש בשיטת מסלול אוילר לחישוב עומקו של כל צומת בעץ בינרי. **(i)** מסלול אוילר הוא רשימה המתאימה לסריקה של העץ. כל מעבד מכיל מספר המשמש את חישוב הרישות המקבילי לחישוב עומקי הצמתים. **(ii)** תוצאת חישוב הרישות המקבילי על הרשימה המקושרת מ־(i). המעבד C של כל צומת (הצבוע באיור בשחור) מכיל את עומקו של הצומת. (ניתן לאמת את תוצאתו של חישוב רישות זה על־ידי חישובו באופן סדרתי.)

אנו טוענים שלאחר ביצוע חישוב הרישות המקבילי, עומקו של כל צומת נמצא במעבד C שלו. מדוע? המספרים מוצבים במעבדים A, B ו־C באופן כזה שהתוספת נטו לסכום המצטבר כתוצאה מביקור בתת־עץ היא 0. המעבד A של כל צומת i תורם 1 לסכום המצטבר בתת־עץ השמאלי של i, ודבר זה משקף את העובדה שעומקו של הבן השמאלי של i גדול ב־1 מעומקו של i. המעבד B תורם 0 משום שעומקו של הבן השמאלי של הצומת i שווה לעומקו של הבן הימני של הצומת i. המעבד C תורם 1−, כך שמנקודת מבטו של אביו של הצומת i, הביקור כולו בתת־עץ המושרש בצומת i לא השפיע כלל על הסכום המצטבר.

את הרשימה המייצגת את מסלול אוילר ניתן לחשב בזמן $O(1)$. הרשימה מכילה $3n$ עצמים, ולפיכך חישוב הרישות המקבילי מתבצע בזמן $O(\lg n)$ בלבד. לכן, הזמן הכולל לחישוב כל עומקי הצומתים הוא $O(\lg n)$. מאחר שלא נדרשות שום גישות בו־זמניות לזיכרון, האלגוריתם הוא אלגוריתם EREW.

תרגילים

30.1-1
כתוב אלגוריתם EREW שרץ בזמן $O(\lg n)$, אשר קובע עבור כל עצם ברשימה בת n עצמים אם הוא העצם האמצעי (ה־$\lfloor n/2 \rfloor$).

30.1-2
כתוב אלגוריתם EREW שרץ בזמן $O(\lg n)$ לביצוע חישוב רישות של מערך $x[1..n]$. אל תשתמש במצביעים, אלא בצע ישירות את חישובי האינדקסים.

30.1-3
נניח שכל עצם ברשימה L בת n עצמים צבוע באדום או בכחול. כתוב אלגוריתם EREW יעיל ליצירת שתי רשימות מן העצמים ב־L: האחת מורכבת מהעצמים הכחולים והשנייה — מהעצמים האדומים.

30.1-4
EREW PRAM מכיל n עצמים המפוזרים בין כמה רשימות מעגליות זרות. כתוב אלגוריתם יעיל אשר קובע עבור כל רשימה עצם נציג שרירותי, ומציג לכל עצם ברשימה את זהותו של הנציג. הנח שכל מעבד מכיר את האינדקס הייחודי שלו.

30.1-5
כתוב אלגוריתם EREW שרץ בזמן $O(\lg n)$ לחישוב גודל התת־עץ המושרש בכל צומת בעץ בינרי n־צומתי. (**רמז**: השתמש בהפרש בין שני ערכים בסכום המצטבר לאוכלוסיה של מסלול אוילר.)

30.1-6
כתוב אלגוריתם EREW יעיל למספור הצמתים בעץ בינרי כלשהו בסדר המתקבל על פי סריקה תחילית, תוכית וסופית.

30.1-7
הרחב את שיטת מסלול אוילר מעצים בינריים לעצים סדורים בעלי דרגות צמתים כלשהן. בפרט, תאר ייצוג עבור עצים סדורים המאפשר ליישם את שיטת מסלול אוילר. כתוב אלגוריתם EREW לחישוב עומקי הצמתים בעץ סדור n־צומתי בזמן $O(\lg n)$.

30.1-8
תאר מימוש EREW של List-Rank, שרץ בזמן $O(\lg n)$, ואשר מבצע במפורש את בדיקת תנאי העצירה של הלולאה. (**רמז**: שלב את הבדיקה בגוף הלולאה.)

30.2 אלגוריתמי CRCW לעומת אלגוריתמי EREW

הוויכוח בשאלה אם החומרה של מחשב מקבילי צריכה או אינה צריכה לאפשר גישות בו־זמניות לזיכרון הוא קשה להכרעה. יש הטוענים שמנגנוני החומרה התומכים באלגוריתמי CRCW יקרים מדי ושכיחות השימוש בהם נמוכה מכדי להצדיק את בנייתם. אחרים מתלוננים שמודל התכנות שמספקים אלגוריתמי PRAM מוגבל מדי. התשובה לשאלה נמצאת מן הסתם אי שם באמצע,

ואכן הוצעו כמה מודלים של פשרה. עם זאת, ניתן להפיק תועלת מדיון ביתרונות האלגוריתמיים שמספקות גישות בו־זמניות לזיכרון.

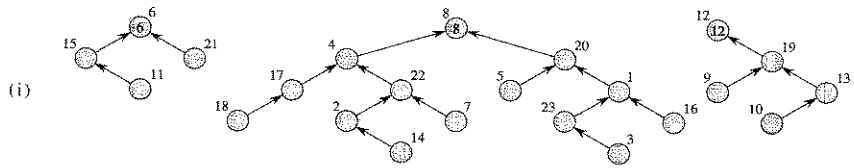
בסעיף זה נראה שקיימות בעיות אשר עבורן ביצועיו של אלגוריתם CRCW עולים על ביצועי האלגוריתם EREW הטוב ביותר האפשרי. עבור הבעיה של מציאת הזהויות של שורשי עצים ביער, קריאות בו־זמניות מאפשרות אלגוריתם מהיר יותר. עבור הבעיה של מציאת איבר המקסימום במערך, כותיבות בו־זמניות מאפשרות אלגוריתם מהיר יותר.

בעיה שבה עוזרות קריאות בו־זמניות

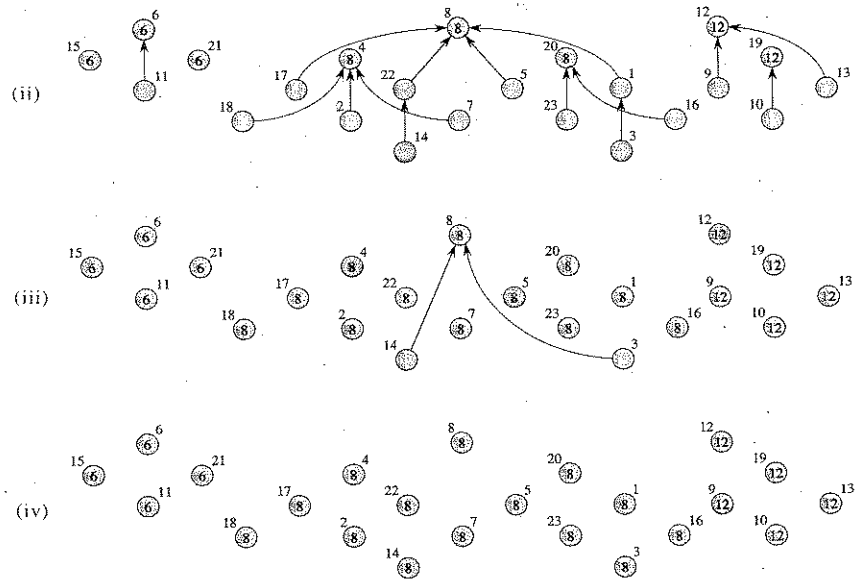
נניח שנתון לנו יער של עצים בינריים שבו כל צומת i מכיל מצביע $parent[i]$ לאביו, וברצוננו למצוא עבור כל צומת את זהותו של שורש העץ שבו הוא נמצא. אם מקשרים מעבד i לכל צומת i ביער F, אלגוריתם הקפצת המצביעים שלהלן מאחסן את זהות שורש העץ של כל צומת i ב־$root[i]$.

```
FIND-ROOTS(F)
1  for each processor i, in parallel
2      do if parent[i] = NIL
3          then root[i] ← i
4  while there exists a node i such that parent[i] ≠ NIL
5      do for each processor i, in parallel
6          do if parent[i] ≠ NIL
7              then if parent[parent[i]] = NIL
8                  then root[i] ← root[parent[i]]
9                       parent[i] ← parent[parent[i]]
```

איור 30.5 מדגים את פעולתו של אלגוריתם זה. לאחר האתחול בשורות 1-3, המוצג באיור 30.5(i), הצמתים היחידים היודעים את זהות שורשיהם הם השורשים עצמם. לולאת ה־**while** בשורות 4-9 מבצעת את הקפצת המצביעים וממלאה את השדות $root$. איורים 30.5(ii)-(iv) מראים את מצב היער לאחר האיטרציה הראשונה, השנייה והשלישית של הלולאה. כפי שאפשר לראות, האלגוריתם משמר את האינווריאנטה שלפיה, אם $parent[i] = \text{NIL}$, אזי ב־$root[i]$ כבר הוצבה זהותו של שורש העץ של הצומת.



איור 30.5 מציאת השורשים ביער של עצים בינריים על־ידי אלגוריתם CREW PRAM. מספרי הצמתים רשומים לצד הצמתים, וערכי השדות $root$ המאוחסנים רשומים בתוך הצמתים. החיצים מייצגים מצביעי $parent$. **(i)**-**(iv)** מצב העצים ביער בכל פעם שמבוצעת שורה 4 של FIND-ROOTS. אפשר לראות שבכל איטרציה, אורכי המסלולים קטנים לכדי מחצית.



איור 30.5 (המשך)

אנו טוענים כי FIND-ROOTS הוא אלגוריתם CREW שרץ בזמן $O(\lg d)$, כאשר d הוא עומקו של העץ העמוק ביותר ביער. פעולות הכתיבה היחידות מופיעות בשורות 3, 7, 8 ו־9, והן כולן בלעדיות (כלומר, זרות זו לזו), משום שבכל ארוח מהן, המעבד i כותב לתוך הצומת i בלבד. אולם פעולות הקריאה בשורות 8 ו־9 הן בו־זמניות, שכן ייתכן שהמצביעים של כמה צמתים יצביעו על אותו צומת. באיור 30.5(ii), לדוגמה, ניתן לראות שבמהלך האיטרציה השנייה של לולאת ה־**while**, $root[4]$ ו־$parent[4]$ נקראים על־ידי המעבדים 18, 2 ו־7.

זמן הריצה של FIND-ROOTS הוא $O(\lg d)$, והסיבות לכך דומות בעיקרן לאלה שהבאנו עבור LIST-RANK: בכל איטרציה קטן אורכו של כל מסלול לכדי מחצית. באיור 30.5 אפשר לראות בבירור את התכונה הזאת של האלגוריתם.

באיזו מהירות יכולים n הצמתים שביער למצוא את שורשי וזעצים הבינריים שלהם באמצעות קריאות בלעדיות בלבד? טיעון פשוט מראה שדרוש זמן של $\Omega(\lg n)$. הבחנת המפתח היא שכאשר הקריאות הן בלעדיות, בכל צעד של ה־PRAM אפשר להעתיק פיסת מידע נתונה לכל היותר למקום נוסף אחד בזיכרון; לכן, מספר המיקומים בזיכרון שיכולים להכיל פיסת מידע נתונה מוכפל לכל היותר בכל צעד. אם נתבונן בעץ יחיד, הרי שבמצב ההתחלתי, לכל היותר מיקום אחד בזיכרון מאחסן את זהותו של השורש. לאחר צעד 1, שני מיקומים לכל היותר יכולים להכיל את זהותו של השורש; לאחר k צעדים, 2^{k-1} מיקומים לכל היותר יכולים להכיל את זהותו של השורש. אם גודלו של העץ הוא $\Theta(n)$, צריכים $\Theta(n)$ מיקומים בזיכרון להכיל את זהותו של השורש כאשר האלגוריתם עוצר; לכן דרושים בסך הכל $\Omega(\lg n)$ צעדים.

בכל מקרה שבו העומק d של העץ העמוק ביותר ביער הוא $2^{o(\lg n)}$, ביצועיו האסימפטוטיים של אלגוריתם ה־FIND-ROOTS CREW עולים על אלה של כל אלגוריתם EREW. ובאופן מפורט, עבור כל יער n־צומתי שהעץ העמוק ביותר בו הוא עץ בינרי מאוזן בן $\Theta(n)$ צמתים, מתקיים $d = O(\lg n)$, ובמקרה זה FIND-ROOTS רץ בזמן $O(\lg\lg n)$. כל אלגוריתם EREW לבעיה זו חייב לרוץ בזמן $\Omega(\lg n)$, שהוא איטי יותר אסימפטוטית. לכן, קריאות בו־זמניות עוזרות בבעיה זו. תרגיל 30.2-1 מציג תרחיש פשוט יותר שבו עוזרות קריאות בו־זמניות.

בעיה שבה עוזרות כתיבות בו־זמניות

כדי להדגים את היתרון הביצועי שמציעות פעולות כתיבה בו־זמניות בהשוואה לפעולות כתיבה בלעדיות, נבחן את הבעיה של מציאת האיבר המקסימלי במערך של מספרים ממשיים. נראה שכל אלגוריתם EREW עבור בעיה זו רץ בזמן $\Omega(\lg n)$, ולא קיים אלגוריתם CREW שביצועיו טובים יותר. ניתן לפתור את הבעיה בזמן $O(1)$ באמצעות אלגוריתם CRCW־משותף, אשר בו, כאשר כמה מעבדים כותבים לאותו מיקום, כולם כותבים אותו ערך.

אלגוריתם CRCW שמוצא את המקסימום מבין n איברי מערך מניח שמערך הקלט הוא $A[0..n-1]$. האלגוריתם משתמש ב־n^2 מעבדים, כאשר כל מעבד משווה בין $A[i]$ ו־$A[j]$ עבור i ו־j כלשהם בתחום $0 \le i, j \le n-1$. למעשה, האלגוריתם מבצע מטריצת השוואות, ולכן אנו יכולים להסתכל על כל אחד מ־n^2 המעבדים לא רק כבעל אינדקס חד־ממדי ב־PRAM, אלא גם כבעל אינדקס דו־ממדי (i, j).

```
FAST-MAX(A)
 1  n ← length[A]
 2  for i ← 0 to n − 1, in parallel
 3      do m[i] ← TRUE
 4  for i ← 0 to n − 1 and j ← 0 to n − 1, in parallel
 5      do if A[i] < A[j]
 6           then m[i] ← FALSE
 7  for i ← 0 to n − 1, in parallel
 8      do if m[i] = TRUE
 9           then max[i] ← A[i]
10  return max
```

שורה 1 פשוט קובעת את אורכו של המערך A; היא צריכה להתבצע על מעבד אחד בלבד, נאמר מעבד 0. אנו משתמשים במערך $m[0..n-1]$, כאשר המעבד i אחראי על $m[i]$. אנו רוצים להגיע למצב שיתקיים $m[i] = \text{TRUE}$ אם ורק אם $A[i]$ הוא הערך המקסימלי במערך A. אנו מתחילים (שורות 2-3) בכך שאנו מאמינים כי כל איבר במערך עשוי להיות המקסימום, ואנו סומכים על ההשוואות בשורה 5 שיקבעו אילו מאיברי המערך אינם המקסימום.

איור 30.6 מדגים את שאירית האלגוריתם. בלולאה שבשורות 4-6 בודקים כל זוג סדור של איברים במערך A. עבור כל זוג $A[i]$ ו־$A[j]$, שורה 5 בודקת אם $A[i] < A[j]$. אם תוצאת השוואה זו היא TRUE, אזי $A[i]$ אינו יכול להיות המקסימום, ושורה 6 מתעדת עובדה זו על־ידי ההצבה $m[i] \leftarrow \text{FALSE}$. ייתכן שכמה זוגות (i, j) יכתבו ל־$m[i]$ בו־זמנית, אולם כולם כותבים אותו ערך: FALSE.

לכן, לאחר ביצועה של שורה 6, $m[i] = \text{TRUE}$ בדיוק עבור אותם אינדקסים i כך ש־$A[i]$ הוא מקסימלי. שורות 7-9 מציבות אז את הערך המקסימלי במשתנה max, המוחזר בשורה 10. ייתכן

שכמה מעבדים יכתבו למשתנה max, אבל אם הם אכן עושים זאת, כולם כותבים אותו ערך, כמתחייב מדרישות המודל CRCW PRAM־משותף.

		$A[j]$					
		5	6	9	2	9	m
$A[i]$	5	F	T	T	F	T	F
	6	F	F	T	F	T	F
	9	F	F	F	F	F	T
	2	T	T	T	F	T	F
	9	F	F	F	F	F	T
							max 9

איור 30.6 מציאת המקסימום מבין n ערכים בזמן $O(1)$ באמצעות אלגוריתם ה־FAST-MAX CRCW. עבור כל זוג סדור של איברים ממערך הקלט $A = \langle 5,6,9,2,9\rangle$, מופיעה במטריצה תוצאת ההשוואה $A[i] < A[j]$, כאשר T מייצג TRUE ו־F מייצג FALSE. עבור כל שורה המכילה ערך TRUE, האיבר המתאים ב־m, המופיע מימין, מקבל את הערך FALSE. איברים ב־m המכילים TRUE מתאימים לאיברים מקסימליים ב־A. בדוגמה זו, הערך 9 נכתב למשתנה max.

מכיוון שכל שלוש "הלולאות" באלגוריתם מתבצעות במקביל, FAST-MAX רץ בזמן $O(1)$. כמובן, האלגוריתם אינו יעיל בעבודה, שכן דרושים לו n^2 מעבדים, ואת בעיית מציאת המספר המקסימלי במערך ניתן לפתור באמצעות אלגוריתם סדרתי שרץ בזמן $\Theta(n)$. עם זאת, ניתן להתקרב יותר לאלגוריתם יעיל בעבודה, כפי שתתבקש להראות בתרגיל 30.2-6.

כמובן מסוים, המפתח לביצועיו של FAST-MAX טמון בכך שמודל CRCW PRAM מסוגל לבצע באמצעות n מעבדים AND בוליאני של n משתנים בזמן $O(1)$. (מאחר שיכולת זו מתקיימת במודל CRCW־משותף, היא מתקיימת גם במודלים CRCW PRAM החזקים יותר.) הקוד אכן מבצע למעשה כמה פעולות AND בעת ובעונה אחת, כאשר הוא מחשב עבור $i = 0, 1, \ldots, n-1$ את:

$$m[i] = \bigwedge_{j=0}^{n-1} (A[i] \ge A[j])$$

נוסחה זו ניתנת לגזירה מחוקי דה־מורגן (5.2). יכולת רבת עוצמה זו לביצוע פעולות AND ניתן לנצל גם בדרכים אחרות. לדוגמה, יכולתו של CRCW PRAM לבצע AND בזמן $O(1)$ מבטלת את הצורך ברשת בקרה נפרדת שתבדוק אם כל המעבדים סיימו את האיטרציות של לולאה מסוימת, כפי שהנחנו עבור אלגוריתמי EREW. ההחלטה לסיים את הלולאה היא פשוט ה־AND של רצונם של כל המעבדים לסיים את הלולאה.

מודל EREW אינו מסוגל לבצע AND רב עוצמה כזה. כל אלגוריתם EREW המחשב את המקסימום מבין n איברים ידרוש זמן של $\Omega(\lg n)$. ההוכחה דומה בתפיסתה לטיעון עבור החסם התחתון למציאת שורשו של עץ בינרי. בהוכחה ההיא בדקנו כמה צמתים יכולים "לדעת" את זהותו של השורש שלהם, והראינו כי מספרם גדל פי שניים לכל היותר בכל צעד. עבור הבעיה של מציאת המקסימום מבין n איברים, אנו בוחנים כמה איברים "חושבים" שאולי הם המקסימום. אינטואיטיבית, בכל צעד של EREW PRAM, מספר זה יכול לקטון פי שניים לכל היותר, ודבר זה מוביל לחסם התחתון $\Omega(\lg n)$.

יש לציין שהחסם התחתון $\Omega(\lg n)$ עבור חישוב המקסימום מתקיים גם אם מרשים קריאות בו־זמניות; כלומר, הוא מתקיים גם עבור אלגוריתמי CREW. למעשה, Cook, Dwork & Reischuk [50] מראים כי כל אלגוריתם CREW למציאת המקסימום מבין n איברים חייב לרוץ בזמן $\Omega(\lg n)$, גם כאשר מספר המעבדים והמקום בזיכרון בלתי מוגבלים. חסם תחתון זה תקף גם עבור הבעיה של חישוב AND של n ערכים בוליאניים.

הדמיית אלגוריתם CRCW על־ידי אלגוריתם EREW

אנו יודעים עתה שאלגוריתמי CRCW יכולים לפתור בעיות מסוימות מהר יותר מאלגוריתמי EREW. יתר על כן, כל אלגוריתם EREW ניתן לביצוע על CRCW PRAM. לכן, עוצמתו של מודל CRCW גדולה ממש מעוצמתו של מודל EREW. אולם בכמה גדולה העוצמה? בסעיף 30.3 נראה כי EREW PRAM בן p מעבדים מסוגל למיין p מספרים בזמן $O(\lg p)$. עתה נשתמש בתוצאה זו למציאת חסם עליון תיאורטי על עוצמתו של CRCW PRAM בהשוואה ל־EREW PRAM.

משפט 30.1

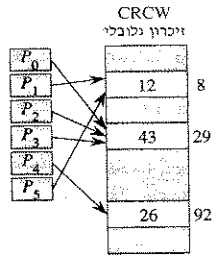
אלגוריתם CRCW על p מעבדים יכול להיות מהיר לכל היותר פי $O(\lg p)$ מאשר אלגוריתם EREW הטוב ביותר על p מעבדים עבור אותה בעיה.

הוכחה ההוכחה היא טיעון הדמיה. אנו מדמים כל צעד של אלגוריתם CRCW על־ידי חישוב EREW שרץ בזמן $O(\lg p)$. מכיוון ששתי המכונות הן בעלות אותו כוח עיבוד, די לנו להתמקד רק בגישות לזיכרון. אנו נסתפק כאן בהצגת ההוכחה עבור הדמיית פעולות כתיבה בו־זמניות, ונותיר לקורא, כתרגיל 30.2-8, את מימושן של פעולות קריאה בו־זמניות.

p המעבדים של EREW PRAM מדמים כתיבה בו־זמנית של אלגוריתם CRCW באמצעות מערך עזר A שאורכו p. איור 30.7 מדגים את הרעיון. כאשר המעבד P_i ב־CRCW, עבור $i = 0, 1, \ldots, p-1$, מבקש לכתוב נתון x_i במיקום l_i, כל מעבד מתאים P_i של EREW כותב במקום זאת את הזוג הסדור (l_i, x_i) למיקום $A[i]$. פעולות כתיבה אלה הן בלעדיות, משום שכל מעבד כותב במיקום שונה בזיכרון. לאחר מכן ממוין המערך A לפי הקואורדינטה הראשונה של כל זוג סדור בזמן $O(\lg p)$, וכתוצאה מכך, כל הנתונים שנכתבו לאותו מיקום מופיעים ביחד בפלט.

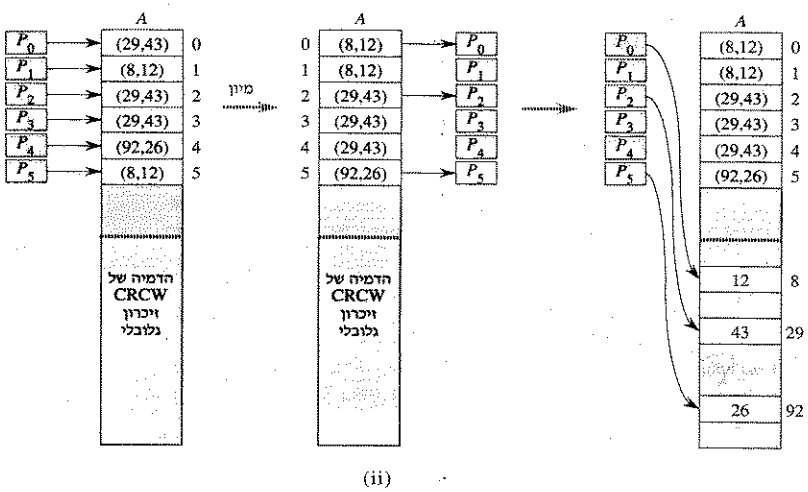
עתה, כל מעבד P_i של EREW, עבור $i = 1, \ldots, p-1$, בודק אם $A[i] = (l_j, x_j)$ ואם $A[i-1] = (l_k, x_k)$, כאשר j ו־k הם ערכים בתחום $0 \le j, k \le p-1$. אם $l_j \ne l_k$ או $i = 0$, אזי המעבד P_i, עבור $i = 0, 1, \ldots, p-1$, כותב את הנתון x_j למיקום l_j בזיכרון הגלובלי. אחרת, המעבד אינו עושה דבר. מאחר שהמערך A ממוין על פי הקואורדינטה הראשונה, רק אחד מן המעבדים הכותבים למיקום נתון כלשהו אכן מצליח לעשות זאת, ולכן הכתיבה היא בלעדית. תהליך זה מממש אפוא בזמן $O(\lg p)$ כל צעד של כתיבה בו־זמנית של מודל CRCW־משותף. ■

ניתן לדמות גם מודלים אחרים של כתיבה בו־זמנית. (ראה תרגיל 30.2-9.)



(i)

איור 30.7 הדמיית כתיבה בו־זמנית על EREW PRAM. **(i)** צעד של אלגוריתם CRCW־משותף שבו 6 מעבדים כותבים בו זמנית לזיכרון הגלובלי. **(ii)** הדמיית צעד זה על־ידי EREW PRAM. תחילה, זוגות סדורים המכילים מיקום ונתון נכתבים במערך A. לאחר מכן המערך ממוין. על־ידי השוואת איברים סמוכים במערך, אנו מבטיחים שרק הראשונה מכל קבוצה של פעולות כתיבה זהות לזיכרון הגלובלי אכן תמומש. במקרה זה, המעבדים P_0, P_2 ו־P_5 הם המבצעים את הכתיבה.



(ii)

איור 30.7 (המשך)

אם כן, מתעוררת השאלה איזה מודל עדיף — CRCW או EREW — ואם CRCW, איזה מודל CRCW? המצדדים במודלים CRCW מצביעים על כך שקל יותר לתכנת במודלים אלה מאשר במודל EREW, ושהאלגוריתמים שלהם מהירים יותר. המבקרים טוענים שהחומרה המשמשת למימוש פעולות זיכרון בו־זמניות איטית יותר מהחומרה למימוש פעולות זיכרון בלעדיות, ולכן זמן הריצה הקצר יותר של אלגוריתמי CRCW אינו ממשי. בפועל, הם אומרים, לא ניתן למצוא את המקסימום מבין n ערכים בזמן $O(1)$.

אחרים אומרים שמודל PRAM — ויהא זה EREW או CRCW — הוא מודל שגוי מעיקרו. המעבדים חייבים להיות מחוברים ביניהם באמצעות רשת תקשורת, ורשת התקשורת חייבת להיות חלק מן המודל. המעבדים צריכים להיות מסוגלים לתקשר אך ורק עם שכניהם ברשת.

ברור למדי ששאלת המודל המקבילי "הנכון" לא תוכרע בנקל לטובת אחד מן המודלים. עם זאת, חשוב להכיר בכך שהמודלים, כשמם כן הם — מודלים. במצב נתון בעולם האמיתי, המודלים השונים מתאימים במידות שונות. מידת ההתאמה בין המודל לבין המצב ההנדסי היא הדרגה שבה הניתוחים האלגוריתמיים במסגרת המודל מנבאים תופעות במציאות. לכן חשוב לחקור את המודלים והאלגוריתמים המקביליים השונים, כדי שעם התפתחות תחום החישוב המקבילי תוכל להיווצר תמימות דעים מבוססת באשר לפרדיגמות החישוב המקבילי המתאימות ביותר למימוש.

תרגילים

30.2-1

נניח שאנו יודעים כי יער עצים בינריים מורכב מעץ אחד בלבד, בן n צמתים. הראה שעל פי הנחה זו, ניתן לגרום לכך שמימוש של CREW של FIND-ROOTS ירוץ בזמן $O(1)$, בלי תלות בעומק העץ. הראה שכל אלגוריתם EREW ירוץ בזמן $\Omega(\lg n)$.

30.2-2

כתוב אלגוריתם EREW עבור FIND-ROOTS שרץ בזמן $O(\lg n)$ על יער בן n צמתים.

30.2-3

כתוב אלגוריתם CRCW עבור n מעבדים אשר מסוגל לחשב בזמן $O(1)$ את ה־OR של n ערכים בוליאניים.

30.2-4

תאר אלגוריתם CRCW יעיל הכופל שתי מטריצות בוליאניות בגודל $n \times n$ תוך שימוש ב־n^3 מעבדים.

30.2-5

תאר אלגוריתם EREW שזמן ריצתו $O(\lg n)$ אשר כופל שתי מטריצות $n \times n$ של מספרים ממשיים תוך שימוש ב־n^3 מעבדים. האם קיים אלגוריתם CRCW־משותף מהיר יותר? האם קיים אלגוריתם מהיר יותר באחד מן המודלים CRCW החזקים יותר?

***30.2-6** *

הוכח שעבור קבוע $\epsilon > 0$ כלשהו, קיים אלגוריתם CRCW שזמן ריצתו $O(1)$ המשתמש ב־$O(n^{1+\epsilon})$ מעבדים למציאת האיבר המקסימלי במערך בן n איברים.

***30.2-7** *

הראה כיצד ניתן למזג שני מערכים ממויינים, כל אחד בן n מספרים, בזמן $O(1)$ באמצעות אלגוריתם CRCW־קדימות. תאר כיצד ניתן להשתמש באלגוריתם זה לביצוע מיון בזמן $O(\lg n)$. האם אלגוריתם המיון שלך יעיל בעבודה?

30.2-8

השלם את הוכחתו של משפט 30.1 על־ידי כך שתתאר כיצד ממומשות פעולות קריאה בו־זמניות ב־CRCW PRAM בן p מעבדים בזמן $O(\lg p)$ על EREW PRAM בן p מעבדים.

30.2-9

הראה כיצד ניתן לממש מכונת CRCW־צירוף בת p מעבדים על־ידי EREW PRAM בן p מעבדים, תוך ירידה של $O(\lg p)$ בלבד בביצועים. (**רמז**: השתמש בחישוב רישות מקבילי.)

30.3 משפט ברנט ויעילות־בעבודה

משפט ברנט (Brent) מראה כיצד ניתן לדמות ביעילות מעגל צירופי על־ידי PRAM. בעזרת משפט זה ניתן להתאים למודל PRAM רבות מן התוצאות שהתקבלו בפרק 28 עבור רשתות מיון ורבות מן התוצאות שהתקבלו בפרק 29 עבור מעגלים אריתמטיים. לקוראים שאינם בקיאים במעגלים צירופיים מומלץ לחזור ולעיין בסעיף 29.1.

מעגל צירופי (combinational circuit) הוא רשת חסרת מעגלים (כלומר, אין בה מסלולים מעגליים) של **רכיבים צירופיים** (combinational elements). כל רכיב צירופי הוא בעל קלט אחד או יותר, ובסעיף זה נניח שכל רכיב הוא בעל פלט אחד בדיוק. (על רכיבים צירופיים בעלי $k > 1$ פלטים ניתן להסתכל כעל k רכיבים נפרדים.) מספר הקלטים הוא **פרישת־הכניסה** (fan-in) של הרכיב, ומספר המקומות שאליהם מוזנים הפלטים שלו הוא **פרישת־היציאה** (fan-out). ככלל, נניח בסעיף זה שפרישת־הכניסה של כל רכיב צירופי במעגל חסומה (על־ידי $O(1)$). את פרישת־היציאה, לעומת זאת, אין אנו מגבילים.

הגודל (size) של מעגל צירופי הוא מספר הרכיבים הצירופיים שהוא מכיל. מספר הרכיבים הצירופיים על מסלול ארוך ביותר מקלט של המעגל לפלט של רכיב צירופי הוא **העומק** (depth) של הרכיב. **העומק** של המעגל כולו הוא העומק המקסימלי של רכיביו.

משפט 30.2 (משפט ברנט (Brent))
ניתן לדמות כל מעגל צירופי בעל גודל n ועומק d שפרישת־הכניסה שלו חסומה, על־ידי אלגוריתם CREW על p מעבדים שזמן ריצתו $O(n/p + d)$.

הוכחה מאחסנים את קלטי המעגל הצירופי בזיכרון הגלובלי של ה־PRAM ושומרים עבור כל רכיב צירופי במעגל מיקום בזיכרון לאחסון ערך הפלט של הרכיב כאשר הוא מחושב. אזי, ניתן לדמות רכיב צירופי נתון על־ידי מעבד PRAM יחיד בזמן $O(1)$ באופן הבא: המעבד פשוט קורא את ערכי הקלט לרכיב מן המיקומים בזיכרון שבהם שמורים קלטי המעגל או פלטי רכיבים המוזנים כקלט לרכיב זה, וכך הוא מדמה את התילים במעגל. לאחר מכן מחשב המעבד את הפונקציה שמבצע הרכיב הצירופי, וכותב את התוצאה במיקום המתאים בזיכרון. מכיוון שפרישת־הכניסה של כל רכיב במעגל חסומה, ניתן לחשב כל פונקציה בזמן $O(1)$.

משימתנו היא אפוא למצוא לוח זמנים עבור p המעבדים של ה־PRAM כך שהדמיית כל הרכיבים הצירופיים תתבצע בזמן $O(n/p + d)$. האילוץ העיקרי הוא שלא ניתן לדמות רכיב כל עוד לא הסתיים חישובם של כל פלטי הרכיבים המוזנים כקלטים לרכיב זה. בכל פעם שכמה רכיבים צירופיים שהדמייתם מתבצעת במקביל זקוקים לאותו ערך, משתמשים בפעולות קריאה בו־זמניות.

מכיוון שכל הרכיבים בעומק 1 תלויים אך ורק בקלטים למעגל, אלה הם הרכיבים היחידים שניתן לדמותם בשלב ההתחלתי. עם סיום הדמייתם, ניתן לדמות את כל הרכיבים בעומק 2, וכן הלאה, עד שמסתיימת הדמייתם של כל הרכיבים בעומק d. רעיון המפתח הוא שאם הסתיימה הדמייתם של כל הרכיבים בעומקים מ־1 עד i, ניתן לדמות במקביל כל תת־קבוצה של רכיבים בעומק $i + 1$, שכן החישובים שהם מבצעים הם בלתי־תלויים זה בזה.

האסטרטגיה שלנו לבניית לוח הזמנים היא אפוא פשוטה למדי. אנו מדמים את כל הרכיבים בעומק i לפני הדמיית הרכיבים בעומק $i + 1$. את הרכיבים בעומק נתון i אנו מדמים בקבוצות של p בכל פעם. באיור 30.8 מדגמת אסטרטגיה כזאת עבור $p = 2$.

הבה ננתח אסטרטגיית הדמיה זו. עבור $i = 1, 2, \ldots, d$, יהי n_i מספר רכיבי המעגל בעומק i. אזי:

$$\sum_{i=1}^{d} n_i = n$$

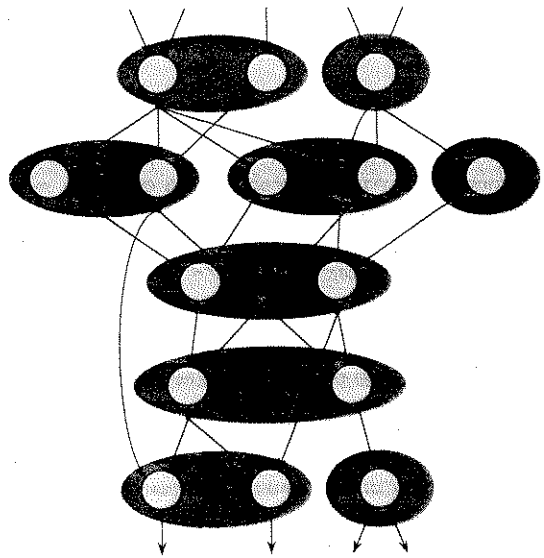
נתבונן ב־n_i הרכיבים הצירופיים בעומק i. על־ידי קיבוצם ל־$\lceil n_i/p \rceil$ קבוצות, כאשר כל אחת מ־$\lfloor n_i/p \rfloor$ הקבוצות הראשונות מכילה p רכיבים והקבוצה האחרונה מכילה את הרכיבים שנותרו,

אם יש כאלה, יכול ה־PRAM לדמות את החישובים המתבצעים על־ידי רכיבים צירופיים אלה בזמן $O(\lceil n_i/p \rceil)$. סדר הגודל של זמן ההדמיה הכולל הוא אפוא:

$$\sum_{i=1}^{d} \left\lceil \frac{n_i}{p} \right\rceil \le \sum_{i=1}^{d} \left(\frac{n_i}{p} + 1 \right)$$
$$= \frac{n}{p} + d$$

■

את משפט ברנט ניתן להרחיב גם להדמיות EREW של מעגלים צירופיים אשר רכיביהם הצירופיים הם בעלי פרישת־יציאה של $O(1)$.



איור 30.8 משפט ברנט (Brent). הדמייתו של מעגל צירופי בגודל 15 ובעומק 5 על־ידי CREW PRAM בעל 2 מעבדים ב־$5 + 15/2 \le 9$ צעדים. ההדמיה מתקדמת דרך המעגל מלמעלה למטה. ההצללות מצביעות על קבוצות של רכיבי מעגל שהדמייתם מתבצעת במקביל. המספר המופיע בכל קבוצה מציין את זמנו של הצעד שבו מתבצעת הדמיית הרכיבים השייכים לקבוצה.

מסקנה 30.3

כל מעגל צירופי בגודל n ובעומק d שפרישת־הכניסה ופרישת־היציאה שלו חסומות ניתן לדמות על EREW PRAM בעל p מעבדים בזמן $O(n/p + d)$.

הוכחה מבצעים הדמיה דומה לזו שהתבצעה בהוכחת משפט ברנט. ההבדל היחיד הוא בהדמיית התילים, שבה משפט 30.2 דורש פעולות קריאה בו־זמניות. עבור הדמיית ה־EREW, לאחר שחושב הפלט של רכיב צירופי, הוא אינו נקרא ישירות על־ידי המעבדים הזקוקים לערכו. במקום זאת, ערך הפלט מועתק על־ידי המעבד המדמה את הרכיב אל $O(1)$ הקלטים שבהם הוא דרוש. המעבדים הזקוקים לערך זה יכולים אז לקרוא אותו בלי להפריע זה לזה. ■

אסטרטגיית הדמיה זו של EREW אינה פועלת עבור רכיבים בעלי פרישת־יציאה בלתי חסומה, שכן אז העתקת ערכי הפלט עלולה לדרוש יותר מאשר זמן קבוע בכל צעד. לכן, עבור מעגלים המכילים רכיבים בעלי פרישת־יציאה בלתי חסומה, אנו זקוקים לכוח החישוב של קריאות בו־זמניות. (במקרה של פרישת־כניסה בלתי חסומה ניתן לטפל לעתים על־ידי הדמיית CRCW, אם הרכיבים הצירופיים פשוטים דיים. ראה תרגיל 30.3-1.)

מסקנה 30.3 מספקת לנו אלגוריתם EREW מהיר למיון. כפי שהוסבר בהערות הביבליוגרפיות לפרק 28, רשת המיון AKS שעומקה $O(\lg n)$ מסוגלת למיין n מספרים באמצעות $O(n \lg n)$ משווים. מאחר שמשווים הם בעלי פרישת־כניסה חסומה, קיים אלגוריתם EREW למיון n מספרים בזמן $O(\lg n)$ באמצעות n מעבדים. (השתמשנו בתוצאה זו בהוכחת משפט 30.1 כדי להראות ש־EREW PRAM מסוגל לדמות CRCW PRAM במחיר האטה שהיא לכל היותר לוגריתמית.) לרוע המזל, הקבועים החבויים בסימון O הם כה גדולים, עד שהעניין באלגוריתם מיון זה הוא תיאורטי בלבד. עם זאת, קיימים אלגוריתמי EREW מעשיים יותר למיון, והראוי לציון מביניהם הוא אלגוריתם המיון־מיזוג המקבילי של Cole [46].

עתה נניח שנתון לנו אלגוריתם PRAM המשתמש ב־p מעבדים לכל היותר, אולם ברשותנו PRAM בעל $p' < p$ מעבדים בלבד. היינו רוצים להיות מסוגלים להריץ את האלגוריתם עבור p המעבדים על ה־PRAM הקטן יותר, בן p' המעבדים, כך שהביצוע יהיה יעיל־בעבודה. תוך שימוש ברעיון שעליו מבוססת הוכחת משפט ברנט, ניתן לנסח תנאי המאפשר זאת.

משפט 30.4

אם אלגוריתם A על PRAM בן p מעבדים רץ בזמן t, אזי עבור כל $p' < p$, קיים עבור אותה בעיה אלגוריתם A' על PRAM בן p' מעבדים שרץ בזמן $O(pt/p')$.

הוכחה נמספר על־ידי $1, 2, \ldots, t$ את הצעדים של האלגוריתם A (כל צעד נמשך יחידת זמן אחת). האלגוריתם A' מדמה את ביצועו של כל צעד i, עבור $i = 1, 2, \ldots, t$, בזמן $O(\lceil p/p' \rceil)$. ישנם t צעדים, ולכן ההדמיה כולה מתבצעת בזמן $O(\lceil p/p' \rceil t) = O(pt/p')$, שכן $p' < p$. ■

העבודה המתבצעת על־ידי האלגוריתם A היא pt, והעבודה המתבצעת על־ידי האלגוריתם A' היא $p'(pt/p') = pt$; ההדמיה היא אפוא יעילה בעבודה. לכן, אם האלגוריתם A עצמו יעיל בעבודה, כזה הוא גם האלגוריתם A'.

לפיכך, כאשר מפתחים אלגוריתמים יעילים בעבודה עבור בעיה מסוימת, אין בהכרח צורך ביצירת אלגוריתם שונה עבור כל מספר שונה של מעבדים. לדוגמה, נניח שניתן להוכיח את קיומו של חסם תחתון הדוק t על זמן הריצה של אלגוריתם מקבילי כלשהו, בלי תלות במספר המעבדים, לפתרון בעיה מסוימת, ונניח עוד כי האלגוריתם הסדירתי הטוב ביותר עבור בעיה זו מבצע עבודה w. אזי, כל שעלינו לעשות הוא לפתח עבור הבעיה אלגוריתם יעיל בעבודה המשתמש ב־$p = \Theta(w/t)$ מעבדים, כדי לקבל אלגוריתמים יעילים בעבודה עבור כל מספר מעבדים המאפשר אלגוריתם יעיל בעבודה. עבור $p' = o(p)$, משפט 30.4 מבטיח שאכן קיים אלגוריתם יעיל בעבודה. עבור $p' = \omega(p)$, לא קיים אלגוריתם יעיל בעבודה, שכן אם t הוא חסם תחתון על זמן הריצה של אלגוריתם מקבילי כלשהו, אזי $p't = \omega(pt) = \omega(w)$.

תרגילים

30.3-1

הוכח תוצאה אנלוגית למשפט ברנט עבור הדמיית CRCW של מעגלים צירופיים בוליאניים המכילים שערי AND ו־OR בעלי פרישת־כניסה בלתי חסומה. (**רמז:** הגדר "גודל" כמספר הכולל של הקלטים לשערים במעגל.)

30.3-2

הראה שניתן לממש חישוב רישות מקבילי על n ערכים המאוחסנים במערך בזיכרון בזמן $O(\lg n)$ על EREW PRAM המשתמש ב־$O(n/\lg n)$ מעבדים. מדוע לא ניתן להרחיב את התוצאה ישירות לרשימה של n ערכים?

30.3-3

הראה כיצד ניתן לכפול מטריצה A שממדיה $n \times n$ בווקטור b שאורכו n בזמן $O(\lg n)$, באמצעות אלגוריתם EREW יעיל בעבודה. (**רמז:** בנה מעגל צירופי עבור הבעיה.)

30.3-4

כתוב אלגוריתם CRCW המשתמש ב־n^2 מעבדים כדי לכפול שתי מטריצות של $n \times n$. האלגוריתם צריך להיות יעיל בעבודה בהשוואה לאלגוריתם הסדרתי הרגיל לכפל מטריצות שרץ בזמן $O(n^3)$.

30.3-5

קיימים מודלים מקביליים המאפשרים למעבדים להיעשות לא־פעילים כך שמספר המעבדים המבצעים כל צעד משתנה. נגדיר את העבודה במודל זה כמספר הכולל של צעדים המבוצעים במהלך ריצתו של אלגוריתם על־ידי מעבדים פעילים. הראה שכל אלגוריתם CRCW שמבצע עבודה w ורץ בזמן t, יכול לרוץ על EREW PRAM בן p מעבדים בזמן $O((w/p + t)\lg p)$. (**רמז:** החלק הקשה הוא תזמון פעולתם של המעבדים הפעילים כל עוד נמשך החישוב.)

* 30.4 חישוב רֵישות מקבילי יעיל בעבודה

בסעיף 30.1.2 בחנו את האלגוריתם List-Rank, אלגוריתם EREW שרץ בזמן $O(\lg n)$ ויכול לבצע חישוב רֵישות על רשימה מקושרת בת n עצמים. האלגוריתם משתמש ב־n מעבדים ומבצע עבודה של $\Theta(n \lg n)$. מכיוון שניתן לבצע בנקל חישוב רֵישות על מכונה סדרתית בזמן $\Theta(n)$, List-Rank אינו יעיל בעבודה.

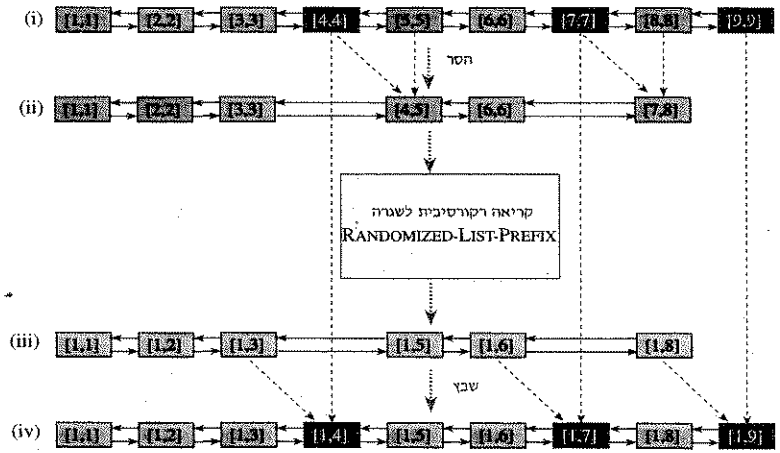
בסעיף זה נציג אלגוריתם EREW אקראי יעיל־בעבודה לחישוב רישות מקבילי. האלגוריתם משתמש ב־$\Theta(n/\lg n)$ מעבדים וזמן הריצה שלו הוא $O(\lg n)$ בהסתברות גבוהה. לכן האלגוריתם יעיל בעבודה בהסתברות גבוהה. יתר על כן, על־פי משפט 30.4, אלגוריתם זה מניב מיד אלגוריתמים יעילים בעבודה עבור כל מספר $p = O(n/\lg n)$ של מעבדים.

חישוב רישות מקבילי רקורסיבי

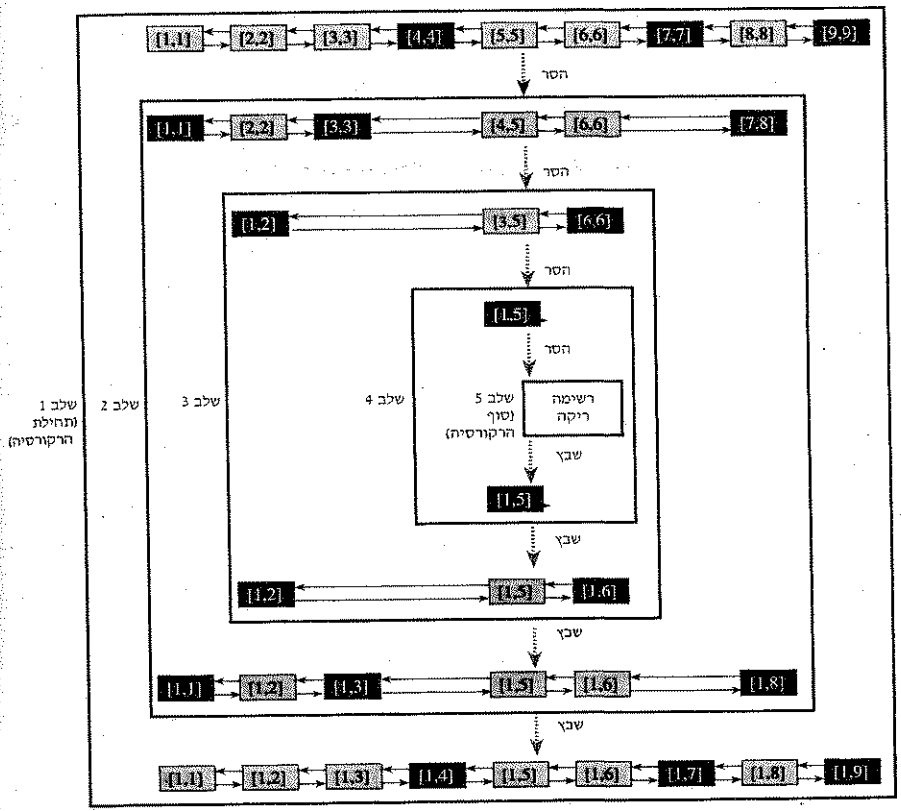
האלגוריתם המקבילי האקראי Randomized-List-Prefix מבצע חישוב רישות מקבילי על רשימה מקושרת בת n עצמים באמצעות $p = \Theta(n/\lg n)$ מעבדים. במהלך ריצתו של האלגוריתם, כל מעבד אחראי על $n/p = \Theta(\lg n)$ מן העצמים שברשימה המקורית. חלוקת העצמים בין המעבדים השונים היא שרירותית (העצמים שעליהם אחראי מעבד מסוים אינם בהכרח עוקבים). החלוקה מתבצעת לפני תחילת הרקורסיה וה"בעלות" על עצמים אינה משתנה במהלך האלגוריתם. מטעמי נוחות, אנו מניחים שהרשימה היא דו־מקושרת, שכן הפיכת רשימה חד־מקושרת לדו־מקושרת אורכת זמן של $O(1)$.

הרעיון של Randomized-List-Prefix הוא להסיר חלק מהעצמים מן הרשימה, לבצע חישוב רישות רקורסיבי על הרשימה המתקבלת, ואז להרחיב אותה על־ידי כך שמשבצים בחזרה את העצמים שהוסרו, כך שיתקבל חישוב רישות על הרשימה המקורית. איור 30.9 מדגים את התהליך הרקורסיבי, ואיור 30.10 מראה כיצד נפרשת הרקורסיה. אנו נראה בהמשך שכל שלב ברקורסיה מקיים שתי תכונות:

1. עצם אחד לכל היותר נבחר להסרה מבין העצמים השייכים למעבד נתון.
2. אף פעם לא נבחרים להסרה שני עצמים סמוכים.



איור 30.9 האלגוריתם המקבילי, הרקורסיבי, האקראי, היעיל בעבודה Randomized-List-Prefix עבור ביצוע חישוב רישות ברשימה מקושרת בת $n = 9$ עצמים. **(i)-(ii)** קבוצה של עצמים שאינם סמוכים זה לזה (מופיעים בשחור) נבחרים להסרה מן הרשימה. הערך בכל עצם שחור משמש לעדכון הערך בעצם העוקב לו ברשימה ואז העצם השחור מוסר מן הרשימה. האלגוריתם נקרא כדי לבצע חישוב רישות מקבילי על הרשימה המכווצת. **(iv)-(iii)** הערכים המתקבלים הם הערכים הסופיים הנכונים עבור העצמים ברשימה המכווצת. העצמים שהוסרו משובצים מחדש ברשימה, וכל אחד מהם משתמש בערך של העצם הקודם לו לחישוב ערכו הסופי.



איור 30.10 שלבי הרקורסיה של RANDOMIZED-LIST-PREFIX מוצגים כאן עבור רשימה מקורית בת $n = 9$ עצמים. בכל שלב, מסירים מן הרשימה את העצמים השחורים. הרקורסיה נמשכת עד שהרשימה ריקה, ואז העצמים שהוסרו משובצים מחדש ברשימה.

לפני שנראה כיצד יש לבחור עצמים המקיימים תכונות אלה, הבה נבחן ביתר פירוט כיצד מתבצע חישוב הרישות. נניח שבצעד הראשון של הרקורסיה העצם ה־k ברשימה נבחר לצורך הסרה. עצם זה מכיל את הערך $[k, k]$ הנשלף על־ידי העצם ה־$(k+1)$ ברשימה. (מקרי גבול, כמו המקרה כאן שבו k נמצא בסוף הרשימה, ניתנים לטיפול ישיר ולכן אינם מתוארים כאן.) העצם ה־$(k+1)$, המכיל את הערך $[k+1, k+1]$, מחשב אז את $[k, k+1] = [k, k] \otimes [k+1, k+1]$ ומאחסן אותו. לאחר מכן מוסר העצם ה־k מן הרשימה על־ידי שינוי מצביעים כך שהרשימה "תדלג" עליו.

לאחר מכן השגרה RANDOMIZED-LIST-PREFIX קוראת לעצמה באופן רקורסיבי לביצוע חישוב רישות על הרשימה ה"מכווצת". (הירידה הרקורסיבית מסתיימת כאשר הרשימה כולה ריקה.) הבחנת המפתח כאן היא, שלאחר החזרה מן הקריאה הרקורסיבית, כל עצם ברשימה המכווצת מכיל את הערך הסופי הנכון אשר לו הוא זקוק לצורך חישוב הרישות המקבילי על הרשימה המקורית. לא נותר אלא לשבץ בחזרה את העצמים שהוסרו, כמו העצם ה־k, ולעדכן את ערכיהם.

לאחר שהעצם ה־k משובץ בחזרה ברשימה, ניתן לחשב את ערך הרישא הסופי שלו באמצעות הערך המאוחסן בעצם ה־$(k-1)$. לאחר החזרה מהרקורסיה, העצם ה־$(k-1)$ מכיל את $[1, k-1]$, ולכן העצם ה־k — שעדיין מכיל את הערך $[k, k]$ — צריך רק לשלוף את הערך $[1, k-1]$ ולחשב את $[1, k] = [1, k-1] \otimes [k, k]$.

עקב תכונה 1, כל אחד מן העצמים שנבחרים לצורך הסרה שייך למעבד שונה המבצע את העבודה הדרושה להסרתו של העצם ולשיבוצו מחדש ברשימה. תכונה 2 מבטיחה שלא ייווצר שום בלבול בין המעבדים בעת הסרתם של עצמים ושיבוצם מחדש (ראה תרגיל 30.4-1). שתי התכונות יחד מבטיחות שכל צעד של הרקורסיה ניתן למימוש בזמן $O(1)$ על מכונת EREW.

בחירת איברים להסרה מן הרשימה

כיצד בוחרת RANDOMIZED-LIST-PREFIX את העצמים שיוסרו מן הרשימה? היא חייבת לציית לשתי התכונות שלעיל, ונוסף על כך, אנו רוצים שהזמן לבחירת העצמים יהיה קצר (ועדיף שיהיה קבוע). יתר על כן, היינו רוצים שייבחרו עצמים רבים ככל האפשר.

שיטת הבחירה האקראית שנציג להלן מקיימת תנאים אלה. עצמים נבחרים על־ידי כך שכל מעבד מבצע את הצעדים הבאים:

1. המעבד בוחר עצם i שלא נבחר קודם לכן מבין העצמים שבבעלותו.
2. לאחר מכן הוא "מטיל מטבע", ובוחר אחד מהערכים "עץ" או "פלי" בהסתברות שווה.
3. אם המעבד בוחר "עץ", הוא מסמן את העצם i כעצם שנבחר, אלא אם כן $next[i]$ נבחר על־ידי מעבד אחר שגם המטבע שלו הראה "עץ".

בשיטה אקראית זו ניתן לבחור עצמים להסרה בזמן $O(1)$, והיא אינה דורשת גישות בו־זמניות לזיכרון.

עלינו להראות שתהליך זה מציית לשתי התכונות שתוארו לעיל. ניתן לראות בקלות שתכונה 1 מתקיימת, שכן כל מעבד בוחר עצם אחד בלבד כמועמד להסרה. כדי לראות שתכונה 2 מתקיימת אף היא, נניח בשלילה שנבחרו שני עצמים עוקבים, i ו־$next[i]$. דבר כזה מתרחש אם שני העצמים נבחרו על־ידי המעבדים שלהם, ושני המעבדים הטילו "עץ". אולם העצם i אינו נבחר אם המעבד האחראי על $next[i]$ הטיל "עץ", וקיבלנו סתירה.

ניתוח

מכיוון שכל צעד רקורסיבי של RANDOMIZED-LIST-PREFIX מתבצע בזמן $O(1)$, הרי שכדי לנתח את האלגוריתם איננו צריכים אלא לקבוע כמה צעדים דרושים כדי להסיר את כל העצמים מן הרשימה המקורית. בכל צעד, ההסתברות שמעבד יסיר את העצם i שבחר היא לפחות $1/4$. מדועי ההסתברות לקבלת "עץ" בהטלת המטבע היא $1/2$, וההסתברות לכך שמעבד לא יבחר את $next[i]$ או יבחר אותו ויטיל "פלי" היא לפחות $1/2$. מאחר ששתי הטלות המטבע הן מאורעות בלתי־תלויים, נוכל לכפול את הסתברויותיהם ולקבל שההסתברות שמעבד יסיר מן הרשימה את העצם שבחר היא לפחות $1/4$. מכיוון שכל מעבד אחראי על $\Theta(\lg n)$ עצמים, תוחלת הזמן הנדרש למעבד להסיר את כל העצמים שלו היא $\Theta(\lg n)$.

לרוע המזל, ניתוח פשוט זה אינו מראה שתוחלת זמן הריצה של RANDOMIZED-LIST-PREFIX היא $\Theta(\lg n)$. לדוגמה, אם רוב המעבדים מסירים את כל העצמים שלהם במהירות ומספר קטן של

מעבדים זקוקים לזמן ארוך בהרבה, הזמן הממוצע שיידרש למעבד להסיר את כל העצמים שלו עדיין עשוי להיות $\Theta(\lg n)$, אבל זמן הריצה של האלגוריתם עשוי להיות גדול.

תוחלת זמן הריצה של השגרה RANDOMIZED-LIST-PREFIX היא אכן $\Theta(\lg n)$, למרות שהניתוח הפשוט שהבאנו אינו מראה זאת. אנו נשתמש בטיעון על הסתברויות גבוהות, כדי להוכיח שבהסתברות של $1 - 1/n$ לפחות, כל העצמים מוסרים ב־$c\lg n$ צעדים של הרקורסיה, עבור קבוע c מסוים. בתרגילים 30.4-4 ו־30.4-5 תתבקש להכליל טיעון זה להוכחת החסם $\Theta(\lg n)$ על תוחלת זמן הריצה.

טיעון ההסתברות הגבוהה שלנו מבוסס על ההבחנה שאת הניסוי שעורך מעבד נתון להסרת העצמים שהוא בוחר אפשר לראות כסדרה של ניסויי ברנולי (ראה פרק 6). הניסוי מסתיים בהצלחה אם העצם נבחר להסרה, ואחרת, הוא מסתיים בכישלון. מכיוון שברצוננו להראות שקיימת הסתברות קטנה בלבד למספר מועט מאוד של הצלחות, נוכל להניח שההסתברות להצלחה היא בדיוק $1/4$ ולא לפחות $1/4$. (ראה הצדקה פורמלית להנחות דומות בתרגילים 6.4-8 ו־6.4-9.)

כדי לפשט עוד יותר את הניתוח, נניח שקיימים בדיוק $n/\lg n$ מעבדים, שכל אחד מהם אחראי על $\lg n$ עצמים מן הרשימה. אנו ערוכים $c\lg n$ ניסויים, עבור קבוע c מסוים שייקבע על ידינו, ואנו מעוניינים במאורע שבו מתרחשות פחות מ־$\lg n$ הצלחות. יהי X המשתנה המקרי המציין את מספר ההצלחות הכולל. על פי מסקנה 6.3, ההסתברות שמעבד יסיר פחות מ־$\lg n$ עצמים במהלך $c\lg n$ ניסויים היא לכל היותר:

$$\begin{aligned}
\Pr\{X < \lg n\} &\le \binom{c\lg n}{\lg n}\left(\frac{3}{4}\right)^{c\lg n - \lg n} \\
&\le \left(\frac{ec\lg n}{\lg n}\right)^{\lg n}\left(\frac{3}{4}\right)^{(c-1)\lg n} \\
&= \left(ec\left(\frac{3}{4}\right)^{c-1}\right)^{\lg n} \\
&\le \left(\frac{1}{4}\right)^{\lg n} \\
&= 1/n^2
\end{aligned}$$

כל עוד $c \ge 20$. (השורה השנייה נובעת מאי־שוויון (6.9).) לכן, ההסתברות שלאחר $c\lg n$ צעדים עדיין לא הוסרו כל העצמים השייכים למעבד נתון היא לכל היותר $1/n^2$.

עתה ברצוננו לחסום את ההסתברות שלאחר $c\lg n$ צעדים עדיין לא הוסרו כל העצמים השייכים לכל המעבדים. על פי אי־שוויון בול (6.22), הסתברות זו היא לכל היותר סכום ההסתברויות שכל אחד מן המעבדים עדיין לא הסיר את העצמים שלו. מכיוון שקיימים $n/\lg n$ מעבדים, ולכל אחד מהם קיימת הסתברות של $1/n^2$ לכל היותר שעדיין לא הסיר את כל העצמים שלו, ההסתברות שמעבד כלשהו לא הסיר את כל עצמיו היא לכל היותר:

$$\frac{n}{\lg n}\cdot\frac{1}{n^2} \le \frac{1}{n}$$

הוכחנו אפוא שבהסתברות של $1 - 1/n$ לפחות, כל העצמים מוסרים מן הרשימה ב־$O(\lg n)$ קריאות רקורסיביות. מכיוון שכל קריאה רקורסיבית נמשכת זמן של $O(1)$, זמן הריצה של RANDOMIZED-LIST-PREFIX הוא $O(\lg n)$ בהסתברות גבוהה.

הקבוע $c \ge 20$ בזמן הריצה $c\lg n$ עלול להיראות גדול מעט עבור יישום מעשי. למעשה, קבוע זה הוא יותר תוצר של הניתוח ולאו דווקא גורם המשקף את ביצועי האלגוריתם; באופן מעשי, האלגוריתם נוטה להיות מהיר. הגורמים הקבועים בניתוח הם גדולים משום שהמאורע שמעבד אחד מסיים להסיר מהרשימה את כל העצמים שלו תלוי במאורע שמעבד אחר מסיים את כל עבודתו שלו. בשל תנאי תלות אלה, השתמשנו באי־שוויון בול, שאינו דורש אי־תלות, אולם תוצאתו היא קבוע חלש יותר מזה שנתקלים בו בדרך כלל הלכה למעשה.

תרגילים

30.4-1
הדגם באמצעות איורים את השיבושים העלולים להיגרם ב־RANDOMIZED-LIST-PREFIX אם נבחרים להסרה שני עצמים עוקבים ברשימה.

*** 30.4-2**
הצע שינוי פשוט שיגרום לכך שהשגרה RANDOMIZED-LIST-PREFIX תרוץ בזמן $O(n)$ במקרה הגרוע על רשימה בת n עצמים. השתמש בהגדרת התוחלת כדי להוכיח שעם שינוי זה, תוחלת זמן הריצה של האלגוריתם היא $O(\lg n)$.

*** 30.4-3**
הראה כיצד ניתן לממש את RANDOMIZED-LIST-PREFIX כך שכל מעבד ישתמש במקרה הגרוע במקום בזיכרון של $O(n/p)$ לכל היותר, ללא תלות בעומק הרקורסיה.

*** 30.4-4**
הראה שעבור כל קבוע $k \ge 1$, RANDOMIZED-LIST-PREFIX רצה בזמן $O(\lg n)$ בהסתברות של $1 - 1/n^k$ לפחות. הראה כיצד משפיע k על הקבוע המופיע בחסם על זמן הריצה.

*** 30.4-5**
על סמך תוצאת תרגיל 30.4-4, הראה שתוחלת זמן הריצה של RANDOMIZED-LIST-PREFIX היא $O(\lg n)$.

30.5 שבירת סימטריה באופן דטרמיניסטי

נחשוב על מצב שבו שני מעבדים מעוניינים להשיג באותו זמן רשות גישה בלעדית לעצם. כיצד יכולים המעבדים לקבוע איזה מהם יהיה הראשון שיקבל רשות גישה? ברצוננו למנוע את התרחיש שבו שניהם מקבלים רשות גישה וגם את התרחיש שבו אף אחד מהם אינו מקבל רשות גישה. בעיית הבחירה של אחד המעבדים היא דוגמה ל**שבירת סימטריה** (symmetry breaking). כולנו מכירים את הבלבול הרגעי ואת הקיפאון הדיפלומטי שאליהם נקלעים שני אנשים המנסים לעבור בו־בזמן באותה דלת. בעיות שבירת סימטריה דומות מתעוררות לעתים קרובות בתכנון אלגוריתמים מקביליים, ופתרונות יעילים יש בהם תועלת רבה.

אחת הדרכים לשבירת סימטריה היא להטיל מטבע. במחשב, הטלת מטבע ניתנת למימוש באמצעות מחולל מספרים אקראיים. בדוגמת שני המעבדים, שניהם יכולים להטיל מטבע. אם האחד מקבל "עץ" והאחר "פלי", זה שהטיל "עץ" ממשיך. אם שניהם מטילים אותו ערך, הם מנסים שוב. (ראה תרגיל 30.5-1).

את האפקטיביות של אסטרטגיה אקראית ראינו בסעיף 30.4. בשגרה RANDOMIZED-LIST-PREFIX אסור שייבחרו להסרה שני עצמים סמוכים ברשימה, אבל ברצוננו לבחור עצמים רבים ככל האפשר. אך בתוך רשימת העצמים שנבחרו, כל העצמים נראים פחות או יותר אותו דבר. כפי שראינו, אסטרטגיה אקראית סיפקה דרך פשוטה ויעילה לשבירת הסימטריה בין עצמים סמוכים ברשימה, ועם זאת הבטיחה, בהסתברות גבוהה, שייבחרו עצמים רבים.

בסעיף זה נחקור שיטה דטרמיניסטית לשבירת סימטריה. המפתח לאלגוריתם הוא להשתמש באינדקסים של המעבדים או בכתובות בזיכרון ולאו דווקא בהטלת מטבע אקראיות. בדוגמת שני המעבדים, למשל, נוכל לשבור את הסימטריה על־ידי כך שנאפשר למעבד בעל האינדקס הקטן יותר להתקדם ראשון — תהליך שבבירור מתבצע בזמן קבוע.

אנו נשתמש באותו רעיון, אולם בדרך חכמה בהרבה, באלגוריתם לשבירת הסימטריה ברשימה מקושרת בת n עצמים. מטרתנו היא לבחור חלק קבוע מן העצמים ברשימה אולם להימנע מבחירת עצמים סמוכים. אלגוריתם זה ניתן לביצוע באמצעות n מעבדים בזמן $O(\lg^* n)$ על־ידי אלגוריתם EREW דטרמיניסטי. מכיוון ש־$\lg^* n \le 5$ עבור כל $n \le 2^{65536}$, הרי שלכל מטרה מעשית ניתן לראות את $\lg^* n$ כבעל ערך קבוע קטן (ראה סעיף 2.2, פונקציית הלוגריתם החוזר).

באלגוריתם הדטרמיניסטי שלנו שני חלקים. החלק הראשון מחשב "צביעת־6" של הרשימה המקושרת בזמן $O(\lg^* n)$. החלק השני ממיר צביעת־6 זו ל"קבוצה בלתי־תלויה מקסימלית" של הרשימה בזמן $O(1)$. קבוצה בלתי־תלויה מקסימלית זו תכיל חלק קבוע מ־n העצמים שברשימה, אבל לא יהיו בה אף פעם שני עצמים סמוכים.

צביעות וקבוצות בלתי־תלויות מקסימלית

צביעה (coloring) של גרף בלתי־מכוון $G = (V, E)$ היא פונקציה $C : V \to N$ כך שעבור כל $u, v \in V$, אם $C(u) = C(v)$, אזי $(u, v) \notin E$; כלומר, קדקודים סמוכים לעולם אינם צבועים באותו צבע. בצביעת־6 של רשימה מקושרת, כל הצבעים שייכים לתחום $\{0, 1, 2, 3, 4, 5\}$, ולא קיימים שני קדקודים עוקבים הצבועים באותו צבע. למעשה, עבור כל רשימה מקושרת קיימת צביעת־2, שכן ניתן לצבוע את כל העצמים שמיקומם אי־זוגי בצבע 0, ואת כל העצמים שמיקומם זוגי בצבע 1. ניתן לחשב צביעה כזאת בזמן $O(\lg n)$ באמצעות חישוב רישות מקבילי, אולם עבור יישומים רבים די לחשב רק צביעת־$O(1)$. אנו נראה שניתן לחשב צביעת־6 בזמן $O(\lg^* n)$, ללא שימוש באקראיות.

קבוצה בלתי־תלויה (independent set) בגרף $G = (V, E)$ היא תת־קבוצה $V' \subseteq V$ של קדקודים כך שכל קשת ב־E קשורה לכל היותר לקדקוד אחד ב־V'. **קבוצה בלתי־תלויה מקסימלית** (maximal independent set, או MIS) היא קבוצה בלתי־תלויה V' כך שעבור כל הקדקודים $v \in V - V'$, הקבוצה $V' \cup \{v\}$ אינה בלתי־תלויה — כל קדקוד שאינו ב־V' סמוך לאיזשהו קדקוד ב־V'. אין לבלבל בין בעיית החישוב של קבוצה בלתי־תלויה *גדולה מקסימלי* — בעיה קשה — לבין בעיית החישוב של קבוצה בלתי־תלויה *מקסימלי* — בעיה קשה. הבעיה של מציאת קבוצה בלתי־תלויה שגודלה מקסימלי בגרף כללי היא NP־שלמה. (ראה דיון בבעיות NP־שלמות בפרק 36. בעיה 36-1 עוסקת בקבוצות בלתי תלויות שגודלן מקסימלי.)

עבור רשימות בנות n עצמים, ניתן למצוא קבוצה בלתי־תלויה בגודל מקסימלי (ולכן מקסימלית) בזמן $O(\lg n)$ על־ידי שימוש בחישוב רישות מקבילי, כמו בצביעת־2 שהוזכרה לעיל, לזיהוי העצמים שמיקומם אי־זוגי. בשיטה זו נבחרים $\lceil n/2 \rceil$ עצמים. עם זאת, יש לשים לב לכך שכל קבוצה בלתי־תלויה מקסימלית ברשימה מקושרת מכילה לפחות $n/3$ עצמים, שכן מבין כל 3 עצמים עוקבים, אחד לפחות חייב להשתייך לקבוצה. אולם אנו נראה שניתן למצוא קבוצה בלתי־תלויה מקסימלית ברשימה בזמן $O(1)$, אם נתונה צביעת־$O(1)$ של הרשימה.

חישוב צביעת־6

האלגוריתם SIX-COLORS מחשב צביעת־6 של רשימה. לא נביא כאן פסאודו־קוד עבור אלגוריתם זה, אולם נתאר אותו בפירוט מסוים. אנו מניחים שבמצב ההתחלתי כל עצם x ברשימה המקושרת קשור למעבד $P(x) \in \{0, 1, \ldots, n-1\}$, כאשר כל עצם קשור למעבד שונה.

הרעיון של SIX-COLORS הוא לחשב סדרה $C_0[x], C_1[x], \ldots, C_m[x]$ של צבעים עבור כל עצם x ברשימה. הצביעה ההתחלתית C_0 היא צביעת־n. כל איטרציה של האלגוריתם מגדירה צביעה חדשה C_{k+1} המבוססת על הצביעה הקודמת C_k, עבור $k = 0, 1, \ldots, m-1$. הצביעה הסופית C_m היא צביעת־6, ואנו נוכיח כי $m = O(\lg^* n)$.

הצביעה ההתחלתית היא צביעת־n הטריוויאלית, שבה $C_0[x] = P(x)$. מכיוון שאין ברשימה שני עצמים בעלי אותו צבע, אין ברשימה שני עצמים סמוכים בעלי אותו צבע, ולכן הצביעה היא חוקית. נשים לב כי כל אחד מ־n הצבעים ההתחלתיים ניתן לתיאור על־ידי $\lceil \lg n \rceil$ סיביות, ופירוש הדבר שניתן לאחסן אותו במילת מחשב רגילה.

הצביעות הבאות מתקבלות כך: האיטרציה ה־k, עבור $k = 0, 1, \ldots, m-1$, מתחילה עם צביעה C_k ומסתיימת עם צביעה C_{k+1} המשתמשת במספר קטן יותר של סיביות עבור כל עצם, כפי שניתן לראות בחלק הראשון של איור 30.11. נניח שבתחילתה של איטרציה, הצבע C_k של כל עצם מיוצג על־ידי r סיביות. את צבעו החדש של עצם x קובעים על־ידי התבוננות בהמשך הרשימה, בצבעו של $next[x]$.

ביתר דיוק, נניח שעבור כל עצם x קיבלנו $C_k[x] = a$ ו־$C_k[next[x]] = b$, כאשר $a = \langle a_{r-1}, a_{r-2}, \ldots, a_0 \rangle$ ו־$b = \langle b_{r-1}, b_{r-2}, \ldots, b_0 \rangle$ הם צבעים בני r סיביות. מכיוון ש־$C_k[x] \ne C_k[next[x]]$, קיים אינדקס קטן ביותר i שבו הסיביות של שני הצבעים שונות זו מזו: $a_i \ne b_i$. מכיוון ש־$0 \le i \le r-1$, אפשר לכתוב את i באמצעות $\lceil \lg r \rceil$ סיביות בלבד: $i = \langle i_{\lceil \lg r \rceil - 1}, i_{\lceil \lg r \rceil - 2}, \ldots, i_0 \rangle$. אנו צובעים מחדש את x בערך המתקבל מהשרשור של i עם הסיבית a_i. כלומר, אנו קובעים:

$$\begin{aligned} C_{k+1}[x] &= \langle i, a_i \rangle \\ &= \langle i_{\lceil \lg r \rceil - 1}, i_{\lceil \lg r \rceil - 2}, \ldots, i_0, a_i \rangle \end{aligned}$$

זוג הרשימה מקבל את הצבע החדש $\langle 0, a_0 \rangle$. מספר הסיביות בייצוגו של כל צבע חדש הוא אפוא $\lceil \lg r \rceil + 1$ לכל היותר.

עלינו להראות שאם כל איטרציה של SIX-COLORS מתחילה בצביעה, אזי "הצביעה" החדשה שהיא יוצרת היא אכן צביעה חוקית. לשם כך, נוכיח כי $C_k[x] \ne C_k[next[x]]$ גורר $C_{k+1}[x] \ne C_{k+1}[next[x]]$. נניח כי $C_k[x] = a$ ו־$C_k[next[x]] = b$, וכי $C_{k+1}[x] = \langle i, a_i \rangle$ ו־$C_{k+1}[next[x]] = \langle j, b_j \rangle$. יש לבחון שני מקרים. אם $i \ne j$, אזי

$\langle i, a_i \rangle \neq \langle j, b_j \rangle$, ולכן הצבעים החדשים שונים זה מזה. אבל אם $i = j$, אזי על פי שיטת הצביעה מחדש שלנו, $a_i \neq b_i = b_j$, ולכן גם הפעם הצבעים החדשים שונים זה מזה. (המצב בזנב הרשימה ניתן לטיפול באופן דומה.)

שיטת הצביעה מחדש שבה משתמש האלגוריתם Six-Colors לוקחת צבע בן r סיביות ומחליפה אותה בצבע בן $(1 + \lceil \lg r \rceil)$ סיביות, ופירוש הדבר שמספר הסיביות קטן ממש כל עוד $r \geq 4$. כאשר $r = 3$, שני צבעים יכולים להיות שונים זה מזה במיקומי הסיביות 0, 1 או 2. כל צבע חדש הוא אפוא השרשור של $\langle 00 \rangle$, $\langle 01 \rangle$ או $\langle 10 \rangle$ עם 0 או עם 1, ושוב מתקבל מספר בן 3 סיביות. אולם, מבין 8 הערכים האפשריים של מספרים בני 3 סיביות, משתמשים רק ב־6, ולכן Six-Colors אכן עוצר עם צביעת־6.

בהנחה שכל מעבד יכול לקבוע את האינדקס המתאים i בזמן $O(1)$ ולבצע פעולת הזזה־שמאלה בזמן $O(1)$ — פעולות רגילות שמחשבים רבים תומכים בהן — כל איטרציה מתבצעת בזמן $O(1)$. השגרה Six-Colors היא אלגוריתם EREW: לכל עצם x, המעבד של x מבצע גישות ל־x ול־$next[x]$ בלבד.

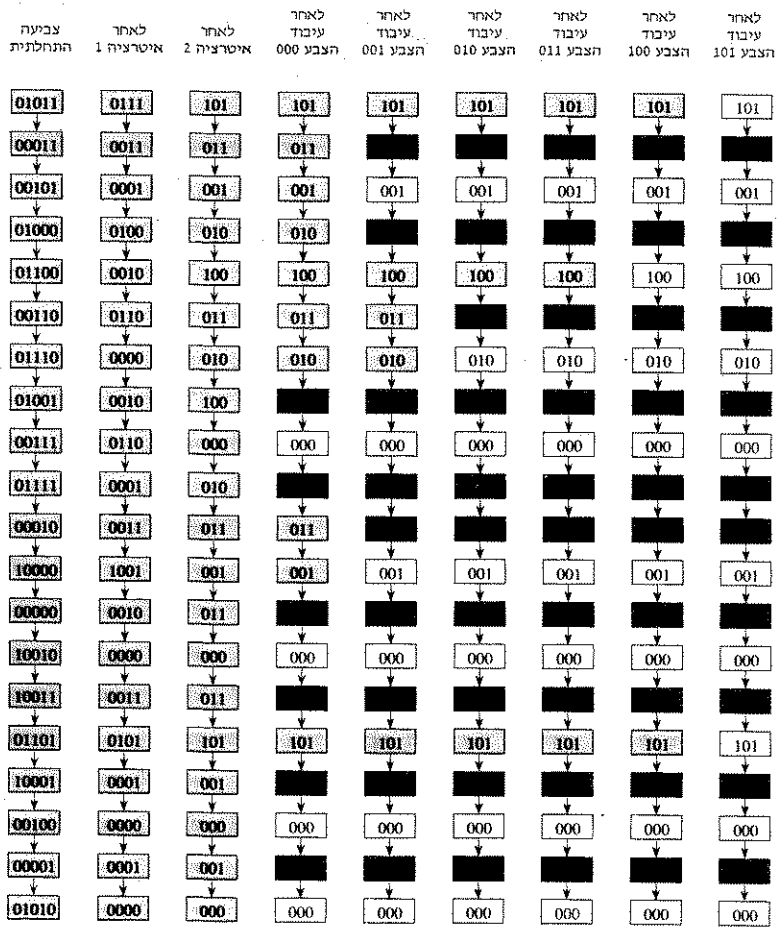
לבסוף, הבה נראה מדוע דרושות $O(\lg^* n)$ איטרציות בלבד כדי לצמצם את צביעת־n ההתחלתית לצביעת־6. הגדרנו את $\lg^* n$ כמספר הפעמים שיש להפעיל את פונקציית הלוגריתם על n כדי לקבל ערך קטן או שווה ל־1, או, אם נסמן i הפעלות עוקבות של פונקציית הלוגריתם על־ידי $\lg^{(i)} n$, אזי:

$$\lg^* n = \min \{ i \geq 0 : \lg^{(i)} n \leq 1 \}$$

יהי r_i מספר הסיביות המייצגות כל צבע בצביעה ממש לפני האיטרציה ה־i. אנו נוכיח באינדוקציה שאם $\lceil \lg^{(i)} n \rceil \geq 2$, אזי $r_i \leq \lceil \lg^{(i)} n \rceil + 2$. במצב ההתחלתי, $r_1 \leq \lceil \lg n \rceil$. האיטרציה ה־$i$ מצמצמת את מספר הסיביות המייצגות כל צבע בצביעה ל־$1 + \lceil \lg r_i \rceil = r_{i+1}$. אם נניח כי הנחת האינדוקציה מתקיימת עבור r_{i-1}, נקבל:

$$\begin{aligned}
r_i &= \lceil \lg r_{i-1} \rceil + 1 \\
&\leq \lceil \lg(\lceil \lg^{(i-1)} n \rceil + 2) \rceil + 1 \\
&\leq \lceil \lg(\lg^{(i-1)} n + 3) \rceil + 1 \\
&\leq \lceil \lg(2 \lg^{(i-1)} n) \rceil + 1 \\
&= \lceil \lg(\lg^{(i-1)} n) + 1 \rceil + 1 \\
&= \lceil \lg^{(i)} n \rceil + 2
\end{aligned}$$

השורה הרביעית נובעת מן ההנחה כי $\lceil \lg^{(i)} n \rceil \geq 2$, שפירושה כי $\lceil \lg^{(i-1)} n \rceil \geq 3$. לכן, לאחר $m = \lg^* n$ צעדים, מספר הסיביות המייצגות כל צבע בצביעה הוא $r_m \leq \lceil \lg^{(m)} n \rceil + 2 = 3$, שכן $\lg^{(m)} n \leq 1$, על פי הגדרת הפונקציה $\lg^*$. לכן, די בעוד איטרציה אחת לכל היותר כדי ליצור צביעת־6. זמן הריצה הכולל של Six-Colors הוא אפוא $O(\lg^* n)$.



איור 30.11 האלגוריתמים Six-Colors ו־List-MIS אשר שוברים סימטריה ברשימה. ביחד, האלגוריתמים מוצאים קבוצה גדולה של עצמים שאינם סמוכים זה לזה בזמן $O(\lg^* n)$, תוך שימוש ב־n מעבדים. הרשימה ההתחלתית של $n = 20$ העצמים מוצגת משמאל, מלמעלה למטה. במצב ההתחלתי, כל עצם צבוע בצבע שונה בן 5 סיביות. עבור פרמטרים אלה, האלגוריתם Six-Colors זקוק רק לשתי האיטרציות המופיעות באיור כדי לצבוע מחדש כל עצם בצבע הנמצא בתחום $\{0, 1, 2, 3, 4, 5\}$. העצמים הלבנים באיור הם אלה שאותם List-MIS מצרפת לקבוצה הבלתי תלויה מקסימלית תוך כדי עיבוד הצבעים, והשחורים הם אלה ש"ינזרקו".

בניית קבוצה בלתי תלויה מקסימלית מתוך צביעת־6

צביעה היא החלק הקשה בשבירת סימטריה. אלגוריתם LIST-MIS EREW משתמש ב־n מעבדים כדי למצוא קבוצה בלתי תלויה מקסימלית בזמן $O(c)$ בהינתן צביעת־c של רשימה בת n עצמים. לכן, ברגע שחישבנו צביעת־6 של רשימה, נוכל למצוא קבוצה בלתי תלויה מקסימלית של הרשימה המקושרת בזמן $O(1)$.

החלק הימני של איור 30.11 מדגים את הרעיון שמאחורי LIST-MIS. נתונה לנו צביעת־c שנקרא לה C. עבור כל עצם x, אנו שומרים סיבית $alive[x]$, המציינת אם x הוא עדיין מועמד להיכלל בקבוצה הבלתי תלויה מקסימלית. במצב ההתחלתי, $alive[x] = \text{TRUE}$ עבור כל העצמים x.

לאחד מכן האלגוריתם מבצע איטרציה עבור כל אחד מ־c הצבעים. באיטרציה עבור הצבע i, כל מעבד האחראי על עצם x בודק אם מתקיים $C[x] = i$ וגם $alive[x] = \text{TRUE}$. אם שני התנאים מתקיימים, המעבד מסמן את x כשייך לקבוצה ההולכת ונבנית. עבור כל העצמים הסמוכים לאלה שנוספו לקבוצה — אלה שנמצאים ממש לפניהם או מייד אחריהם — משנים את סיביות $alive$ שלהם ל־FALSE; הם אינם יכולים להשתייך לקבוצה הבלתי תלויה מקסימלית משום שהם סמוכים לעצם שכבר שייך לקבוצה זו. לאחר סיום כל c האיטרציות, כל עצם או שנזרק — בסיבית $alive$ שלו הוצב הערך FALSE — או שצורף לקבוצה הבלתי תלויה מקסימלית.

עלינו להראות שהקבוצה המתקבלת היא אכן בלתי־תלויה ומקסימלית. כדי לראות שהיא בלתי־תלויה, נניח ששני עצמים סמוכים x ו־$next[x]$ מצורפים לקבוצה. מאחר שהם סמוכים, $C[x] \neq C[next[x]]$, שכן C היא צביעה. ללא הגבלת הכלליות, נניח כי $C[x] < C[next[x]]$, כך ש־x מצורף לקבוצה לפני $next[x]$. אבל או הוצב ב־$alive[next[x]]$ הערך FALSE עוד לפני שהתחיל עיבוד העצמים שצבעם $C[next[x]]$, ולא ייתכן ש־$next[x]$ צורף לקבוצה.

כדי לראות שהקבוצה מקסימלית, נניח שאף עצם מבין שלושה עצמים עוקבים x, y ו־z, לא צורף לקבוצה. אולם האפשרות היחידה שבה נמנע צירופו של y לקבוצה היא זו שבה y כבר נזרק בעת שעצם סמוך לו צורף לקבוצה. מכיוון שעל פי הנחתנו x ו־z לא צורפו לרשימה, הרי שהעצם y בהכרח היה עדיין חי בזמן שבו עובדו עצמים שצבעם $C[y]$. לכן, בהכרח y צורף לקבוצה.

כל איטרציה של LIST-MIS מתבצעת על PRAM בזמן $O(1)$. LIST-MIS הוא אלגוריתם EREW, שכן כל עצם ניגש רק לעצמו, לקודם לו ולעוקב לו ברשימה. על־ידי צירוף של LIST-MIS ו־SIX-COLOR ניתן אפוא לשבור סימטריה ברשימה מקושרת בת n עצמים באופן דטרמיניסטי בזמן $O(\lg^* n)$.

תרגילים

30.5-1
הראה שבדוגמה של שבירת הסימטריה באמצעות שני מעבדים שהוצגה בתחילת הפרק, הסימטריה נשברת בתוך זמן קבוע.

30.5-2
בהינתן צביעת־6 של רשימה בת n עצמים, הראה כיצד ניתן לצבוע את הרשימה בצביעת־3 בזמן $O(1)$ באמצעות EREW PRAM בן n מעבדים.

30.5-3
נניח שכל צומת שאינו השורש בעץ n-צומתי מכיל מצביע לאביו. כתוב אלגוריתם CREW לצביעת־$O(1)$ של העץ בזמן $O(\lg^* n)$.

30.5-4 *
כתוב אלגוריתם PRAM יעיל לצביעת־$O(1)$ של גרף בעל דרגה 3. נתח את האלגוריתם שלך.

30.5-5
קבוצה k-שולטת (k-ruling set) ברשימה מקושרת היא קבוצה של עצמים (שליטים) כך שאין שני שליטים הסמוכים זה לזה, ובין כל שני שליטים מפרידים לכל היותר k עצמים שאינם שליטים (נתינים). לכן, קבוצה בלתי תלויה מקסימלית היא קבוצה 2־שולטת. הראה כיצד ניתן למצוא קבוצה $O(\lg n)$־שולטת ברשימה בת n עצמים בזמן $O(1)$ תוך שימוש ב־n מעבדים. הראה כיצד ניתן למצוא באותם תנאים קבוצה $O(\lg\lg n)$־שולטת בזמן $O(1)$.

30.5-6 *
הראה כיצד ניתן למצוא צביעת־6 של רשימה מקושרת בת n עצמים בזמן $O(\lg(\lg^* n))$. הנח שכל מעבד יכול לאחסן טבלה מחושבת מראש בגודל $O(\lg n)$. (**רמז:** בשגרה SIX-COLOR, בכמה ערכים תלוי צבעו הסופי של עצם?)

בעיות

30-1 חישוב רישות מקבילי מקוטע

כמו חישוב רישות רגיל, **חישוב רישות מקוטע** (segmented prefix computation) מוגדר במונחי אופרטור בינרי אסוציאטיבי $\otimes$. הקלט הוא סדרה $x = \langle x_1, x_2, \ldots, x_n \rangle$ שאיבריה לקוחים מתחום S, **וסדרת קיטוע** (segment sequence) $b = \langle b_1, b_2, \ldots, b_n \rangle$ שאיבריה לקוחים מן התחום $\{0, 1\}$, כאשר $b_1 = 1$. הפלט הוא סדרה $y = \langle y_1, y_2, \ldots, y_n \rangle$ מעל התחום S. הסיביות של b קובעות חלוקה של x ושל y לקטעים; קטע חדש מתחיל בכל פעם ש־$b_i = 1$, והקטע הנוכחי נמשך אם $b_i = 0$. חישוב הרישות המקוטע מבצע חישוב רישות בלתי תלוי בתוך כל קטע של x לקבלת הקטע המתאים של y. איור 30.12 מדגים חישוב רישות מקוטע תוך שימוש בחיבור הרגיל.

b =	1	0	0	1	0	1	1	0	0	0	0	0	1	0
x =	1	2	3	4	5	6	7	8	9	10	11	12	13	14
y =	1	3	6	4	9	6	7	15	24	34	45	57	13	27

איור 30.12 חישוב רישות מקוטע עם סדרת קיטוע b, סדרת קלט x וסדרת פלט y. ישנם 5 קטעים.

א. נגדיר את האופרטור $\hat{\otimes}$ על זוגות סדורים $(a, z), (a', z') \in \{0, 1\} \times S$ כדלקמן:

$$(a, z) \hat{\otimes} (a', z') = \begin{cases} (a, z \otimes z') & \text{אם } a' = 0 \\ (1, z') & \text{אם } a' = 1 \end{cases}$$

הוכח כי $\otimes$ אסוציאטיבי.

ב. הראה כיצד ניתן לממש כל חישוב רישות מקוטע על רשימה בת n איברים באמצעות EREW PRAM בזמן $O(\lg n)$.

ג. תאר אלגוריתם EREW שרץ בזמן $O(k \lg n)$ למיון רשימה בת n מספרים שכל אחד מהם בן k סיביות.

30-2 אלגוריתם יעיל במעבדים למציאת מקסימום

ברצוננו למצוא את המקסימום מבין n מספרים באמצעות CRCW PRAM בעל $p = n$ מעבדים.

א. הראה שניתן לעשות רדוקציה של הבעיה של מציאת המקסימום מבין $m \le p/2$ מספרים לבעיה של מציאת המקסימום מבין m^2/p מספרים לכל היותר בזמן $O(1)$, באמצעות CRCW PRAM בן p מעבדים.

ב. אם מתחילים עם $m = \lfloor p/2 \rfloor$ מספרים, כמה מספרים יישארו לאחר k איטרציות של האלגוריתם מחלק (א)?

ג. הראה שניתן לפתור את הבעיה של מציאת המקסימום מבין n מספרים בזמן $O(\lg\lg n)$ באמצעות CRCW PRAM בן $p = n$ מעבדים.

30-3 רכיבים קשירים

בבעיה זו נחקור אלגוריתם CRCW שרירותי לחישוב הרכיבים הקשירים בגרף בלתי־מכוון $G = (V, E)$ המשתמש ב־$|V| + |E|$ מעבדים. מבנה הנתונים שמשתמשים בו הוא יער קבוצות־זרות (ראה סעיף 22.3), בכל קדקוד $v \in V$ נשמר מצביע $p[v]$ לאב. במצב ההתחלתי, $p[v] = v$: הקדקוד מצביע על עצמו. בסיום האלגוריתם, עבור כל שני קדקודים $u, v \in V$ מקבלים $p[u] = p[v]$ אם ורק אם $u \rightsquigarrow v$ ב־G. במהלך האלגוריתם, המצביעים p יוצרים יער של **עצי מצביעים** (pointer trees) מושרשים. **כוכב** (star) הוא עץ מצביעים שבו $p[u] = p[v]$ עבור כל הקדקודים u ו־v בעץ.

האלגוריתם למציאת הרכיבים הקשירים מניח שכל קשת $(u, v) \in E$ מופיעה פעמיים: פעם כ־(u, v) ופעם כ־(v, u). האלגוריתם משתמש בשתי פעולות בסיסיות, HOOK ו־JUMP, ובשגרה STAR המציבה $star[v] = \text{TRUE}$ אם v שייך לכוכב.

```
HOOK(G)
 1  STAR(G)
 2  for each edge (u, v) ∈ E[G], in parallel
 3      do if star[u] and p[u] > p[v]
 4          then p[p[u]] ← p[v]
 5  STAR(G)
 6  for each edge (u, v) ∈ E[G], in parallel
 7      do if star[u] and p[u] ≠ p[v]
 8          then p[p[u]] ← p[v]

JUMP(G)
 1  for each v ∈ V[G], in parallel
 3      do p[v] ← p[p[v]]
```

האלגוריתם למציאת הרכיבים הקשירים מבצע פעולת HOOK התחלתית ולאחר מכן חוזר ומבצע HOOK, JUMP, HOOK, JUMP, וכן הלאה, עד אשר פעולת JUMP שוב אינה משנה שום מצביע. (יש לשים לב לכך שלפני פעולת ה־JUMP הראשונה מבוצעות שתי פעולות HOOK.)

א. כתוב פסאודו־קוד עבור $\text{STAR}(G)$.

ב. הראה שהמצביעים p אכן יוצרים עצים מושרשים, כאשר שורש העץ מצביע על עצמו. הראה שאם u ו־v שייכים לאותו עץ מצביעים, אזי $u \rightsquigarrow v$ ב־G.

ג. הראה שהאלגוריתם נכון: הוא עוצר, וכאשר הוא עוצר, $p[u] = p[v]$ אם ורק אם $u \rightsquigarrow v$ ב־G.

כדי לנתח את האלגוריתם למציאת הרכיבים הקשירים, הבה נבחן רכיב קשיר אחד C, אשר, אנו מניחים, מכיל לפחות שני קדקודים. נניח שבשלב מסוים במהלך האלגוריתם, C מורכב מקבוצה $\{T_i\}$ של עצי מצביעים. נגדיר את הפוטנציאל של C על־ידי:

$$\Phi(C) = \sum_{T_i} \text{height}(T_i)$$

מטרת הניתוח שלנו היא להוכיח שכל איטרציה של HOOK ו־JUMP מקטינה את $\Phi(C)$ פי גורם קבוע.

ד. הוכח שלאחר ביצוע פעולת ה־HOOK ההתחלתית, לא קיימים עצי מצביעים שגובהם 0, ו־$\Phi(C) \le |V|$.

ה. הראה שלאחר ה־HOOK ההתחלתי, הפעולות HOOK העוקבות לעולם אינן מגדילות את $\Phi(C)$.

ו. הראה שלאחר כל פעולת HOOK מלבד הראשונה, שום עץ מצביעים אינו כוכב אלא אם כן הוא מכיל את כל הקדקודים ב־C.

ז. הראה שאם C לא קרס לכוכב יחיד, אזי לאחר פעולת JUMP, ערכו של $\Phi(C)$ שווה לכל היותר ל־$2/3$ מערכו הקודם. הדגם את המקרה הגרוע ביותר.

ח. הסק שהאלגוריתם מוצא את כל הרכיבים הקשירים ב־G בזמן $O(\lg V)$.

30-4 חילוף תמונת סריקה

על חוצץ מסגרת (frame buffer) של גרפיקת־סריקה ניתן להסתכל כעל מטריצת סיביות M שממדיה $p \times p$. התקן התצוגה של גרפיקת סריקה מציג על המסך את התת־מטריצה השמאלית העליונה בגודל $n \times n$ של M. פעולת BITBLT (BLock Transfer of BITs), העברת בלוקים של סיביות) משמשת להעברת מלבן סיביות ממיקום אחד לאחר. ובמפורט, הפעולה $\text{BITBLT}(r_1, c_1, r_2, c_2, nr, nc, *)$, מציבה:

$$M[r_2 + i, c_2 + j] \leftarrow M[r_2 + i, c_2 + j] * M[r_1 + i, c_1 + j]$$

עבור $i = 0, 1, \ldots, nr - 1$ ו־$j = 0, 1, \ldots, nc - 1$, כאשר $*$ יכולה להיות כל אחת מ־16 הפונקציות הבוליאניות על שני קלטים.

אנו מעוניינים לבצע חילוף (טרנספוזיציה) של התמונה (כלומר, לבצע $M[i, j] \leftarrow M[j, i]$) בחלק המוצג של חוצץ המסגרת. אנו מניחים שהעלות של העתקת סיביות קטנה מעלות הקריאה לפעולת היסוד BITBLT, ולכן אנו מעוניינים להשתמש בפעולות BITBLT מעטות ככל האפשר.

הראה שניתן לבצע חילוף של תמונה כלשהי המופיעה על המסך באמצעות $O(\lg n)$ פעולות BITBLT. הנח ש־p גדול מ־n במידה מספקת, כך שהחלק הבלתי־נראה של חוצץ המסגרת מספק מקום אחסון גדול דיו לצורך עבודה. לכמה מקום אחסון נוסף אתה זקוק? (**רמז:** השתמש בגרסה המקבילות של גישת הפרד־ומשול שבה חלק מן הפעולות BITBLT מבוצעות עם פעולות AND בוליאניות.)

הערות ביבליוגרפיות לפרק 30

Akl [9], Karp & Ramachandran [118], ו־Leighton [135] סוקרים אלגוריתמים מקביליים עבור בעיות קומבינטוריות. ארכיטקטורות שונות של מכונות מקביליות מתוארות אצל Hwang & Briggs [109] ואצל Hwang & DeGroot [110].

תיאוריית החישוב המקבילי החלה להתפתח בשנות ה־40' המאוחרות, כאשר J.Von Neumann [38] הציג מודל מוגבל של חישוב מקבילי שנקרא אוטומט תאי (cellular automaton), שהוא במהותו מערך דו־ממדי של מעבדים בעלי מספר סופי של מצבים, הקשורים ביניהם במעין רשת שתי וערב. מודל ה־PRAM הוגדר באופן פורמלי ב־1978 על־ידי Fortune & Wyllie [73], אם כי מדענים רבים אחרים דנו עוד קודם לכך במודלים דומים במהותם.

הקפצת מצביעים פותחה על ידי Wyllie [204]. חקר חישוב הרישות המקבילי התפתח מעבודתו של Ofman [152] בהקשר של חיבור צופה־נשא. שיטת מסלול־אוילר היא של Tarjan & Vishkin [191].

יחסי "קח־ותן" בין זמן הריצה למספר המעבדים במציאת המקסימום מבין קבוצה של n מספרים הוצגו על ידי Valiant [193], שגם הראה כי לא קיים אלגוריתם יעיל בעבודה שרץ בזמן $O(1)$. Cook, Dwork & Reischuk [50] הוכיחו שבעיית חישוב המקסימום דורשת זמן של $\Omega(\lg n)$ על CREW PRAM. ההדמיה של אלגוריתם CRCW על ידי אלגוריתם EREW היא של Vishkin [195].

משפט 30.2 הוא של Brent [34]. האלגוריתם האקראי היעיל בעבודה למציאת מיקומם של אינברים ברשימה התגלה על ידי Anderson & Miller [11]. הם הציגו גם אלגוריתם דטרמיניסטי יעיל בעבודה עבור בעיה זו [10]. האלגוריתם לשבירה דטרמיניסטית של סימטריה הוא של Goldberg & Plotkin [84]. הוא מבוסס על אלגוריתם דומה בעל אותו זמן ריצה של Cole & Vishkin [47].

31 פעולות על מטריצות

פעולות על מטריצות תופסות מקום מרכזי בחישובים מדעיים. לכן, יש עניין מעשי רב באלגוריתמים יעילים לעבודה עם מטריצות. פרק זה מספק מבוא קצר לתורת המטריצות ולפעולות על מטריצות, ושם דגש על בעיות של כפל מטריצות ושל פתרון מערכות של משוואות לינאריות.

לאחר שנביא בסעיף 31.1 מושגים וסימונים בסיסיים בתורת המטריצות, נציג בסעיף 31.2 את האלגוריתם המפתיע של שטראסן (Strassen) לכפל שתי מטריצות בגודל $n \times n$ בזמן $\Theta(n^{\lg 7}) = O(n^{2.81})$. בסעיף 31.3 נגדיר מעין־חוגים (quasirings), חוגים ושדות, ונבחיר את ההגדרות הדרושות כדי שאלגוריתם שטראסן יפעל היטב. סעיף זה יציג גם אלגוריתם מהיר אסימפטוטית לכפל מטריצות בוליאניות. בסעיף 31.4 נראה כיצד פותרים מערכת משוואות לינאריות באמצעות פירוקי LUP. לאחר מכן נחקור בסעיף 31.5 את הקשר ההדוק בין הבעיה של כפל מטריצות לבין הבעיה של חישוב המטריצה ההופכית. לבסוף, סעיף 31.6 ידון במחלקה החשובה של מטריצות סימטריות חיוביות־לחלוטין, ויראה כיצד ניתן להשתמש בהן למציאת פתרון באמצעות ריבועים פחותים למערכת מוגדרת־יתר של משוואות לינאריות.

31.1 תכונות של מטריצות

בסעיף זה נחזור על כמה מושגים בסיסיים בתורת המטריצות ועל כמה תכונות בסיסיות של מטריצות, תוך התמקדות באלה שנזדקק להם בסעיפים הבאים.

מטריצות ווקטורים

מטריצה (matrix) היא מערך מלבני של מספרים. לדוגמה:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \tag{31.1}$$

$$= \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

היא מטריצה $A = (a_{ij})$ שממדיה 2×3, כאשר עבור $i = 1, 2$ ו־$j = 1, 2, 3$, האיבר בשורה ה־i ובעמודה ה־j הוא a_{ij}. אנו מסמנים מטריצות באותיות לטיניות גדולות ואת איבריהן באותיות קטנות בצירוף האינדקסים המתאימים. קבוצת כל המטריצות שממדיהן $m \times n$ ואיבריהן בעלי ערכים ממשיים מסומנת על־ידי $\mathbf{R}^{m \times n}$. ככלל, קבוצת כל המטריצות שממדיהן $m \times n$ ואיבריהן לקוחים מקבוצה S, מסומנת על־ידי $S^{m \times n}$.

המטריצה המוחלפת (transpose) של מטריצה A היא המטריצה A^{T} המתקבלת על־ידי החלפת השורות והעמודות אלה באלה במטריצה A. המטריצה המוחלפת של המטריצה A ממשוואה (31.1) היא:

$$A^{\mathrm{T}} = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

וקטור (vector) הוא מערך חד־ממדי של מספרים. לדוגמה:

$$x = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix} \tag{31.2}$$

הוא וקטור בגודל 3. אנו מסמנים וקטורים באותיות לטיניות קטנות, ואת האיבר ה־i של וקטור x שגודלו n, אנו מסמנים על־ידי x_i, עבור $i = 1, 2, \ldots, n$. הצורה הסטנדרטית של וקטור תהיה עבורנו **וקטור עמודה** (column vector) השקול למטריצה $n \times 1$; **וקטור השורה** (row vector) המתאים הוא הווקטור המוחלף:

$$x^{\mathrm{T}} = (2 \;\; 3 \;\; 5)$$

וקטור היחידה e_i (unit vector) הוא הווקטור שבו האיבר ה־i הוא 1 וכל האיברים האחרים הם 0. בדרך כלל, גודלו של וקטור יחידה ברור מן ההקשר.

מטריצת אפס (zero matrix) היא מטריצה שכל איבריה הם 0. מטריצה כזו מסומנת לעתים קרובות ב־0, שכן מתוך ההקשר ניתן להבין בדרך כלל אם הכוונה למספר 0 או למטריצה של אפסים. במקרה שהכוונה היא למטריצת אפס, גם את גודלה יש להסיק מן ההקשר.

לעתים קרובות עוסקים **במטריצות ריבועיות** (square matrix), דהיינו, מטריצות שממדיהן $n \times n$. סוגים אחדים של מטריצות ריבועיות מיוחדות מעניינים אותנו במיוחד:

1. **מטריצה אלכסונית** (diagonal matrix) היא מטריצה שבה $a_{ij} = 0$ עבור כל $i \neq j$. מכיוון שכל האיברים שמחוץ לאלכסון הם אפס, ניתן לציין את המטריצה על־ידי רישום האיברים לאורך האלכסון:

$$\operatorname{diag}(a_{11}, a_{22}, \ldots, a_{nn}) = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix}$$

2. **מטריצת היחידה** I_n (identity matrix) היא מטריצה אלכסונית בגודל $n \times n$ שבה כל איברי האלכסון הם 1:

$$I_n = \operatorname{diag}(1, 1, \ldots, 1) = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

כאשר הסימן I מופיע ללא אינדקס, ניתן להסיק מן ההקשר את גודלה של מטריצת היחידה. העמודה ה־i של מטריצת יחידה היא וקטור היחידה e_i.

3. **מטריצה תלת־אלכסונית** (tridiagonal matrix) T היא מטריצה שבה $t_{ij} = 0$ כאשר $|i - j| > 1$. איברים שונים מאפס מופיעים רק באלכסון הראשי, ממש מעליו ($t_{i,i+1}$ עבור $i = 1, 2, \ldots, n-1$) וממש מתחתיו ($t_{i+1,i}$ עבור $i = 1, 2, \ldots, n-1$):

$$T = \begin{pmatrix} t_{11} & t_{12} & 0 & 0 & \cdots & 0 & 0 & 0 \\ t_{21} & t_{22} & t_{23} & 0 & \cdots & 0 & 0 & 0 \\ 0 & t_{32} & t_{33} & t_{34} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & t_{n-2,n-2} & t_{n-2,n-1} & 0 \\ 0 & 0 & 0 & 0 & \cdots & t_{n-1,n-2} & t_{n-1,n-1} & t_{n-1,n} \\ 0 & 0 & 0 & 0 & \cdots & 0 & t_{n,n-1} & t_{nn} \end{pmatrix}$$

4. **מטריצה משולשית עליונה** U (upper-triangular matrix) היא מטריצה שבה $u_{ij} = 0$ עבור $i > j$. כל האיברים מתחת לאלכסון הם אפס:

$$U = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{pmatrix}$$

מטריצה משולשית עליונה נקראת **מטריצת יחידה משולשית עליונה** (unit upper-triangular) אם כל האיברים לאורך האלכסון הם 1.

5. **מטריצה משולשית תחתונה** L (lower-triangular matrix) היא מטריצה שבה $l_{ij} = 0$ עבור $i < j$. כל האיברים מעל לאלכסון הם אפס:

$$L = \begin{pmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix}$$

מטריצה משולשית תחתונה היא **מטריצת יחידה משולשית תחתונה** (unit lower-triangular) אם כל האיברים לאורך האלכסון הם 1.

6. **מטריצת תמורה** (permutation matrix) P היא מטריצה שבה מופיע בדיוק 1 אחד בכל שורה ובכל עמודה, ושאר האיברים הם אפס. דוגמה למטריצת תמורה היא:

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

מטריצה כזאת נקראת מטריצת תמורה מכיוון שכאשר מכפילים וקטור x במטריצה כזאת מתקבלת תמורה (סידור מחדש) של איברי x.

7. **מטריצה סימטרית** (symmetric matrix) A מקיימת את התנאי $A = A^{\mathrm{T}}$. דוגמה למטריצה סימטרית היא:

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 6 & 4 \\ 3 & 4 & 5 \end{pmatrix}$$

פעולות על מטריצות

האיברים של מטריצה או וקטור הם מספרים הלקוחים מתוך מערכת מספרים, כגון המספרים הממשיים, המספרים המרוכבים או המספרים השלמים מודולו מספר ראשוני. מערכת המספרים מגדירה כיצד יש לחבר ולכפול מספרים. ניתן להרחיב הגדרות אלה כך שיכללו גם חיבור וכפל מטריצות.

נגדיר **חיבור מטריצות** (matrix addition) באופן הבא: אם $A = (a_{ij})$ ו־$B = (b_{ij})$ הן מטריצות בגודל $m \times n$, אזי מטריצת הסכום שלהן $C = (c_{ij}) = A + B$ היא מטריצה בגודל $m \times n$ המוגדרת על־ידי:

$$c_{ij} = a_{ij} + b_{ij}$$

עבור $i = 1, 2, \ldots, m$ ו־$j = 1, 2, \ldots, n$. כלומר, חיבור מטריצות מתבצע איבר איבר. מטריצת האפס היא איבר היחידה עבור חיבור מטריצות:

$$\begin{aligned} A + 0 &= A \\ &= 0 + A \end{aligned}$$

אם λ הוא מספר ו־$A = (a_{ij})$ היא מטריצה, אזי $\lambda A = (\lambda a_{ij})$ היא הכפולה של A **בסקלר** (scalar multiple), כלומר, המטריצה המתקבלת כאשר כופלים ב־λ כל אחד מאיברי A. מקרה פרטי של כפל בסקלר היא המטריצה **הנגדית** (negative) של מטריצה $A = (a_{ij})$, המוגדרת על־ידי $-A = -1 \cdot A$, כך שהאיבר ה־ij של $-A$ הוא $-a_{ij}$. כלומר:

$$\begin{aligned} A + (-A) &= 0 \\ &= (-A) + A \end{aligned}$$

בהינתן הגדרה זו, נוכל להגדיר **חיסור מטריצות** (matrix subtraction) כחיבור המטריצה הנגדית: $A - B = A + (-B)$.

נגדיר **כפל מטריצות** (matrix multiplication) באופן הבא: מתחילים עם שתי מטריצות A ו־B שהן **תואמות** (compatible) במובן זה שמספר העמודות ב־A שווה למספר השורות ב־B. (ככלל, אם ביטוי מכיל מכפלת מטריצות AB, ההנחה היא שמשתמע מכך כי A ו־B תואמות.) אם $A = (a_{ij})$ היא מטריצה בגודל $m \times n$ ו־$B = (b_{jk})$ היא מטריצה בגודל $n \times p$, אזי מטריצת המכפלה $C = AB$ היא המטריצה $C = (c_{ik})$, שגודלה $m \times p$, ואשר בה:

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk} \tag{31.3}$$

עבור $i = 1, 2, \ldots, m$ ו־$k = 1, 2, \ldots, p$. השגרה MATRIX-MULTIPLY מסעיף 26.1 מממשת כפל מטריצות בצורה ישירה המבוססת על משוואה (31.3), בהנחה שהמטריצות ריבועיות: $m = n = p$. כדי לכפול שתי מטריצות בגודל $n \times n$, MATRIX-MULTIPLY* מבצעת n^3 פעולות כפל ו־$n^2(n-1)$ פעולות חיבור, וזמן הריצה שלה הוא $\Theta(n^3)$.

מטריצות מקיימות רבות מן התכונות האלגבריות האופייניות למספרים (אולם לא את כולן). מטריצות יחידה הן איבר היחידה עבור כפל מטריצות; עבור כל מטריצה A בגודל $m \times n$:

$$I_m A = A I_n = A$$

כפל במטריצת אפס נותן מטריצת אפס:

$$A0 = 0$$

כפל מטריצות הוא אסוציאטיבי; עבור מטריצות תואמות A, B ו־C, מתקיים:

$$A(BC) = (AB)C \tag{31.4}$$

כפל מטריצות מתפלג מעל לחיבור:

$$\begin{aligned} A(B + C) &= AB + AC \\ (B + C)D &= BD + CD \end{aligned} \tag{31.5}$$

אולם כפל של מטריצות בגודל $n \times n$ אינו חילופי (קומוטטיבי), אלא אם כן $n = 1$. לדוגמה, אם $A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ ו־$B = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$, אזי:

$$AB = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

אבל:

$$BA = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

כפל מטריצה בווקטור או כפל שני וקטורים מוגדר כאילו הווקטור הוא המטריצה השקולה בגודל $n \times 1$ (או המטריצה בגודל $1 \times n$ במקרה של וקטור שורה). לכן, אם A היא מטריצה בגודל $m \times n$ ו־x הוא וקטור בגודל n, אזי Ax הוא וקטור בגודל m. אם x ו־y הם שני וקטורים בגודל n, אזי:

$$x^{\mathrm{T}} y = \sum_{i=1}^{n} x_i y_i$$

הוא מספר (למעשה, מטריצה בגודל 1×1). מספר זה נקרא **המכפלה הפנימית** (inner product) של x ו־y. המטריצה xy^{T} היא מטריצה Z בגודל $n \times n$ שבה $z_{ij} = x_i y_j$. Z נקראת **המכפלה החיצונית** (outer product) של x ו־y. **הנורמה (האויקלידית)** (euclidean norm) של וקטור x שגודלו n מוגדרת על־ידי:

$$\|x\| = (x_1^2 + x_2^2 + \cdots + x_n^2)^{1/2}$$
$$= (x^T x)^{1/2}$$

הנורמה של x היא אפוא אורכו של x במרחב האויקלידי ה־n ממדי.

מטריצות הופכיות, דרגות ודטרמיננטות

המטריצה **ההופכית** (inverse) למטריצה A בגודל $n \times n$ היא המטריצה בגודל $n \times n$ המסומנת על־ידי A^{-1} (אם היא קיימת) והמקיימת $AA^{-1} = I_n = A^{-1}A$. לדוגמה:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{-1} = \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix}$$

לרבות מן המטריצות בגודל $n \times n$ השונות מאפס אין מטריצות הופכיות. מטריצה שאין לה הופכית נקראת **בלתי־הפיכה** (noninvertible) או **סינגולרית** (singular). דוגמה למטריצה סינגולרית שונה מאפס היא:

$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$$

מטריצה שיש לה הופכית מכונה **הפיכה** (invertible), או **לא־סינגולרית**, או **רגולרית**. מטריצות הופכיות, אם הן קיימות, הן יחידות. (ראה תרגיל 31.1-4.) אם A ו־B הן מטריצות רגולריות בגודל $n \times n$, אזי:

$$(BA)^{-1} = A^{-1}B^{-1} \tag{31.6}$$

פעולת חישוב המטריצה ההופכית ופעולת חישוב המטריצה המוחלפת הן קומוטטיביות זו ביחס לזו:

$$(A^{-1})^T = (A^T)^{-1}$$

הווקטורים $x_1, x_2, \ldots, x_n$ הם **תלויים לינארית** (linearly dependent) אם קיימים מקדמים $c_1, c_2, \ldots, c_n$, שלא כולם אפס, כך שמתקיים: $c_1x_1 + c_2x_2 + \cdots + c_nx_n = 0$. לדוגמה, הווקטורים $x_1 = (1\ 2\ 3)^T$, $x_2 = (2\ 6\ 4)^T$ ו־$x_3 = (4\ 11\ 9)^T$ תלויים לינארית, שכן $2x_1 + 3x_2 - 2x_3 = 0$. וקטורים שאינם מקיימים את תכונת התלות הלינארית נקראים **בלתי־תלויים לינארית** (linearly independent). לדוגמה, העמודות של מטריצת יחידה הן בלתי־תלויות לינארית.

דרגת העמודות (column rank) של מטריצה A בגודל $n \times m$ שונה מאפס היא גודלה של הקבוצה הגדולה ביותר של עמודות בלתי־תלויות לינארית ב־A. באופן דומה, **דרגת השורות** (row rank) של A היא גודל הקבוצה הגדולה ביותר של שורות בלתי־תלויות לינארית ב־A. תכונה בסיסית של כל מטריצה A היא שדרגת השורות שלה תמיד שווה לדרגת העמודות שלה, כך שניתן פשוט לדבר על **הדרגה** (rank) של A. הדרגה של מטריצה בגודל $n \times m$ היא מספר שלם בין 0 ל־$\min(m, n)$, כולל. (דרגתה של מטריצת האפס היא 0, ודרגתה של מטריצת יחידה בגודל $n \times n$ היא n.) הגדרה חלופית, אך שקולה ולעתים קרובות שימושית יותר, היא, שהדרגה של מטריצה A בגודל $n \times m$ השונה מאפס היא המספר הקטן ביותר r כך שקיימות מטריצות B ו־C, בגודל $m \times r$ ו־$r \times n$ בהתאמה, המקיימות:

$$A = BC$$

מטריצה ריבועית A בגודל $n \times n$ היא בעלת **דרגה מלאה** (full rank) אם דרגתה היא n. תכונה בסיסית של דרגות נתונה במשפט הבא.

משפט 31.1
מטריצה ריבועית היא בעלת דרגה מלאה אם ורק אם היא רגולרית. ■

מטריצה בגודל $n \times m$ היא בעלת **דרגת עמודות מלאה** (full column rank) אם דרגתה היא n.

וקטור מאפס (null vector) עבור מטריצה A הוא וקטור x שונה מאפס המקיים $Ax = 0$. המשפט הבא, שאת הוכחתו השארנו כתרגיל 31.1-8, והמסקנה הנובעת ממנו קושרים בין דרגת עמודות וסינגולריות לבין וקטורים מאפסים.

משפט 31.2
מטריצה A היא בעלת דרגת עמודות מלאה אם ורק אם אין לה וקטור מאפס. ■

מסקנה 31.3
מטריצה ריבועית A היא סינגולרית אם ורק אם יש לה וקטור מאפס. ■

המינור (minor) ה־ij של מטריצה A בגודל $n \times n$, עבור $n > 1$, הוא המטריצה $A_{[ij]}$ שגודלה $(n-1) \times (n-1)$, המתקבלת על־ידי מחיקת השורה ה־i והעמודה ה־j של A. את **הדטרמיננטה** (determinant) של מטריצה A בגודל $n \times n$ ניתן להגדיר הגדרה רקורסיבית, במונחי המינורים שלה, על־ידי:

$$\det(A) = \begin{cases} a_{11} & \text{אם } n = 1 \\ a_{11}\det(A_{[11]}) - a_{12}\det(A_{[12]}) & \\ \quad + \cdots + (-1)^{n+1} a_{1n} \det(A_{[1n]}) & \text{אם } n > 1 \end{cases} \tag{31.7}$$

הביטוי $(-1)^{i+j}\det(A_{[ij]})$ מכונה **המשלים** (cofactor) של האיבר a_{ij}.

המשפטים הבאים, שהוכחותיהם אינן מובאות כאן, מבטאים תכונות בסיסיות של דטרמיננטות.

משפט 31.4 (תכונות הדטרמיננטה)
הדטרמיננטה של מטריצה ריבועית A מקיימת את התכונות הבאות:
- אם שורה כלשהי או עמודה כלשהי ב־A שוות לאפס, אזי $\det(A) = 0$.
- אם מכפילים את כל האיברים בשורה אחת (או בעמודה אחת) כלשהי ב־λ, אזי הדטרמיננטה של A מוכפלת ב־λ.
- הדטרמיננטה של A אינה משתנה אם מחברים את האיברים בשורה אחת לאלה שבשורה אחרת (ובאותו אופן לגבי עמודות).
- הדטרמיננטה של A שווה לדטרמיננטה של A^T.
- אם מחליפים זו בזו שתי שורות (או שתי עמודות), הדטרמיננטה של A מוכפלת ב־-1.

כמו כן, עבור כל שתי מטריצות ריבועיות A ו־B מתקיים $\det(AB) = \det(A)\det(B)$. ■

משפט 31.5
מטריצה A בגודל $n \times n$ היא סינגולרית אם ורק אם $\det(A) = 0$. ■

מטריצות חיוביות־לחלוטין

מטריצות חיוביות־לחלוטין ממלאות תפקיד חשוב ביישומים רבים. מטריצה A בגודל $n \times n$ היא **חיובית־לחלוטין** (positive-definite) אם $x^T A x > 0$ עבור כל וקטור $x \neq 0$ בגודל n. לדוגמה, מטריצת היחידה היא מטריצה חיובית־לחלוטין, שכן עבור כל וקטור שונה מאפס $x = (x_1\ x_2\ \cdots\ x_n)^T$ מתקיים:

$$\begin{aligned} x^T I_n x &= x^T x \\ &= \|x\|^2 \\ &= \sum_{i=1}^{n} x_i^2 \\ &> 0 \end{aligned}$$

כפי שנראה בהמשך, מטריצות המופיעות ביישומים הן לעתים קרובות חיוביות־לחלוטין עקב המשפט הבא.

משפט 31.6
עבור כל מטריצה A בעלת דרגת עמודות מלאה, המטריצה $A^T A$ היא חיובית־לחלוטין.

הוכחה עלינו להראות כי $x^T(A^T A)x > 0$ עבור כל וקטור x שונה מאפס. עבור כל וקטור x מתקיים:

$$\begin{aligned} x^T(A^T A)x &= (Ax)^T(Ax) && \text{(על פי תרגיל 31.1-3)} \\ &= \|Ax\|^2 && (31.8) \\ &\geq 0 \end{aligned}$$

נשים לב כי $\|Ax\|^2$ אינו אלא סכום הריבועים של איברי הווקטור Ax. לכן, אם $\|Ax\|^2 = 0$, כל איבר של Ax הוא אפס, כלומר, $Ax = 0$. מכיוון ש־A היא בעלת דרגת עמודות מלאה, $Ax = 0$ גורר $x = 0$, על פי משפט 31.2. לכן $A^T A$ היא חיובית־לחלוטין. ■

בסעיף 31.6 נחקור תכונות נוספות של מטריצות חיוביות־לחלוטין.

תרגילים

31.1-1
הוכח שהמכפלה של שתי מטריצות משולשיות תחתונות היא מטריצה משולשית תחתונה. הוכח שהדטרמיננטה של מטריצה משולשית (תחתונה או עליונה) שווה למכפלת איברי האלכסון של המטריצה. הוכח שהמטריצה ההופכית למטריצה משולשת תחתונה, אם היא קיימת, היא משולשית תחתונה.

31.1-2
הוכח שאם P היא מטריצת תמורה בגודל $n \times n$ ו־A היא מטריצה ריבועית בגודל $n \times n$, אזי ניתן לקבל את PA מ־A על ידי תמורה של שורותיה, ואת AP מ־A על ידי תמורה של עמודותיה. הוכח שמכפלתן של שתי מטריצות תמורה היא מטריצת תמורה. הוכח שאם P היא מטריצת תמורה, אזי P הפיכה, ההופכית שלה היא P^T, ו־P^T היא מטריצת תמורה.

31.1-3
הוכח כי $(AB)^T = B^T A^T$ וכי $A^T A$ היא תמיד מטריצה סימטרית.

31.1-4
הוכח שאם B ו־C הן מטריצות הופכיות של A, אזי $B = C$.

31.1-5
יהיו A ו־B מטריצות בגודל $n \times n$ המקיימות $AB = I$. הוכח שאם A' מתקבלת מ־A על־ידי חיבור השורה j לשורה i, אזי ניתן לקבל את ההופכית B' של A' על־ידי חיסור העמודה i מהעמודה j ב־B.

31.1-6
תהי A מטריצה רגולרית $n \times n$ שאיבריה מספרים מרוכבים. הראה שכל איבר של A^{-1} הוא ממשי אם ורק אם כל איבר של A הוא ממשי.

31.1-7
הראה שאם A היא מטריצה סימטרית רגולרית, אזי A^{-1} היא סימטרית. הראה שאם B היא מטריצה כלשהי (תואמת ל־A), אזי BAB^T היא סימטרית.

31.1-8
הראה שמטריצה A היא בעלת דרגת עמודות מלאה אם ורק אם $Ax = 0$ גורר $x = 0$. (**רמז:** בטא את התלות הלינארית של עמודה אחת באחרות כמשוואת מטריצה־וקטור.)

31.1-9
הוכח שעבור כל שתי מטריצות תואמות A ו־B:

$$\text{rank}(AB) \leq \min(\text{rank}(A), \text{rank}(B))$$

כאשר שוויון מתקיים אם אחת מן המטריצות היא ריבועית ורגולרית. (**רמז:** השתמש בהגדרה החלופית של דרגתה של מטריצה.)

31.1-10
בהינתן מספרים $x_0, x_1, \ldots, x_{n-1}$, הוכח שהדטרמיננטה של **מטריצת ואנדרמונדה** (Vandermonde matrix):

$$V(x_0, x_1, \ldots, x_{n-1}) = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix}$$

היא:

$$\det(V(x_0, x_1, \ldots, x_{n-1})) = \prod_{0 \leq j < k \leq n-1} (x_k - x_j)$$

(**רמז:** עבור $i = n-1, n-2, \ldots, 1$, כפול את עמודה i ב־$-x_0$ והוסף אותה לעמודה $i+1$, ואז השתמש באינדוקציה.)

31.2 האלגוריתם של שטראסן לכפל מטריצות

סעיף זה מציג את האלגוריתם הרקורסיבי המפתיע של שטראסן (Strassen) עבור כפל מטריצות בגודל $n \times n$, שרץ בזמן $\Theta(n^{\lg 7}) = O(n^{2.81})$. עבור n גדול דיו, ביצועיו עולים אפוא על אלה של האלגוריתם הנאיבי לכפל מטריצות MATRIX-MULTIPLY מסעיף 26.11, שרץ בזמן $\Theta(n^3)$.

האלגוריתם בקווים כלליים

את האלגוריתם של שטראסן ניתן לראות כיישום של שיטת תכנון מוכרת: הפרד־ומשול. נניח שברצוננו לחשב את המכפלה $C = AB$, כאשר A, B ו־C הן מטריצות בגודל $n \times n$. בהנחה ש־n הוא חזקה מדויקת של 2, אנו מחלקים כל אחת מן המטריצות A, B ו־C לארבע מטריצות בגודל $n/2 \times n/2$, וכותבים מחדש את המשוואה $C = AB$ באופן הבא:

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & g \\ f & h \end{pmatrix} \tag{31.9}$$

(תרגיל 31.2-2 עוסק במצב שבו n אינו חזקה מדויקת של 2.) מטעמי נוחות, התת־מטריצות של A מסומנות בסדר אלפביתי משמאל לימין, ואילו התת־מטריצות של B מסומנות מלמעלה למטה, בהתאם לאופן שבו מתבצע כפל מטריצות. משוואה (31.9) מתאימה לארבע המשוואות:

$$r = ae + bf \tag{31.10}$$

$$s = ag + bh \tag{31.11}$$

$$t = ce + df \tag{31.12}$$

$$u = cg + dh \tag{31.13}$$

כל אחת מארבע משוואות אלה מורכבת משתי פעולות כפל של מטריצות בגודל $n/2 \times n/2$ ומפעולת חיבור אחת, של שתי מטריצות המכפלה, שגם הן מטריצות בגודל $n/2 \times n/2$. נשתמש במשוואות אלה כדי להגדיר אסטרטגיית הפרד־ומשול ישירה, ונקבל את נוסחת הנסיגה שלהלן עבור הזמן $T(n)$ לכפל שתי מטריצות בגודל $n \times n$:

$$T(n) = 8T(n/2) + \Theta(n^2) \tag{31.14}$$

לרוע המזל, פתרונה של נוסחת הנסיגה (31.14) הוא $T(n) = \Theta(n^3)$, ולכן שיטה זו איננה מהירה יותר מן השיטה הרגילה.

שטראסן (Strassen) גילה גישה רקורסיבית שונה שבה דרושות רק 7 פעולות כפל רקורסיביות של מטריצות בגודל $n/2 \times n/2$ ו־$\Theta(n^2)$ פעולות חיבור וחיסור סקלריות, כך שמתקבלת נוסחת הנסיגה:

$$\begin{aligned} T(n) &= 7T(n/2) + \Theta(n^2) \\ &= \Theta(n^{\lg 7}) \\ &= O(n^{2.81}) \end{aligned} \tag{31.15}$$

שיטת שטראסן מורכבת מארבעה צעדים:

1. חלק את מטריצות הקלט A ו־B לתת־מטריצות בגודל $n/2 \times n/2$, כמו במשוואה (31.9).

2. באמצעות $\Theta(n^2)$ פעולות חיבור וחיסור, חשב 14 מטריצות בגודל $n/2 \times n/2$: $A_1, B_1, A_2, B_2, \ldots, A_7, B_7$.
3. חשב באופן רקורסיבי את שבע מכפלות המטריצות $P_i = A_i B_i$ עבור $i = 1, 2, \ldots, 7$.
4. חשב את התת־מטריצות המבוקשות r, s, t ו־u של מטריצת התוצאה C על־ידי חיבור ו/או חיסור של צירופים שונים של המטריצות P_i, באמצעות $\Theta(n^2)$ חיבורים וחיסורים סקלריים בלבד.

תהליך כזה מקיים את נוסחת הנסיגה (31.15). כל שעלינו לעשות עתה הוא למלא את הפרטים החסרים.

חישוב המכפלות של התת־מטריצות

אין זה ברור בדיוק כיצד גילה שטראסן את אותן מכפלות התת־מטריצות שהן המפתח לכך שהאלגוריתם שלו יפעל כהלכה. להלן נשחזר שיטת גילוי אפשרית אחת.

הבה ננחש שניתן לכתוב כל מטריצת מכפלה P_i בצורה:

$$\begin{aligned} P_i &= A_i B_i \\ &= (\alpha_{i1}a + \alpha_{i2}b + \alpha_{i3}c + \alpha_{i4}d) \cdot (\beta_{i1}e + \beta_{i2}f + \beta_{i3}g + \beta_{i4}h) \end{aligned} \tag{31.16}$$

כאשר המקדמים α_{ij} ו־β_{ij} לקוחים כולם מן הקבוצה $\{-1, 0, 1\}$. כלומר, אנו מנחשים שכל מכפלה מחושבת על־ידי חיבור או חיסור חלק מהתת־מטריצות של A, חיבור או חיסור חלק מהתת־מטריצות של B, ואז כפל שתי התוצאות זו בזו. אמנם קיימות אסטרטגיות כלליות יותר, אך נראה כי אסטרטגיה פשוטה זו פועלת כהלכה.

אם ניצור את כל המכפלות שלנו באופן זה, נוכל להשתמש בשיטה זו באופן רקורסיבי בלי להניח קומוטטיביות של הכפל, משום שבכל מכפלה, כל התת־מטריצות של A מופיעות משמאל וכל התת־מטריצות של B מופיעות מימין. תכונה זו חיונית לפעולה רקורסיבית של השיטה, שכן כפל מטריצות אינו קומוטטיבי.

מטעמי נוחות, נשתמש במטריצות בגודל 4×4 כדי להציג צירופים לינאריים של מכפלות של תת־מטריצות, כאשר בכל מכפלה מופיע צירוף של תת־מטריצה אחת של A ותת־מטריצה אחת של B, כמו במשוואה (31.16). לדוגמה, ניתן לכתוב מחדש את משוואה (31.10) בצורה:

$$\begin{aligned} r &= ae + bf \\ &= \begin{pmatrix} a & b & c & d \end{pmatrix} \begin{pmatrix} +1 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix} \\ &= \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} e & f & g & h \end{array} \\ \begin{pmatrix} + & \cdot & \cdot & \cdot \\ \cdot & + & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \end{array} \end{aligned}$$

בביטוי האחרון מופיע סימון מקוצר שלפיו "+" מייצג 1+, "·" מייצג 0 ו־"−" מייצג 1−. (מכאן ואילך נשמיט את סימוני השורות והעמודות.) תוך שימוש בסימונים אלה, אנו מקבלים את המשוואות שלהלן עבור התת־מטריצות האחרות של מטריצת התוצאה C:

$$s = ag + bh$$
$$= \begin{pmatrix} \cdot & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

$$t = ce + df$$
$$= \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \\ \cdot & + & \cdot & \cdot \end{pmatrix}$$

$$u = cg + dh$$
$$= \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & + \end{pmatrix}$$

אנו פותחים את החיפוש אחר אלגוריתם מהיר יותר לכפל מטריצות בהבחנה שאת התת־מטריצה s ניתן לחשב על־ידי $s = P_1 + P_2$, כאשר כל אחת מהמטריצות P_1 ו־P_2 מחושבת באמצעות פעולת כפל מטריצות אחת בלבד:

$$P_1 = A_1B_1$$
$$= a \cdot (g - h)$$
$$= ag - ah$$
$$= \begin{pmatrix} \cdot & \cdot & + & - \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

$$P_2 = A_2B_2$$
$$= (a + b) \cdot h$$
$$= ah + bh$$
$$= \begin{pmatrix} \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

את המטריצה t ניתן לחשב באופן דומה על־ידי $t = P_3 + P_4$, כאשר:

$$P_3 = A_3B_3$$
$$= (c + d) \cdot e$$
$$= ce + de$$
$$= \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \end{pmatrix}$$

וכן:

$$P_4 = A_4B_4$$
$$= d \cdot (f - e)$$
$$= df - de$$
$$= \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ - & + & \cdot & \cdot \end{pmatrix}$$

הבה נגדיר **ביטוי חיוני** (essential term) כאחד משמונת הביטויים המופיעים באגפי ימין של המשוואות (31.10)-(31.13). עד כה השתמשנו בארבע מכפלות לחישוב שתי התת־מטריצות s ו־t, שהביטויים החיוניים שלהן הם ag, bh, ce ו־df. נשים לב כי במכפלה P_1 מחשבים את הביטוי החיוני ag, במכפלה P_2 מחשבים את הביטוי החיוני bh, במכפלה P_3 מחשבים את הביטוי החיוני ce, ובמכפלה P_4 מחשבים את הביטוי החיוני df. לכן עלינו לחשב עוד את שתי התת־מטריצות הנותרות r ו־u שהביטויים החיוניים שלהן הם הביטויים האלכסוניים ae, bf, cg ו־dh, וזאת באמצעות לא יותר מ־3 מכפלות נוספות. אנו ננסה עתה את החידוש P_5 כדי לחשב בבת־אחת שני ביטויים חיוניים:

$$P_5 = A_5B_5$$
$$= (a + d) \cdot (e + h)$$
$$= ae + ah + de + dh$$
$$= \begin{pmatrix} + & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & + \end{pmatrix}$$

בנוסף לחישוב שני הביטויים החיוניים ae ו־dh, מחשבים במכפלה P_5 גם את הביטויים ah ו־de שאינם חיוניים, שאותם יש לבטל באיזושהי דרך. נוכל לבטלם באמצעות P_4 ו־P_2, אבל אז יופיעו שני ביטויים אחרים שאינם חיוניים:

$$P_5 + P_4 - P_2 = ae + dh + df - bh$$
$$= \begin{pmatrix} + & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & - \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & + & \cdot & + \end{pmatrix}$$

אולם, על־ידי הוספת המכפלה:

$$\begin{aligned} P_6 &= A_6 B_6 \\ &= (b-d)\cdot(f+h) \\ &= bf + bh - df - dh \end{aligned}$$

$$= \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & + & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & - & \cdot & - \end{pmatrix}$$

אנו מקבלים:

$$\begin{aligned} r &= P_5 + P_4 - P_2 + P_6 \\ &= ae + bf \end{aligned}$$

$$= \begin{pmatrix} + & \cdot & \cdot & \cdot \\ \cdot & + & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

נוכל לקבל את u מ־P_5 באופן דומה, תוך שימוש ב־P_1 ו־P_3 כדי להזיז את הביטויים שאינם חיוניים של P_5 בכיוון שונה:

$$P_5 + P_1 - P_3 = ae + ag - ce + dh$$

$$= \begin{pmatrix} + & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ - & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & + \end{pmatrix}$$

על־ידי חיסור מכפלה נוספת:

$$\begin{aligned} P_7 &= A_7 B_7 \\ &= (a-c)\cdot(e+g) \\ &= ae + ag - ce - cg \end{aligned}$$

$$= \begin{pmatrix} + & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ - & \cdot & - & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

נקבל:

$$\begin{aligned} u &= P_5 + P_1 - P_3 - P_7 \\ &= cg + dh \end{aligned}$$

$$= \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & + \end{pmatrix}$$

ניתן אפוא לחשב את המכפלה $C = AB$ באמצעות שבע המכפלות של תתי־מטריצות $P_1, P_2, \ldots, P_7$. בכך נשלם תיאורה של שיטת שטראסן.

דיון

בשל הקבוע הגדול החבוי בזמן הריצה שלו, האלגוריתם של שטראסן אינו מעשי אלא אם כן מדובר במטריצות גדולות (n בסביבות 45 לפחות) וצפופות (מעט איברים השווים לאפס). עבור מטריצות קטנות, האלגוריתם הישיר עדיף, ועבור מטריצות גדולות ודלילות, קיימים אלגוריתמים מיוחדים שביצועיהם ביישומים מעשיים טובים מאלה של האלגוריתם של שטראסן. לכן, העניין בשיטת שטראסן הוא תיאורטי בעיקרו.

באמצעות שיטות מתקדמות, החורגות מתחומו של ספר זה, ניתן למעשה לכפול מטריצות בגודל $n \times n$ בזמן קצר יותר מ־$\Theta(n^{\lg 7})$. החסם העליון הטוב ביותר המוכר כיום הוא $O(n^{2.376})$ בקירוב. החסם התחתון הטוב ביותר המוכר כיום הוא החסם המובן מאליו $\Omega(n^2)$ (מובן מאליו מכיוון שעלינו למלא בערכים n^2 תאים במטריצת המכפלה). אם כן, כיום איננו יודעים מהי מידת הקושי האמיתית של כפל מטריצות.

האלגוריתם של שטראסן אינו דורש שאיברי המטריצות יהיו מספרים ממשיים. די בכך שמערכת המספרים תהיה חוג אלגברי. אולם, אם איברי המטריצה אינם יוצרים חוג, ניתן לעתים להפעיל שיטות אחרות כך שיתאפשר יישומה של שיטת שטראסן. נושאים אלה יידונו ביתר הרחבה בסעיף הבא.

תרגילים

31.2-1

חשב באמצעות האלגוריתם של שטראסן את המכפלה:

$$\begin{pmatrix} 1 & 3 \\ 5 & 7 \end{pmatrix}\begin{pmatrix} 8 & 4 \\ 6 & 2 \end{pmatrix}$$

הצג את שלבי עבודתך.

31.2-2

כיצד תשנה את האלגוריתם של שטראסן כדי לכפול מטריצות בגודל $n \times n$ כאשר n אינו חזקה מדויקת של 2? הראה שהאלגוריתם המתקבל רץ בזמן $\Theta(n^{\lg 7})$.

31.2-3

מהו ה־k הגדול ביותר כך שאם ניתן לכפול מטריצות בגודל 3×3 תוך שימוש ב־k פעולות כפל (בלי להניח קומוטטיביות של הכפל), אזי ניתן לכפול מטריצות בגודל $n \times n$ בזמן $o(n^{\lg 7})$? מה יהיה זמן הריצה של אלגוריתם זה?

31.2-4

V. Pan גילה דרך לכפול מטריצות בגודל 68×68 תוך שימוש ב־132,464 פעולות כפל, דרך לכפול מטריצות בגודל 70×70 תוך שימוש ב־143,640 פעולות כפל, ודרך לכפול מטריצות בגודל 72×72 תוך שימוש ב־155,424 פעולות כפל. איזו שיטה מניבה את זמן הריצה

האסימפטוטי הטוב ביותר כאשר משתמשים בה באלגוריתם הפרד ומשול לכפל מטריצות? השווה עם זמן הריצה של האלגוריתם של שטראסן.

31.2-5

באיזו מהירות תוכל לכפול מטריצה בגודל $kn \times n$ במטריצה בגודל $n \times kn$ תוך שימוש באלגוריתם של שטראסן כשגרה? ענה על אותה שאלה כאשר סדר מטריצות הקלט הפוך.

31.2-6

הראה כיצד ניתן לכפול את המספרים המרוכבים $a + bi$ ו־$c + di$ תוך שימוש בשלוש פעולות כפל ממשיות בלבד. הקלט לאלגוריתם צריך להיות a, b, c ו־d, והאלגוריתם צריך לחשב בנפרד את הרכיב הממשי $ac - bd$ ואת הרכיב המדומה $ad + bc$.

* 31.3 מערכות מספרים אלגבריות וכפל מטריצות בוליאניות

תכונותיהן של פעולות כפל וחיבור של מטריצות תלויות בתכונותיה של מערכת המספרים שממנה לקוחים איברי המטריצות. בסעיף זה נגדיר שלושה סוגים שונים של מערכות מספרים כאלה: מעין־חוגים, חוגים ושדות. ניתן להגדיר כפל מטריצות מעל מעין־חוגים, והאלגוריתם של שטראסן לכפל מטריצות פועל מעל חוגים. לאחר מכן נציג תכסיס פשוט שבאמצעותו ניתן לעשות רדוקציה של כפל מטריצות בוליאניות המוגדר מעל מעין־חוג שאינו חוג, לכפל מעל חוג. לבסוף, נדון בשאלה מדוע לא ניתן לנצל באופן טבעי את תכונותיו של שדה כדי לקבל אלגוריתמים טובים יותר לכפל מטריצות.

מעין־חוגים

תהי $(S, \oplus, \odot, \bar{0}, \bar{1})$ מערכת מספרים, כאשר S היא קבוצת איברים, $\oplus$ ו־$\odot$ הן פעולות בינריות על S (פעולות החיבור והכפל, בהתאמה), ו־$\bar{0}$ ו־$\bar{1}$ הם איברים מיוחדים, שונים זה מזה, ב־S. מערכת זו היא **מעין־חוג** (quasiring) אם היא מקיימת את התכונות הבאות:

1. $(S, \oplus, \bar{0})$ הוא **מונואיד** (monoid):
   - S **סגורה** (closed) תחת $\oplus$; כלומר, $a \oplus b \in S$ עבור כל $a, b \in S$.
   - $\oplus$ **אסוציאטיבי** (associative); כלומר, $a \oplus (b \oplus c) = (a \oplus b) \oplus c$ עבור כל $a, b, c \in S$.
   - $\bar{0}$ הוא **איבר היחידה** (identity) עבור $\oplus$; כלומר, $a \oplus \bar{0} = \bar{0} \oplus a = a$ עבור כל $a \in S$.

   באופן דומה, $(S, \odot, \bar{1})$ הוא מונואיד.
2. $\bar{0}$ הוא איבר **מאפס** (annihilator); כלומר, $a \odot \bar{0} = \bar{0} \odot a = \bar{0}$ עבור כל $a \in S$.
3. האופרטור $\oplus$ הוא **קומוטטיבי** (commutative); כלומר, $a \oplus b = b \oplus a$, עבור כל $a, b \in S$.

4. האופרטור $\odot$ **מתפלג** (distributes) מעל $\oplus$; כלומר:

   $a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c)$ וכן $(b \oplus c) \odot a = (b \odot a) \oplus (c \odot A)$

   עבור כל $a, b, c \in S$.

דוגמאות למעין־חוגים כוללות את **המעין־חוג הבוליאני** (boolean quasiring) $(\{0, 1\}, \vee, \wedge, 0, 1)$ כאשר $\vee$ מציין OR לוגי ו־$\wedge$ מציין AND לוגי, ואת מערכת המספרים הטבעיים $(\mathbf{N}, +, \cdot, 0, 1)$, כאשר $+$ ו־$\cdot$ מציינים חיבור וכפל רגילים. כל חוג־למחצה סגור (ראה סעיף 26.4) הוא גם מעין־חוג. חוגים־למחצה סגורים מקיימים נוסף לכך גם אידמפוטנטיות ותכונות הנוגעות לסכומים אינסופיים.

ניתן להרחיב את הגדרת האופרטורים $\oplus$ ו־$\odot$ למטריצות, כפי שעשינו עבור $+$ ו־$\cdot$ בסעיף 31.1. אם נסמן ב־$\bar{I}_n$ את מטריצת היחידה בגודל $n \times n$ המורכבת מהאיברים $\bar{0}$ ו־$\bar{1}$, נראה שכפל מטריצות מוגדר היטב ושמערכת המטריצות היא עצמה מעין־חוג, כפי שמראה המשפט הבא.

משפט 31.7 (מטריצות מעל מעין־חוג יוצרות מעין־חוג)
אם $(S, \oplus, \odot, \bar{0}, \bar{1})$ הוא מעין־חוג ו־$n \geq 1$, אזי $(S^{n \times n}, \oplus, \odot, \bar{0}, \bar{I}_n)$ הוא מעין־חוג.

הוכחה הקורא יתבקש להוכיח את המשפט בתרגיל 31.3-3. ■

חוגים

חיסור אינו מוגדר במעין־חוגים, אולם הוא מוגדר בחוגים. **חוג** (ring) הוא מעין־חוג $(S, \oplus, \odot, \bar{0}, \bar{1})$ המקיים גם את התכונה הנוספת הבאה:

5. לכל איבר ב־S יש **הופכי חיבורי** (additive inverse); כלומר, לכל $a \in S$ קיים איבר $b \in S$ כך ש־$a \oplus b = b \oplus a = \bar{0}$. כזה מכונה גם **השלילי** (negative) של a ומסומן על־ידי $(-a)$.

אם השלילי של כל איבר מוגדר, ניתן להגדיר חיסור על־ידי $a - b = a + (-b)$.

קיימות דוגמאות רבות לחוגים. השלמים $(\mathbf{Z}, +, \cdot, 0, 1)$ תחת פעולות החיבור והכפל הרגילות יוצרים חוג. השלמים מודולו n עבור שלם כלשהו $n > 1$, כלומר, $(\mathbf{Z}_n, +, \cdot, 0, 1)$, כאשר $+$ הוא חיבור מודולו n ו־$\cdot$ הוא כפל מודולו n, יוצרים חוג. דוגמה נוספת היא הקבוצה $\mathbf{R}[x]$, קבוצת הפולינומים ב־x ממעלה סופית בעלי מקדמים ממשיים תחת הפעולות הרגילות — כלומר, $(\mathbf{R}[x], +, \cdot, 0, 1)$, כאשר $+$ מציין חיבור פולינומים ו־$\cdot$ מציין כפל פולינומים.

המסקנה הבאה מראה שניתן להכליל את משפט 31.7 לחוגים באופן טבעי.

מסקנה 31.8 (מטריצות מעל חוג יוצרות חוג)
אם $(S, \oplus, \odot, \bar{0}, \bar{1})$ הוא חוג ו־$n \geq 1$, אזי $(S^{n \times n}, \oplus, \odot, \bar{0}, \bar{I}_n)$ הוא חוג.

הוכחה הקורא יתבקש להוכיח את המסקנה בתרגיל 31.3-3. ■

בעזרת מסקנה זו ניתן להוכיח את המשפט הבא:

משפט 31.9
האלגוריתם של שטראסן לכפל מטריצות פועל כהלכה מעל כל חוג (כלומר, איברי המטריצות משתייכים לחוג).

הוכחה נכונותו של האלגוריתם של שטראסן תלויה בנכונותו של האלגוריתם עבור מטריצות בגודל 2×2, הדורש רק שאיברי המטריצות ישתייכו לחוג. מכיוון שאיברי המטריצות אכן שייכים לחוג, מסקנה 31.8 גוררת שהמטריצות עצמן יוצרות חוג. לכן, באינדוקציה, האלגוריתם של שטראסן פועל כהלכה בכל שלב של הרקורסיה. ■

למעשה, האלגוריתם של שטראסן לכפל מטריצות תלוי באופן מכריע בקיומם של איברים הופכיים חיבוריים. מבין שבע המכפלות $P_1, P_2, \ldots, P_7$, ארבע כוללות הפרשים של תת־מטריצות. אם כן, ככלל, האלגוריתם של שטראסן אינו פועל עבור מעין־חוגים.

כפל מטריצות בוליאניות

לא ניתן להשתמש ישירות באלגוריתם של שטראסן לכפל מטריצות בוליאניות, מכיוון שהמעין־חוג הבוליאני $(\{0, 1\}, \vee, \wedge, 0, 1)$ אינו חוג. קיימים מקרים שבהם מעין־חוג מוכל במערכת גדולה יותר המהווה חוג. לדוגמה, המספרים הטבעיים (מעין־חוג) הם תת־קבוצה של השלמים (חוג), ולכן ניתן להשתמש באלגוריתם של שטראסן לכפל מטריצות של מספרים טבעיים אם מאמצים את השלמים כמערכת שממנה לקוחים איברי המטריצה. לרוע המזל, לא ניתן להרחיב לחוג באופן דומה את המעין־חוג הבוליאני. (ראה תרגיל 31.3-4.)

המשפט הבא מציג תכסיס פשוט שבאמצעותו ניתן לעשות רדוקציה של כפל מטריצות בוליאניות לכפל מטריצות מעל חוג. בעיה 31-1 מציגה גישה יעילה נוספת.

משפט 31.10
אם $M(n)$ מציין את מספר הפעולות האריתמטיות הדרושות לכפל שתי מטריצות בגודל $n \times n$ מעל השלמים, אזי ניתן לכפול שתי מטריצות בוליאניות בגודל $n \times n$ באמצעות $O(M(n))$ פעולות אריתמטיות.

הוכחה יהיו A ו־B שתי המטריצות, ותהי $C = AB$ במעין־חוג הבוליאני, כלומר:

$$c_{ij} = \bigvee_{k=1}^{n} a_{ik} \wedge b_{kj}$$

במקום לחשב מעל המעין־חוג הבוליאני, נחשב את המכפלה C' מעל חוג השלמים באמצעות האלגוריתם לכפל מטריצות המבצע $M(n)$ פעולות אריתמטיות. נקבל אפוא:

$$c'_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

כל איבר $a_{ik}b_{kj}$ בסכום הזה שווה ל־0 אם ורק אם $a_{ik} \wedge b_{kj} = 0$, ושווה ל־1 אם ורק אם $a_{ik} \wedge b_{kj} = 1$. לכן, הסכום השלם c'_{ij} שווה ל־0 אם ורק אם כל איבר שווה ל־0, או באופן שקול, אם ורק אם ה־OR הבוליאני של האיברים, שהוא c_{ij}, שווה ל־0. לכן, ניתן לשחזר את המטריצה C ממטריצת השלמים C' באמצעות $\Theta(n^2)$ פעולות אריתמטיות, פשוט על־ידי כך שמשווים כל c'_{ij} עם אפס. מספר הפעולות האריתמטיות לביצוע התהליך כולו הוא אפוא $O(M(n)) + \Theta(n^2) = O(M(n))$, שכן $M(n) + \Omega(n^2)$. ■

לכן, באמצעות האלגוריתם של שטראסן ניתן לבצע כפל מטריצות בוליאניות בזמן $\Theta(n^{\lg 7})$.

בשיטה הרגילה לכפל מטריצות בוליאניות, כל המשתנים הם בוליאניים. אולם אם משתמשים בגרסה זו של אלגוריתם שטראסן, מטריצת המכפלה הסופית עשויה להכיל איברים שערכם יכול להגיע עד n, ולכן לשם אחסונם תידרש מילת זיכרון ולא סיבית יחידה. מטרידה עוד יותר היא העובדה שתוצאות הביניים, שהן מספרים שלמים, עשויות להגיע אף לערכים גדולים יותר. אחת הדרכים שבהן ניתן לגרום לכך שערכיהן של תוצאות הביניים לא יגדלו יותר מדי היא לבצע את כל החישובים מודולו $n + 1$. בתרגיל 31.3-5 תתבקש להראות שהחישוב מודולו $n + 1$ אינו משפיע על נכונות האלגוריתם.

שדות

חוג $(S, \oplus, \odot, \bar{0}, \bar{1})$ הוא **שדה** (field) אם הוא מקיים גם את שתי התכונות הנוספות הבאות:

6. האופרטור $\odot$ הוא **קומוטטיבי** (commutative); כלומר, מתקיים $a \odot b = b \odot a$ עבור כל $a, b \in S$.
7. לכל איבר שונה מאפס ב־S קיים **הופכי כפלי** (multiplicative inverse); כלומר, לכל $a \in S - \{\bar{0}\}$ קיים איבר $b \in S$ כך ש־$a \odot b = b \odot a = \bar{1}$. איבר b כזה מכונה לעתים **ההופכי** (inverse) של a ומסומן על־ידי a^{-1}.

דוגמאות של שדות כוללות את המספרים הממשיים $(\mathbf{R}, +, \cdot, 0, 1)$, המספרים המרוכבים $(\mathbf{C}, +, \cdot, 0, 1)$ והמספרים השלמים מודולו מספר ראשוני p: $(\mathbf{Z}_p, +, \cdot, 0, 1)$.

מכיוון ששדות כוללים הופכיים כפליים של איבריהם, ניתן לבצע בהם חילוק. שדות מציעים גם קומוטטיביות. על־ידי הכללה ממעין־חוגים לחוגים, הצליח שטראסן לשפר את זמן הריצה עבור כפל מטריצות. מאחר שאיברי מטריצות לקוחים לעתים קרובות משדה — המספרים הממשיים, למשל — ניתן היה לקוות ששימוש בשדות במקום חוגים באלגוריתם רקורסיבי דמוי־שטראסן יביא לשיפור נוסף בזמן הריצה.

אולם, אין זה נראה סביר שגישה זו תניב את התוצאות המקוות. כדי שאלגוריתם הפרד־ומשול רקורסיבי המבוסס על שדות יפעל היטב, המטריצות בכל שלב של הרקורסיה חייבות ליצור שדה. לרוע המזל, ההרחבה הטבעית של משפט 31.7 ושל מסקנה 31.8 לשדות נכשלת כישלון חרוץ. עבור $n > 1$, קבוצת המטריצות בגודל $n \times n$ *לעולם* אינה יוצרת שדה, גם אם איברי המטריצות לקוחים משדה. כפל מטריצות בגודל $n \times n$ אינו קומוטטיבי, ומטריצות רבות בגודל $n \times n$ אין להן מטריצות הופכיות. סביר יותר אפוא שאלגוריתמים משופרים לכפל מטריצות יתבססו על תורת החוגים ולא על תורת השדות.

תרגילים

31.3-1 *

האם האלגוריתם של שטראסן פועל כהלכה מעל מערכת המספרים $(\mathbf{Z}[x], +, \cdot, 0, 1)$, כאשר $\mathbf{Z}[x]$ היא קבוצת כל הפולינומים במשתנה x שמקדמיהם שלמים, ו־$+$ ו־$\cdot$ הן הפעולות הרגילות של חיבור וכפל פולינומים?

31.3-2 *

הסבר מדוע האלגוריתם של שטראסן אינו פועל מעל חוגים־למחצה סגורים (ראה סעיף 26.4) או מעל המעין־חוג הבוליאני $(\{0, 1\}, \vee, \wedge, 0, 1)$.

31.3-3 *

הוכח את משפט 31.7 ואת מסקנה 31.8.

31.3-4 *

הראה שהמעין־חוג הבוליאני $(\{0, 1\}, \vee, \wedge, 0, 1)$ אינו יכול להיות מוכל בחוג. כלומר, הראה שאין זה אפשרי להוסיף למעין־חוג זה "-1" כך שהמבנה האלגברי המתקבל יהיה חוג.

31.3-5

הראה שאם מבצעים את כל החישובים באלגוריתם ממשפט 31.10 מודולו $n + 1$, האלגוריתם עדיין פועל כהלכה.

31.3-6

הראה כיצד ניתן לקבוע ביעילות אם גרף קלט בלתי־מכוון נתון מכיל משולש (קבוצה של שלושה קדקודים שכולם סמוכים זה לזה).

31.3-7 *

הראה שחישוב המכפלה של שתי מטריצות בוליאניות $n \times n$ מעל המעין־חוג הבוליאני ניתן לרדוקציה לחישוב הסגור הטרנזיטיבי של גרף מכוון נתון בן $3n$ קדקודים.

31.3-8

הראה כיצד ניתן לחשב את הסגור הטרנזיטיבי של גרף מכוון נתון בן n קדקודים בזמן $\Theta(n^{\lg 7} \lg n)$. השוואה תוצאה זו עם הביצועים של השגרה TRANSITIVE-CLOSURE מסעיף 26.2.

31.4 פתרון מערכת משוואות לינאריות

פתרון מערכת של משוואות לינאריות הוא בעיה בסיסית המופיעה ביישומים רבים ושונים. ניתן לכתוב מערכת לינארית כמשוואת מטריצות שבה כל איבר של מטריצה או של וקטור שייך לשדה, בדרך כלל לשדה המספרים הממשיים $\mathbf{R}$. בסעיף זה נדון בדרכים לפתרון מערכת משוואות לינאריות באמצעות שיטה המכונה "פירוק LUP".

נתונה לנו מערכת משוואות לינאריות ב־n נעלמים $x_1, x_2, \ldots, x_n$:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \tag{31.17}$$

קבוצה של ערכים עבור $x_1, x_2, \ldots, x_n$ המספקים בו־בזמן את כל המשוואות (31.17) נקראת **פתרון** של מערכת המשוואות. בסעיף זה נעסוק רק במקרה שבו נתונות בדיוק n משוואות ב־n נעלמים.

את מערכת המשוואות (31.17) ניתן לכתוב מחדש בכתיב מטריצות באופן הבא:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

או, אם נסמן $A = (a_{ij})$, $x = (x_j)$ ו־$b = (b_i)$, נוכל לכתוב זאת גם כך:

$$Ax = b \tag{31.18}$$

אם A היא רגולרית, יש לה מטריצה הופכית A^{-1}, ואז הווקטור:

$$x = A^{-1}b \tag{31.19}$$

הוא וקטור הפתרון. ניתן להוכיח ש־x הוא פתרון יחיד של המשוואה (31.18) כדלקמן. אם קיימים שני פתרונות, x ו־x', אזי $Ax = Ax' = b$ ולפיכך:

$$\begin{aligned} x &= (A^{-1}A)x \\ &= A^{-1}(Ax) \\ &= A^{-1}(Ax') \\ &= (A^{-1}A)x' \\ &= x' \end{aligned}$$

בסעיף זה נתעניין בעיקר במקרה שבו A היא רגולרית, או, באופן שקול (על פי משפט 31.1), דרגתה של A שווה למספר הנעלמים n. עם זאת, קיימות אפשרויות אחרות הראויות לדיון קצר. אם מספר המשוואות קטן ממספר הנעלמים n — או, באופן כללי יותר, אם דרגתה של A קטנה מ־n — אזי המערכת היא **תת־מוגדרת** (underdetermined). למערכת תת־מוגדרת יש בדרך כלל אינסוף פתרונות (ראה תרגיל 31.4-9), אם כי ייתכן שאין לה פתרונות כלל. אם מספר המשוואות עולה על מספר הנעלמים n, המערכת היא **מוגדרת־יתר** (overdetermined) וייתכן שאין לה פתרונות כלל. מציאת פתרונות מקורבים טובים למערכות משוואות לינאריות מוגדרות־יתר היא בעיה חשובה שתידון בסעיף 31.6.

הבה נחזור לבעיה של פתרון מערכת $Ax = b$ של n משוואות ב־n נעלמים. אחת הגישות היא לחשב את A^{-1} ואז לכפול את שני האגפים ב־A^{-1} ולקבל $A^{-1}Ax = A^{-1}b$ או $x = A^{-1}b$. בפועל סובלת גישה זו מ**אי־יציבות נומרית** (numerical instability): כאשר המספרים מיוצגים בשיטת הנקודה הצפה ולא כמספרים ממשיים אידאליים, הטעויות הנובעות מעיגול נוטות להצטבר יותר מדי. למרבה המזל, קיימת גישה אחרת — פירוק LUP — שהיא יציבה מבחינה נומרית ויש לה יתרון נוסף: היא מהירה פי 3 בערך מן הגישה הישירה.

פירוק LUP: סקירה

הרעיון העומד מאחורי פירוק LUP הוא למצוא שלוש מטריצות בגודל $n \times n$: L, U ו־P, המקיימות:

$$PA = LU \tag{31.20}$$

כאשר:

- L היא מטריצת יחידה משולשית־תחתונה,
- U היא מטריצה משולשית־עליונה,
- P היא מטריצת תמורה.

מטריצות L, U ו־P המקיימות את משוואה (31.20) נקראות **פירוק LUP** (LUP decomposition) של המטריצה A. אנו נראה שלכל מטריצה רגולרית A יש פירוק כזה.

היתרון בחישוב פירוק LUP של המטריצה A טמון בכך שקל יותר לפתור מערכות לינאריות שהן משולשיות, כמו במקרה של המטריצות L ו־U. לאחר שמוצאים פירוק LUP עבור A, ניתן לפתור את המשוואה $Ax = b$ (31.18) על־ידי כך שפותרים רק מערכות לינאריות משולשיות, כמתואר להלן. כופלים ב־P את שני האגפים של $Ax = b$ ומקבלים את המשוואה השקולה $PAx = Pb$, אשר, על פי תרגיל 31.1-2, אינה אלא תמורה של המשוואות (31.17). מציבים את הפירוק (31.20) ומקבלים:

$$LUx = Pb$$

את המשוואה הזאת ניתן עתה לפתור על־ידי כך שפותרים שתי מערכות לינאריות משולשיות. נגדיר $y = Ux$, כאשר x הוא וקטור הפתרון המבוקש. תחילה נפתור את המערכת המשולשית התחתונה:

$$Ly = Pb \tag{31.21}$$

עבור הווקטור הנעלם y, באמצעות שיטה המכונה "הצבה־קדימה". לאחר שמצאנו את הווקטור y, נפתור את המערכת המשולשית העליונה:

$$Ux = y \tag{31.22}$$

עבור הווקטור הנעלם x, באמצעות שיטה המכונה "הצבה־אחורה". הווקטור x הוא הפתרון שלנו למערכת $Ax = b$, שכן מטריצת התמורה P הפיכה (תרגיל 31.1-2):

$$\begin{aligned} Ax &= P^{-1}LUx \\ &= P^{-1}Ly \\ &= P^{-1}Pb \\ &= b \end{aligned}$$

הצעד הבא שלנו הוא להראות כיצד פועלות הצבה־קדימה והצבה־אחורה, ואז נתקוף את הבעיה של חישוב פירוק LUP עצמו.

הצבה־קדימה והצבה־אחורה

באמצעות **הצבה־קדימה** (forward substitution) ניתן לפתור את המערכת המשולשית התחתונה (31.21) בזמן $\Theta(n^2)$, בהינתן L, P ו־b. מטעמי נוחות, נייצג את מטריצת התמורה P באופן

קומפקטי על־ידי מערך $\pi[1..n]$. עבור $i = 1, 2, \ldots, n$, התא $\pi[i]$ מציין כי $P_{i,\pi[i]} = 1$ ו־$P_{ij} = 0$ עבור $j \neq \pi[i]$. לכן, בשורה ה־i ובעמודה ה־j של PA נמצא האיבר $a_{\pi[i],j}$ והאיבר ה־i של Pb הוא $b_{\pi[i]}$. מכיוון ש־L היא מטריצת יחידה משולשית תחתונה, ניתן לכתוב מחדש את משוואה (31.21) בצורה:

$$\begin{aligned} y_1 &= b_{\pi[1]} \\ l_{21}y_1 + y_2 &= b_{\pi[2]} \\ l_{31}y_1 + l_{32}y_2 + y_3 &= b_{\pi[3]} \\ &\vdots \\ l_{n1}y_1 + l_{n2}y_2 + l_{n3}y_3 + \cdots + y_n &= b_{\pi[n]} \end{aligned}$$

אפשר לראות בבירור שניתן לקבל ישירות את y_1 מהמשוואה הראשונה: $y_1 = b_{\pi[1]}$. לאחר שמצאנו את y_1, נוכל להציב אותו במשוואה השנייה, ונקבל:

$$y_2 = b_{\pi[2]} - l_{21}y_1$$

עתה נוכל להציב את y_1 וגם את y_2 במשוואה השלישית, ונקבל:

$$y_3 = b_{\pi[3]} - (l_{31}y_1 + l_{32}y_2)$$

ובאופן כללי, אנו מציבים את $y_1, y_2, \ldots, y_{i-1}$ "קדימה" במשוואה ה־i כדי לקבל את y_i:

$$y_i = b_{\pi[i]} - \sum_{j=1}^{i-1} l_{ij}y_j$$

הצבה־אחורה (backward substitution) דומה להצבה־קדימה. בהינתן U ו־y, אנו פותרים תחילה את המשוואה ה־n וממשיכים אחורה עד המשוואה הראשונה. בדומה להצבה־קדימה, גם תהליך זה מתבצע בזמן $\Theta(n^2)$. מכיוון ש־U היא משולשית־עליונה, ניתן לכתוב מחדש את המערכת (31.22) בצורה:

$$\begin{aligned} u_{11}x_1 + u_{12}x_2 + \ldots + u_{1,n-2}x_{n-2} + u_{1,n-1}x_{n-1} + u_{1n}x_n &= y_1 \\ u_{22}x_2 + \ldots + u_{2,n-2}x_{n-2} + u_{2,n-1}x_{n-1} + u_{2n}x_n &= y_2 \\ &\vdots \\ u_{n-2,n-2}x_{n-2} + u_{n-2,n-1}x_{n-1} + u_{n-2,n}x_n &= y_{n-2} \\ u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n &= y_{n-1} \\ u_{n,n}x_n &= y_n \end{aligned}$$

לכן, אפשר למצוא את $x_1, \ldots, x_{n-1}, x_n$ בזה אחר זה באופן הבא:

$$\begin{aligned} x_n &= y_n/u_{n,n} \\ x_{n-1} &= (y_{n-1} - u_{n-1,n}x_n)/u_{n-1,n-1} \\ x_{n-2} &= (y_{n-2} - (u_{n-2,n-1}x_{n-1} + u_{n-2,n}x_n))/u_{n-2,n-2} \\ &\vdots \end{aligned}$$

או, במקרה הכללי:

$$x_i = \left(y_i - \sum_{j=i+1}^{n} u_{ij}x_j\right)/u_{ii}$$

בהינתן P, L, U ו־b, השגרה LUP-Solve מוצאת את x באמצעות שילוב של הצבה־קדימה והצבה־אחורה. הפסאודו־קוד מניח שהממד n מופיע במאפיין $rows[L]$ ושמטריצת התמורה P מיוצגת על־ידי המערך π.

```
LUP-Solve(L, U, π, b)
1  n ← rows[L]
2  for i ← 1 to n
3      do y_i ← b_π[i] − Σ_{j=1}^{i−1} l_ij y_j
4  for i ← n downto 1
5      do x_i ← (y_i − Σ_{j=i+1}^{n} u_ij x_j) / u_ii
6  return x
```

השגרה LUP-Solve מוצאת את y באמצעות הצבה־קדימה בשורות 2-3, ולאחר מכן מוצאת את x באמצעות הצבה־אחורה בשורות 4-5. מכיוון שקיימת לולאה מובלעת בכל אחד מן הסכומים המופיעים בתוך לולאות ה־**for**, זמן הריצה הוא $\Theta(n^2)$.

כדוגמה לשימוש בשיטות אלה, נתבונן במערכת המשוואות הלינאריות המוגדרת על ידי:

$$\begin{pmatrix} 1 & 2 & 0 \\ 3 & 5 & 4 \\ 5 & 6 & 3 \end{pmatrix} x = \begin{pmatrix} 0.1 \\ 12.5 \\ 10.3 \end{pmatrix}$$

כאשר:

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 3 & 5 & 4 \\ 5 & 6 & 3 \end{pmatrix}$$

$$b = \begin{pmatrix} 0.1 \\ 12.5 \\ 10.3 \end{pmatrix}$$

ברצוננו לפתור את המערכת עבור x. פירוק LUP של המערכת הוא:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0.6 & 1 & 0 \\ 0.2 & 0.571 & 1 \end{pmatrix}$$

$$U = \begin{pmatrix} 5 & 6 & 3 \\ 0 & 1.4 & 2.2 \\ 0 & 0 & -1.856 \end{pmatrix}$$

$$P = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

(הקורא מוזמן לוודא כי $PA = LU$.) נפתור את המשוואה $Ly = Pb$ עבור y תוך שימוש בהצבה קדימה:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0.6 & 1 & 0 \\ 0.2 & 0.571 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 10.3 \\ 12.5 \\ 0.1 \end{pmatrix}$$

תחילה מוצאים את y_1, לאחר מכן את y_2, ולבסוף את y_3. מקבלים:

$$y = \begin{pmatrix} 10.3 \\ 6.32 \\ -5.569 \end{pmatrix}$$

את המשוואה $Ux = y$ נפתור עבור x תוך שימוש בהצבה אחורה:

$$\begin{pmatrix} 5 & 6 & 3 \\ 0 & 1.4 & 2.2 \\ 0 & 0 & -1.856 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 10.3 \\ 6.32 \\ -5.569 \end{pmatrix}$$

תחילה מוצאים את x_3, לאחר מכן את x_2, ולבסוף את x_1. בכך מתקבלת התשובה המבוקשת:

$$x = \begin{pmatrix} 0.5 \\ -0.2 \\ 3.0 \end{pmatrix}$$

חישוב פירוק LU

עד כה הראינו שאם ניתן לחשב פירוק LUP עבור מטריצה רגולרית A, ניתן להשתמש אז בהצבות קדימה ואחורה כדי לפתור את מערכת המשוואות הלינאריות $Ax = b$. נותר להראות כיצד ניתן למצוא ביעילות פירוק LUP של A. נתחיל במקרה שבו A היא מטריצה רגולרית בגודל $n \times n$ ו־P נעדרת (או, באופן שקול, $P = I_n$). במקרה זה עלינו למצוא פירוק לגורמים $A = LU$. שתי המטריצות L ו־U נקראות **פירוק LU** (LU decomposition) של A.

התהליך שבאמצעותו מבצעים פירוק LU של A נקרא **חילוץ גאוסיאני** (Gaussian elimination). מתחילים בכך שמחסרים כפולות של המשוואה הראשונה מן המשוואות האחרות כך שהמשתנה הראשון שוב אינו מופיע במשוואות אלה. לאחר מכן מחסרים כפולות של המשוואה השנייה מן המשוואה השלישית ומן המשוואות הבאות אחריה כך שהמשתנה הראשון והמשתנה השני שוב אינם מופיעים בהן. ממשיכים בתהליך זה עד שמתקבלת מערכת משולשית־עליונה — למעשה, זוהי המטריצה U. המטריצה L מורכבת מכופלי השורות הגורמים למשתנים להיעלם.

האלגוריתם שלנו למימוש אסטרטגיה זו הוא רקורסיבי. ברצוננו לבנות פירוק LU של מטריצה רגולרית A בגודל $n \times n$. אם $n = 1$, סיימנו, שכן ניתן לבחור $L = I_1$ ו־$U = A$. עבור $n > 1$, נפרק את A לארבעה חלקים:

$$A = \left(\begin{array}{c|ccc} a_{11} & a_{12} & \cdots & a_{1n} \\ \hline a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{array}\right)$$

$$= \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ v & A' \end{pmatrix}$$

כאשר v הוא וקטור עמודה בגודל $(n-1)$, w^{T} הוא וקטור שורה בגודל $(n-1)$, ו־A' היא מטריצה בגודל $(n-1)\times(n-1)$. לאחר מכן, באמצעות אלגברת מטריצות, נפרק את A לגורמים באופן הבא:

$$A = \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ v & A' \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ 0 & A' - vw^{\mathrm{T}}/a_{11} \end{pmatrix}$$

(הקורא מוזמן לוודא את נכונות המשוואות על ידי ביצוע ישיר של הכפל.) האפסים במטריצה הראשונה והשנייה של הפירוק לגורמים הם, בהתאמה, וקטור שורה ווקטור עמודה בגודל $(n-1)$. האיבר vw^{T}/a_{11}, המתקבל מחישוב המכפלה החיצונית של v ו־w וחלוקת כל איבר של התוצאה ב־a_{11}, הוא מטריצה בגודל $(n-1)\times(n-1)$ המתאימה בגודלה למטריצה A' שממנה מחסרים אותה. המטריצה בגודל $(n-1)\times(n-1)$ המתקבלת:

$$A' = vw^{\mathrm{T}}/a_{11} \tag{31.23}$$

נקראת **משלים־שור** (Schur complement) של A ביחס ל־a_{11}.

למשלים־שור זה אנו מוצאים עתה, באופן רקורסיבי, פירוק LU. נניח שפירוק זה הוא:

$$A' - vw^{\mathrm{T}}/a_{11} = L'U'$$

כאשר L' היא מטריצת יחידה משולשית־תחתונה ו־U' היא מטריצה משולשית־עליונה. אזי, בעזרת אלגברת מטריצות, נקבל:

$$\begin{aligned} A &= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ 0 & A' - vw^{\mathrm{T}}/a_{11} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ 0 & L'U' \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ v/a_{11} & L' \end{pmatrix} \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ 0 & U' \end{pmatrix} \\ &= LU \end{aligned}$$

וזהו הפירוק LU שלנו. (נשים לב כי משום ש־L' היא מטריצת יחידה משולשית־תחתונה, כך היא גם L, ומשום ש־U' משולשית־תחתונה, כך היא גם U.)

כמובן, אם $a_{11}=0$, שיטה זו אינה פועלת, משום שאז יש לחלק באפס. השיטה אינה פועלת גם כאשר האיבר השמאלי העליון במשלים־שור $A' - vw^{\mathrm{T}}/a_{11}$ הוא אפס, שכן זהו האיבר שבו מחלקים בצעד הבא של הרקורסיה. האיברים שבהם מחלקים במהלך פירוק LU נקראים **איברי־ציר** (pivots) והם נמצאים באלכסון של המטריצה U. הסיבה לכך שאנו כוללים מטריצת תמורה P בפירוק LUP היא, שהיא מאפשרת לנו להימנע מחלוקה באיברים השווים לאפס. השימוש בתמורות כדי להימנע מחלוקה באפס (או במספרים קטנים) נקרא **סיבוב על ציר** (pivoting).

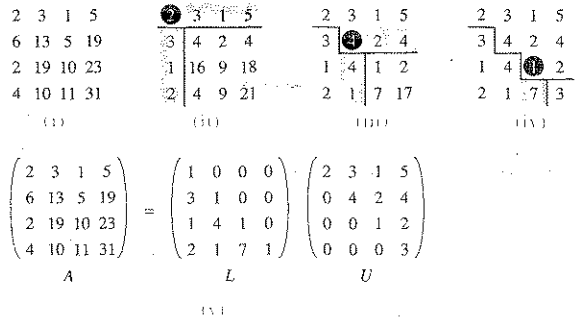
מחלקה חשובה של מטריצות שתמיד ניתן לפרק אותן פירוק LU היא מחלקת המטריצות הסימטריות החיוביות־לחלוטין. במטריצות כאלה לא דרוש סיבוב על ציר, ולכן האסטרטגיה הרקורסיבית המתוארת לעיל ניתנת להפעלה ללא חשש לחלוקה באפס. את התוצאה הזאת ואיתה כמה תוצאות אחרות נוכיח בסעיף 31.6.

הקוד שלנו עבור פירוק LU של מטריצה A מבוסס על האסטרטגיה הרקורסיבית, אלא שלולאה איטרטיבית מחליפה את הרקורסיה. (שינוי זה הוא אופטימיזציה מקובלת של שגרות שבהן הרקורסיה היא "רקורסיית זנב" — שגרות שפעולתן האחרונה היא קריאה רקורסיבית לעצמן.) הקוד מניח שהממד של A מאוחסן במאפיין $rows[A]$. מאחר שאנו יודעים כי במטריצת הפלט U כל האיברים שמתחת לאלכסון הם אפסים, ומאחר ש־LUP-Solve אינה פונה לתאים אלה, הקוד אינו טורח למלאם. באופן דומה, מכיוון שבמטריצת הפלט L כל איברי האלכסון שווים ל־1 וכל האיברים שמעל לאלכסון שווים ל־0, הקוד אינו ממלא גם תאים אלה. הקוד מחשב אפוא רק את ערכי התאים ה"משמעותיים" של L ו־U.

```
LU-DECOMPOSITION(A)
1   n ← rows[A]
2   for k ← 1 to n
3       do u_kk ← a_kk
4          for i ← k + 1 to n
5              do l_ik ← a_ik / u_kk                ▷ l_ik holds v_i
6                 u_ik ← a_ki                       ▷ u_ki holds w_i^T
7          for i ← k + 1 to n
8              do for j ← k + 1 to n
9                     do a_ij ← a_ij − l_ik u_kj
10  return L and U
```

לולאת ה־**for** החיצונית המתחילה בשורה 2 מתבצעת פעם אחת עבור כל צעד של הרקורסיה. בתוך לולאה זו, בשורה 3, נקבע איבר הציר כ־$u_{kk}=a_{kk}$. בתוך לולאת ה־**for** שבשורות 4-6 (שאינה מתבצעת כאשר $k=n$), משתמשים בווקטורים v ו־w^{T} לעדכון L ו־U. איברי הווקטור v נקבעים בשורה 5, כאשר v_i מאוחסן ב־l_{ik}, ואיברי הווקטור w^{T} נקבעים בשורה 6, כאשר w_i^{T} מאוחסן ב־u_{ki}. לבסוף, איברי משלים־שור מחושבים בשורות 7-9 ומאוחסנים בחזרה במטריצה A. מכיוון ששורה 9 מקוננת קינון משולש, זמן הריצה של LU-DECOMPOSITION הוא $\Theta(n^3)$.

איור 31.1 מדגים את פעולתה של LU-DECOMPOSITION. באיור מוצגת אופטימיזציה מקובלת של השגרה, שבה האיברים המשמעותיים של L ושל U מאוחסנים "בְּמָקוֹם" במטריצה A. כלומר, ניתן ליצור התאמה בין כל איבר a_{ij} לבין l_{ij} (אם $i>j$) או u_{ij} (אם $i\le j$) ולעדכן את המטריצה A כך שתכיל גם את L וגם את U כאשר השגרה נעצרת. הפסאודו־קוד עבור אופטימיזציה זו מתקבל מן הפסאודו־קוד שלעיל פשוט על ידי החלפת כל התייחסות ל־l או ל־u בהתייחסות ל־a המתאים; אין זה קשה לוודא ששינוי זה משמר את הנכונות.



איור 31.1 פעולתה של LU-DECOMPOSITION. **(i)** המטריצה A. **(ii)** האיבר $a_{11} = 2$ בעיגול השחור הוא איבר הציר, העמודה המוצללת היא v/a_{11}, והשורה המוצללת היא w^T. איברי U שחושבו עד כה נמצאים מעל לקו האופקי, ואיברי L נמצאים משמאל לקו האנכי. מטריצת משלים־שור $A' - vw^T/a_{11}$ היא המטריצה המופיעה מימין למטה. **(iii)** עתה פועלים על מטריצת משלים־שור שהתקבלה מחלק (ii). האיבר $a_{22} = 4$ בעיגול השחור הוא איבר הציר, והעמודה והשורה המוצללות הן, בהתאמה, v/a_{22} ו־w^T (בחלוקה של משלים־שור). הקווים מחלקים את המטריצה לאיברי U שחושבו עד כה (מעל לקו), איברי L שחושבו עד כה (משמאל לקו), ולמשלים־שור החדש (מימין למטה). **(iv)** הצעד הבא משלים את הפירוק לגורמים (האיבר 3 במשלים־שור החדש הופך עם עצירת הרקורסיה לחלק מ־U.) **(v)** הפירוק לגורמים $A = LU$.

חישוב פירוק LUP

באופן כללי, כאשר פותרים מערכת משוואות לינאריות $Ax = b$, חייבים לבחור כאיברי ציר איברים שאינם נמצאים על האלכסון של A, כדי למנוע חלוקה ב־0. לא רק חלוקה ב־0 אינה רצויה, אלא גם חלוקה בכל ערך קטן, אפילו אם A רגולרית, שכן חלוקה כזאת עלולה לגרום לאי־יציבות מספרית. משום כך נשה לבחור ערכים גדולים כאיברי ציר.

המתמטיקה הנדרשת בפירוק LUP דומה לזו הנדרשת בפירוק LU. נזכר כי נתונה לנו מטריצה רגולרית A בגודל $n \times n$, וברצוננו למצוא מטריצת תמורה P, מטריצת יחידה משולשית־תחתונה L ומטריצה משולשית־עליונה U המקיימות $PA = LU$. לפני שנחלק את המטריצה A כפי שעשינו בפירוק LU, נעביר איבר שונה מ־0, נאמר a_{k1}, מן העמודה הראשונה למיקום $(1, 1)$ במטריצה. (אם העמודה הראשונה מכילה אפסים בלבד, אזי A סינגולרית, משום שעל פי משפטים 31.4 ו־31.5, הדטרמיננטה שלה שווה ל־0.) כדי לשמר את מערכת המשוואות, נחליף זו בזו את שורה 1 ושורה k, פעולה השקולה לכפל A משמאל במטריצת תמורה Q (תרגיל 31.1-2). לכן נוכל לכתוב את QA בצורה:

$$QA = \begin{pmatrix} a_{k1} & w^T \\ v & A' \end{pmatrix}$$

כאשר $v = (a_{21}, a_{31}, \ldots, a_{n1})^T$, פרט לכך שהאיבר a_{11} מחליף את a_{k1}; כמו כן, $w^T = (a_{k2}, a_{k3}, \ldots, a_{kn})$; ו־$A'$ היא מטריצה בגודל $(n-1) \times (n-1)$. מכיוון שמתקיים $a_{k1} \neq 0$, נוכל עתה לבצע פחות או יותר אותה אלגברה לינארית כמו בפירוק LU, אך עתה מובטח לנו שאיננו מחלקים ב־0:

$$\begin{aligned} QA &= \begin{pmatrix} a_{k1} & w^T \\ v & A' \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{pmatrix} \end{aligned}$$

משלים־שור $A' - vw^T/a_{k1}$ הוא רגולרי, שכן אחרת היתה הדטרמיננטה של המטריצה השנייה במשוואה האחרונה שווה ל־0, ומכך היה נובע שהדטרמיננטה של A שווה ל־0, אך דבר זה פירושו ש־A סינגולרית, בסתירה להנחתנו ש־A רגולרית. לכן ניתן למצוא באופן אינדוקטיבי פירוק LUP של משלים־שור, דהיינו מטריצת יחידה משולשית־תחתונה L', מטריצה משולשית־עליונה U' ומטריצת תמורה P' המקיימות:

$$P'(A' - vw^T/a_{k1}) = L'U'$$

נגדיר:

$$P = \begin{pmatrix} 1 & 0 \\ 0 & P' \end{pmatrix} Q$$

P היא מטריצת תמורה, מכיוון שהיא מכפלה של שתי מטריצות תמורה (תרגיל 31.1-2). עתה אנו מקבלים:

$$\begin{aligned} PA &= \begin{pmatrix} 1 & 0 \\ 0 & P' \end{pmatrix} QA \\ &= \begin{pmatrix} 1 & 0 \\ 0 & P' \end{pmatrix} \begin{pmatrix} 1 & 0 \\ v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & P' \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & P'(A' - vw^T/a_{k1}) \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & L'U' \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & L' \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & U' \end{pmatrix} \\ &= LU \end{aligned}$$

וזהו פירוק LUP. מכיוון ש־L' היא מטריצת יחידה משולשית־תחתונה, כך היא גם L, ומכיוון ש־U' היא משולשית־תחתונה, כך היא גם U.

נשים לב כי בפיתוח זה, בניגוד לפיתוח עבור פירוק LU, יש לכפול גם את וקטור העמודה v/a_{k1} וגם את משלים־שור $A' - vw^T/a_{k1}$ במטריצת התמורה P'.

כמו בשגרה LU-DECOMPOSITION, גם בפסאודו־קוד שלנו עבור פירוק LUP הרקורסיה מוחלפת בלולאה איטרטיבית. כשיפור לעומת מימוש ישיר של הרקורסיה, אנו מתחזקים את מטריצת

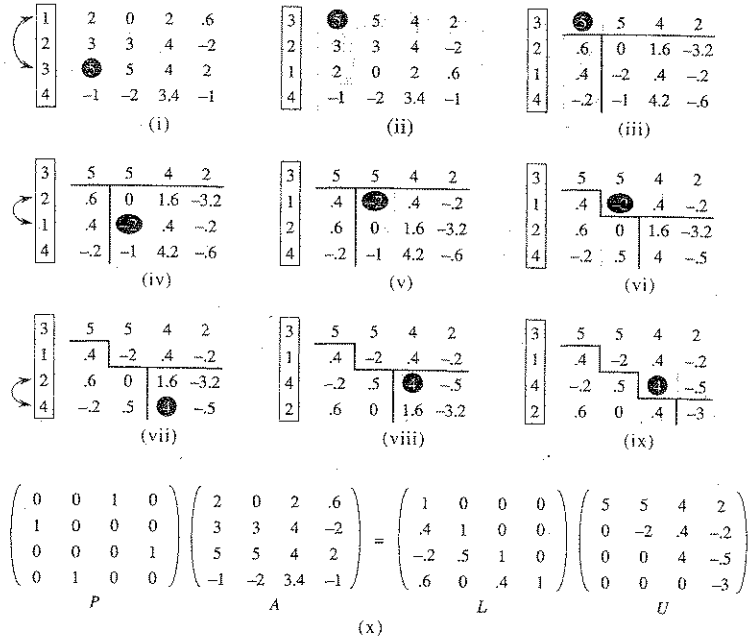
התמורה P באופן דינמי כמערך π, כאשר $\pi[i]=j$ פירושו שהשורה ה־i של P מכילה 1 בעמודה ה־j. כמו כן, הקוד ממומש כך ש־L ו־U מחושבות "בְּמָקוֹם" במטריצה A. לבן, כאשר השגרה עוצרת, מתקיים:

$$a_{ij}=\begin{cases} l_{ij} & i>j \quad \text{אם} \\ u_{ij} & i\le j \quad \text{אם} \end{cases}$$

```
LUP-DECOMPOSITION(A)
 1  n ← rows[A]
 2  for i ← 1 to n
 3      do π[i] ← i
 4  for k ← 1 to n
 5      do p ← 0
 6          for i ← k to n
 7              do if |a_ik| > p
 8                  then p ← |a_ik|
 9                       k' ← i
10          if p = 0
11            then error "singular matrix"
12          exchange π[k] ↔ π[k']
13          for i ← 1 to n
14              do exchange a_ki ↔ a_k'i
15          for i ← k + 1 to n
16              do a_ik ← a_ik / a_kk
17                  for j ← k + 1 to n
18                      do a_ij ← a_ij − a_ik a_kj
```

איור 31.2 מדגים כיצד LUP-DECOMPOSITION מפרקת מטריצה לגורמים. בשורות 2-3 מאותחל המערך π כך שייצג את תמורת הזהות. לולאת ה־**for** החיצונית המתחילה בשורה 4 מממשת את הרקורסיה. בכל איטרציה של הלולאה החיצונית, שורות 5-9 קובעות מהו האיבר $a_{k'k}$ בעל הערך המוחלט הגדול ביותר מבין האיברים בעמודה הראשונה הנוכחית (העמודה ה־k) של המטריצה בגודל $(n-k+1)\times(n-k+1)$ שאת פירוק LU שלה יש למצוא. אם כל האיברים בעמודה הראשונה הנוכחית שווים לאפס, שורות 10-11 מדווחות כי המטריצה סינגולרית. כדי לבצע סיבוב על ציר, אנו מחליפים זה בזה את $\pi[k']$ ו־$\pi[k]$ בשורה 12, ומחליפים זו בזו את השורות ה־k וה־k' של A בשורות 13-14, כך שאיבר הציר הוא a_{kk}. (השורות כולן מוחלפות זו בזו מכיוון שבשיטה שהוצגה לעיל, לא רק $A'-vw^{\mathrm{T}}/a_{k1}$ מוכפל ב־P' אלא גם v/a_{k1}.) לבסוף, מחשבים את משלים־שור בשורות 15-18 באופן דומה מאוד לזה שבו חישבנו אותו בשורות 4-9 של LU-DECOMPOSITION, פרט לכך שכאן הפעולה נכתבה כך שהיא מתבצעת "בְּמָקוֹם".

בשל מבנה הקינון המשולש, זמן הריצה של LUP-DECOMPOSITION הוא $\Theta(n^3)$, כמו זה של LU-DECOMPOSITION. העלות בזמן של הסיבוב על ציר אינה עולה אפוא על גורם קבוע.



איור 31.2 פעולתה של LUP-DECOMPOSITION. **(i)** מטריצת הקלט A, ומשמאלה תמורת הזהות של שורות המטריצה. הציר הראשון של האלגוריתם קובע שהאיבר 5, הנמצא בשורה השלישית ומופיע בתוך עיגול שחור, הוא איבר הציר עבור העמודה הראשונה. **(ii)** האלגוריתם מחליף זו בזו את השורות 1 ו־3 ומעדכן את התמורה. העמודה והשורה המוצללות מייצגות את v ואת w^{T}, בהתאמה. **(iii)** הווקטור v מוחלף ב־$v/5$, והחלק הימני התחתון של המטריצה מעודכן בערכי משלים־שור. הקווים באיור מחלקים את המטריצה לשלושה אזורים: איברים של U (למעלה), איברים של L (משמאל) ואיברים של משלים־שור (מימין למטה). **(iv)-(vi)** הציר השני. **(vii)-(ix)** הציר השלישי. בציר הרביעי והאחרון לא מתרחשים שום שינויים. **(x)** פירוק LUP של A: $PA=LU$.

תרגילים

31.4-1

פתור את המשוואה הבאה באמצעות הצבה קדימה:

$$\begin{pmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ -6 & 5 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}=\begin{pmatrix} 3 \\ 14 \\ -7 \end{pmatrix}$$

31.4-2
מצא פירוק LU של המטריצה:

$$\begin{pmatrix} 4 & -5 & 6 \\ 8 & -6 & 7 \\ 12 & -7 & 12 \end{pmatrix}$$

31.4-3
פתור את המשוואה הבאה באמצעות פירוק LUP:

$$\begin{pmatrix} 1 & 5 & 4 \\ 2 & 0 & 3 \\ 5 & 8 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 12 \\ 9 \\ 5 \end{pmatrix}$$

31.4-4
תאר את פירוק LUP של מטריצה אלכסונית.

31.4-5
תאר את פירוק LUP של מטריצת תמורה A והוכח שהוא יחיד.

31.4-6
הראה שעבור כל $n \geq 1$, קיימות מטריצות סינגולריות בגודל $n \times n$ שיש להן פירוקי־LU.

31.4-7
הראה כיצד ניתן לפתור ביעילות מערכת משוואות מהצורה $Ax = b$ מעל המעין־חוג הבוליאני $(\{0, 1\}, \vee, \wedge, 0, 1)$.

31.4-8 *
נניח ש־A היא מטריצה ממשית בגודל $n \times n$ שדרגתה m, כאשר $m < n$. הראה כיצד ניתן למצוא וקטור x_0 בגודל n ומטריצה B בגודל $m \times (n-m)$ שדרגתה $n-m$ כך שכל וקטור מהצורה $x_0 + By$, כאשר $y \in \mathbf{R}^{n-m}$, הוא פתרון למשוואה התת־מוגדרת $Ax = b$.

31.5 היפוך מטריצות

למרות שביישומים מעשיים אין משתמשים בדרך כלל במטריצות הופכיות בפתרון מערכות של משוואות לינאריות, ומעדיפים תחת זאת שיטות יציבות יותר מבחינה מספרית כגון פירוק LUP. לעתים יש הכרח לחשב מטריצה הופכית. בסעיף זה נראה כיצד ניתן להשתמש בפירוק LUP לחישוב מטריצה הופכית. נדון גם בשאלה המעניינת מבחינה תיאורטית אם ניתן להחיש חישוב של מטריצה הופכית באמצעות שיטות כמו האלגוריתם של שטראסן לכפל מטריצות. ואכן, המניע למאמר המקורי של שטראסן היה רצונו להראות כי ניתן לפתור מערכת של משוואות לינאריות במהירות רבה יותר מאשר בשיטה הרגילה.

חישוב מטריצה הופכית מפירוק LUP

נניח שנתון לנו פירוק LUP של מטריצה A בצורת שלוש מטריצות L, U ו־P כך ש־$PA = LU$. באמצעות LUP-Solve אנו יכולים לפתור משוואה מהצורה $Ax = b$ בזמן $\Theta(n^2)$. מכיוון שפירוק LUP תלוי ב־A אולם לא ב־b, נוכל לפתור מערכת שנייה של משוואות מהצורה $Ax = b'$ בזמן נוסף של $\Theta(n^2)$. באופן כללי, בהינתן פירוק LUP של A, הרי שבזמן $\Theta(kn^2)$ ניתן לפתור k גרסאות של המשוואה $Ax = b$ השונות זו מזו בערכי b בלבד.

את המשוואה:

$$AX = I_n \tag{31.24}$$

ניתן לראות כמערכת של n משוואות שונות מהצורה $Ax = b$. משוואות אלה מגדירות את המטריצה X כהופכית של A. ביתר דיוק, אם נסמן ב־X_i את העמודה ה־i של X ונזכור כי העמודה ה־i של I_n היא וקטור היחידה e_i, אזי ניתן לפתור את משוואה (31.24) עבור X על־ידי כך שמשתמשים בפירוק LUP כדי לפתור בנפרד כל משוואה:

$$AX_i = e_i$$

עבור X_i כל אחד מ־n הווקטורים X_i ניתן למצוא בזמן $\Theta(n^2)$, ולכן חישוב X מתוך פירוק LUP של A מתבצע בזמן $\Theta(n^3)$. מכיוון שפירוק LUP של A ניתן לחישוב בזמן $\Theta(n^3)$, הרי שאת ההופכית A^{-1} של מטריצה A ניתן למצוא בזמן $\Theta(n^3)$.

כפל מטריצות והיפוך מטריצות

עתה נראה שניתן לתרגם את השיפורים התיאורטיים בזמן הריצה של כפל מטריצות לשיפורים בזמן הריצה של היפוך מטריצות. למעשה נוכיח משהו חזק יותר: היפוך מטריצות שקול לכפל מטריצות במובן הבא: אם $M(n)$ מציין את הזמן לכפל שתי מטריצות בגודל $n \times n$ ו־$I(n)$ מציין את הזמן להיפוך מטריצה רגולרית בגודל $n \times n$, אזי $I(n) = \Theta(M(n))$. את התוצאה הזאת נוכיח בשני חלקים. תחילה נראה כי $M(n) = O(I(n))$, וזוהי הוכחה קלה יחסית, ולאחר מכן נוכיח כי $I(n) = O(M(n))$.

משפט 31.11 (כפל אינו קשה יותר מהיפוך)
אם ניתן לבצע היפוך מטריצה בגודל $n \times n$ בזמן $I(n)$, כאשר $I(n) = \Omega(n^2)$, ו־$I(n)$ מקיים את תנאי הרגולריות $I(3n) = O(I(n))$, אזי ניתן לכפול שתי מטריצות בגודל $n \times n$ בזמן $O(I(n))$.

הוכחה יהיו A ו־B מטריצות בגודל $n \times n$ שאת מטריצת המכפלה שלהן C אנו רוצים לחשב. נגדיר מטריצה D בגודל $3n \times 3n$ על־ידי:

$$D = \begin{pmatrix} I_n & A & 0 \\ 0 & I_n & B \\ 0 & 0 & I_n \end{pmatrix}$$

המטריצה ההופכית ל־D היא:

$$D^{-1} = \begin{pmatrix} I_n & -A & AB \\ 0 & I_n & -B \\ 0 & 0 & I_n \end{pmatrix}$$

ניתן אפוא לחשב את המכפלה AB על־ידי כך שלוקחים את התת־מטריצה הימנית העליונה בגודל $n \times n$ של D^{-1}.

את המטריצה D ניתן לבנות בזמן $\Theta(n^2) = O(I(n))$, ואת ההופכית של D ניתן לחשב בזמן $O(I(3n)) = O(I(n))$, על פי תנאי הרגולריות שמקיים $I(n)$. לכן אנו מקבלים:

$$M(n) = O(I(n))$$

■

נשים לב כי $I(n)$ מקיים את תנאי הרגולריות כאשר ערכו אינו משתנה בקפיצות גדולות. לדוגמה, אם $I(n) = \Theta(n^c \lg^d n)$, עבור קבועים כלשהם $c > 0$ ו־$d \geq 0$, אזי $I(n)$ מקיים את תנאי הרגולריות.

רדוקציה של היפוך מטריצות לכפל מטריצות

ההוכחה שהיפוך מטריצות אינו קשה יותר מכפל מטריצות מסתמכת על כמה תכונות של מטריצות סימטריות חיוביות־לחלוטין שאותן נוכיח בסעיף 31.6.

משפט 31.12 (היפוך אינו קשה יותר מכפל)
נניח שניתן לכפול שתי מטריצות ממשיות בגודל $n \times n$ בזמן $M(n)$, כאשר $M(n) = \Omega(n^2)$ ו־$M(n)$ מקיים את שני תנאי הרגולריות $M(n+k) = O(M(n))$ עבור כל k בתחום $0 \leq k \leq n$, ו־$M(n/2) \leq cM(n)$ עבור איזשהו קבוע $c < 1/2$. אזי ניתן לחשב את ההופכית של כל מטריצה ממשית רגולרית בגודל $n \times n$ בזמן $O(M(n))$.

הוכחה נוכל להניח כי n הוא חזקה מדויקת של 2, שכן עבור כל $k > 0$, מתקיים:

$$\begin{pmatrix} A & 0 \\ 0 & I_k \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} & 0 \\ 0 & I_k \end{pmatrix}$$

לכן, על־ידי בחירה של k כך ש־$n + k$ הוא חזקה מדויקת של 2, נוכל להרחיב את המטריצה לגודל שהוא החזקה הבאה של 2 ולקבל את התשובה המבוקשת A^{-1} מן התשובה לבעיה המורחבת. תנאי הרגולריות הראשון שמקיים $M(n)$ מבטיח שהרחבה זו לא תגרום לכך שזמן הריצה יגדל פי יותר מאשר גורם קבוע.

לעת עתה, נניח שהמטריצה A בגודל $n \times n$ היא סימטרית וחיובית־לחלוטין. נחלק את A לארבע תת־מטריצות בגודל $n/2 \times n/2$:

$$A = \begin{pmatrix} B & C^{\mathrm{T}} \\ C & D \end{pmatrix} \tag{31.25}$$

נסמן ב־S את משלים־שור של A ביחס ל־B:

$$S = D - CB^{-1}C^{\mathrm{T}} \tag{31.26}$$

ונקבל:

$$A^{-1} = \begin{pmatrix} B^{-1} + B^{-1}C^{\mathrm{T}}S^{-1}CB^{-1} & -B^{-1}C^{\mathrm{T}}S^{-1} \\ -S^{-1}CB^{-1} & S^{-1} \end{pmatrix} \tag{31.27}$$

שכן $AA^{-1} = I_n$, כפי שניתן לוודא על־ידי ביצוע כפל המטריצות. המטריצות B^{-1} ו־S^{-1} קיימות אם A סימטרית וחיובית־לחלוטין, על פי למות 31.13, 31.14 ו־31.15 שיובאו בסעיף 31.6, מכיוון ש־B וגם S סימטריות וחיוביות־לחלוטין. על פי תרגיל 31.1-3, וכן $B^{-1}C^{\mathrm{T}} = (CB^{-1})^{\mathrm{T}}$ ו־$B^{-1}C^{\mathrm{T}}S^{-1} = (S^{-1}CB^{-1})^{\mathrm{T}}$. לכן ניתן להשתמש במשוואות (31.26) ו־(31.27) כדי לכתוב אלגוריתם רקורסיבי שיכלול 4 מכפלות של מטריצות בגודל $n/2 \times n/2$:

$$C \cdot B^{-1}$$

$$(CB^{-1}) \cdot C^{\mathrm{T}}$$

$$S^{-1} \cdot (CB^{-1})$$

$$(CB^{-1})^{\mathrm{T}} \cdot (S^{-1}CB^{-1})$$

מתנאי הרגולריות השני נובע שניתן לבצע היפוך של מטריצות סימטריות חיוביות־לחלוטין בזמן:

$$\begin{aligned} I(n) &\leq 2I(n/2) + 4M(n/2) + O(n^2) \\ &= 2I(n/2) + O(M(n)) \\ &= O(M(n)) \end{aligned}$$

נותר להוכיח שזמן הריצה האסימפטוטי הזה של כפל מטריצות מתקבל עבור היפוך מטריצות גם כאשר A הפיכה אולם אינה סימטרית ואינה חיובית־לחלוטין. הרעיון הבסיסי הוא שעבור כל מטריצה רגולרית A, המטריצה $A^{\mathrm{T}}A$ היא סימטרית (על פי תרגיל 31.1-3) וחיובית־לחלוטין (על פי משפט 31.6). התחבולה היא אפוא לעשות רדוקציה של בעיית היפוך המטריצה A לבעיית ההיפוך של $A^{\mathrm{T}}A$.

הרדוקציה מבוססת על ההבחנה שאם A היא מטריצה רגולרית בגודל $n \times n$, אזי:

$$A^{-1} = (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}$$

שכן $A^{-1} = ((A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}})A = (A^{\mathrm{T}}A)^{-1}(A^{\mathrm{T}}A) = I_n$ ומטריצה הופכית היא יחידה. לכן, נוכל לחשב את A^{-1} על־ידי כך שנכפול תחילה את A^{T} ב־A כדי לקבל את $A^{\mathrm{T}}A$, לאחר מכן נבצע היפוך של המטריצה הסימטרית החיובית־לחלוטין $A^{\mathrm{T}}A$ באמצעות האלגוריתם הפרד־ומשול שלעיל, ולבסוף נכפול את התוצאה ב־A^{T}. כל אחד משלושת הצעדים האלה מתבצע בזמן $O(M(n))$, ומכאן שניתן לבצע היפוך של כל מטריצה רגולרית בעלת ערכים ממשיים בזמן $O(M(n))$. ■

ההוכחה של משפט 31.12 מציעה אמצעי לפתרון המשוואה $Ax = b$ ללא סיבוב על ציר, כל עוד A רגולרית. כופלים את שני אגפי המשוואה ב־A^{T} ומקבלים $(A^{\mathrm{T}}A)x = A^{\mathrm{T}}b$. טרנספורמציה זו אינה משפיעה על הפתרון x, שכן A^{T} הפיכה, ולכן ניתן לפרק לגורמים את המטריצה הסימטרית החיובית־לחלוטין $A^{\mathrm{T}}A$ על־ידי חישוב של פירוק LU. לאחר מכן משתמשים בהצבה קדימה ואחורה כדי לפתור עבור x את המשוואה שאגף ימין שלה הוא $A^{\mathrm{T}}b$. אף כי שיטה זו נכונה באופן תיאורטי, הרי שביישומים מעשיים השגרה LUP-Decomposition פועלת בצורה טובה בהרבה. מספר הפעולות האריתמטיות שדורש פירוק LUP קטן יותר פי גורם קבוע, והוא בעל תכונות מספריות טובות במעט.

תרגילים

31.5-1

יהי $M(n)$ הזמן לביצוע כפל מטריצות בגודל $n \times n$, ויהי $S(n)$ הזמן הדרוש להעלאה בריבוע של מטריצה בגודל $n \times n$. הראה שכפל מטריצות והעלאתן בריבוע הם בעיקרם בעלי אותה דרגת קושי: $S(n) = \Theta(M(n))$.

31.5-2

יהי $M(n)$ הזמן לביצוע כפל מטריצות בגודל $n \times n$, ויהי $L(n)$ הזמן לחישוב פירוק LUP של מטריצה בגודל $n \times n$. הראה שכפל מטריצות וחישוב פירוק LUP של מטריצות הם בעיקרם בעלי אותה דרגת קושי: $L(n) = \Theta(M(n))$.

31.5-3

יהי $M(n)$ הזמן לביצוע כפל מטריצות בגודל $n \times n$, ויהי $D(n)$ הזמן לחישוב הדטרמיננטה של מטריצה בגודל $n \times n$. הראה שחישוב הדטרמיננטה אינו קשה יותר מכפל מטריצות: $D(n) = O(M(n))$.

31.5-4

יהי $M(n)$ הזמן לביצוע כפל מטריצות בוליאניות בגודל $n \times n$, ויהי $T(n)$ הזמן הדרוש למציאת הסגור הטרנזיטיבי של מטריצות בוליאניות בגודל $n \times n$. הראה כי $M(n) = O(T(n))$ וכן כי $T(n) = O(M(n)\lg n)$.

31.5-5

האם האלגוריתם להיפוך מטריצות המבוסס על משפט 31.12 פועל כאשר איברי המטריצות לקוחים משדה השלמים מודולו 2? הסבר.

* 31.5-6

הכלל את האלגוריתם להיפוך מטריצות המבוסס על משפט 31.12 כך שיטפל במטריצות של מספרים מרוכבים, והוכח שההכללה שלך פועלת כהלכה. **רמז:** במקום המטריצה המוחלפת של A, השתמש במטריצה **המוחלפת הצמודה** (conjugate transpose) A^*, המתקבלת מהמטריצה המוחלפת של A על־ידי החלפת כל איבר בצמוד המרוכב שלו. במקום מטריצות סימטריות, התייחס למטריצות **הרמיטיות** (Hermitian), שהן מטריצות A המקיימות $A = A^*$.)

31.6 מטריצות סימטריות חיוביות־לחלוטין וקירוב על־ידי ריבועים פחותים

מטריצות סימטריות חיוביות־לחלוטין הן בעלות תכונות רבות שהן מעניינות ורצויות. לדוגמה, הן רגולריות, וניתן לבצע פירוק LU שלהן ללא חשש לחלוקה ב־0. בסעיף זה נוכיח כמה תכונות חשובות אחרות של מטריצות סימטריות חיוביות־לחלוטין ונראה יישום מעניין של מטריצות אלה בהתאמת עקומים על־ידי קירוב בשיטת הריבועים הפחותים.

התכונה הראשונה שנוכיח היא אולי הבסיסית ביותר.

למה 31.13

כל מטריצה סימטרית חיובית־לחלוטין היא רגולרית.

הוכחה נניח שמטריצה A היא סינגולרית. אזי על פי מסקנה 31.3, קיים וקטור שונה מאפס x כך ש־$Ax = 0$. מכאן, $x^T Ax = 0$, ו־A אינה יכולה להיות חיובית־לחלוטין. ■

ההוכחה לכך שניתן לבצע פירוק LU של מטריצה סימטרית חיובית־לחלוטין A ללא חלוקה ב־0 היא מורכבת יותר. נתחיל בהוכחת תכונות של תת־מטריצות מסוימות של A. **התת־מטריצה המובילה** ה־k (**kth leading submatrix**) של A היא המטריצה A_k המורכבת מהחיתוך של k השורות הראשונות ו־k העמודות הראשונות של A.

למה 31.14

אם A היא מטריצה סימטרית חיובית־לחלוטין, אזי כל תת־מטריצה מובילה של A היא סימטרית וחיובית־לחלוטין.

הוכחה העובדה שכל תת־מטריצה מובילה A_k היא סימטרית ברורה מאליה. כדי להוכיח ש־A_k היא חיובית־לחלוטין, יהי x וקטור עמודה שונה מאפס בגודל k, ונחלק את A באופן הבא:

$$A = \begin{pmatrix} A_k & B^T \\ B & C \end{pmatrix}$$

אזי אנו מקבלים:

$$\begin{aligned} x^T A_k x &= (x^T \ 0) \begin{pmatrix} A_k & B^T \\ B & C \end{pmatrix} \begin{pmatrix} x \\ 0 \end{pmatrix} \\ &= (x^T \ 0) A \begin{pmatrix} x \\ 0 \end{pmatrix} \\ &> 0 \end{aligned}$$

שכן A חיובית־לחלוטין, ולכן גם A_k חיובית־לחלוטין. ■

נפנה עתה לכמה תכונות מהותיות של משלים־שור. תהי A מטריצה סימטרית חיובית־לחלוטין, ותהי A_k תת־מטריצה מובילה בגודל $k \times k$ של A. נחלק את A באופן הבא:

$$A = \begin{pmatrix} A_k & B^T \\ B & C \end{pmatrix} \tag{31.28}$$

אזי, **משלים־שור** (Schur complement) של A ביחס ל־A_k מוגדר על־ידי:

$$S = C - BA_k^{-1}B^T \tag{31.29}$$

(על פי למה 31.14, A_k היא סימטרית וחיובית־לחלוטין; לכן, על פי למה 31.13, A_k^{-1} קיימת ו־S מוגדרת היטב.) אפשר לראות כי ההגדרה הקודמת שלנו (31.23) של משלים־שור קונסיסטנטית עם ההגדרה (31.29), על־ידי הצבה של $k = 1$.

הלמה הבאה מראה שמטריצות משלים־שור של מטריצות סימטריות חיוביות־לחלוטין הן עצמן סימטריות וחיוביות־לחלוטין. בתוצאה זו השתמשנו בהוכחת משפט 31.12 והמסקנה הנובעת ממנה דרושה להוכחת נכונותו של פירוק LU עבור מטריצות סימטריות חיוביות־לחלוטין.

למה 31.15 (למת משלים־שור)
אם A היא מטריצה סימטרית חיובית־לחלוטין ו־A_k היא תת־מטריצה מובילה בגודל $k \times k$ של A, אזי משלים־שור של A ביחס ל־A_k הוא מטריצה סימטרית וחיובית־לחלוטין.

הוכחה הסימטריות של S נובעת מתרגיל 31.1-7. נותר להראות כי S חיובית־לחלוטין. נתבונן בחלוקת A הנתונה במשוואה (31.28).

עבור כל וקטור שונה מאפס x, מתקיים על פי ההנחה $x^{\mathrm{T}}Ax > 0$. נפרק את x לשני תת־וקטורים y ו־z תואמים ל־A_k ול־C, בהתאמה. מכיוון ש־A_k^{-1} קיימת, נקבל באמצעות אלגברת מטריצות:

$$\begin{aligned} x^{\mathrm{T}}Ax &= \begin{pmatrix} y^{\mathrm{T}} & z^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} A_k & B^{\mathrm{T}} \\ B & C \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} \\ &= y^{\mathrm{T}}A_k y + y^{\mathrm{T}}B^{\mathrm{T}}z + z^{\mathrm{T}}By + z^{\mathrm{T}}Cz \\ &= (y + A_k^{-1}B^{\mathrm{T}}z)^{\mathrm{T}}A_k(y + A_k^{-1}B^{\mathrm{T}}z) + z^{\mathrm{T}}(C - BA_k^{-1}B^{\mathrm{T}})z \end{aligned} \tag{31.30}$$

(ניתן לוודא זאת על־ידי ביצוע כפל ישיר.) המשוואה האחרונה אינה אלא "השלמת הריבוע" של הצורה הריבועית (ראה תרגיל 31.6-2).

מכיוון ש־$x^{\mathrm{T}}Ax > 0$ מתקיים עבור כל x שונה מאפס, הבה נבחר z שונה מאפס כלשהו, ואז נבחר $y = -A_k^{-1}B^{\mathrm{T}}z$; דבר זה גורם לכך שהאיבר הראשון במשוואה (31.30) מתאפס, כך שהאגף הימני של המשוואה הוא:

$$z^{\mathrm{T}}(C - BA_k^{-1}B^{\mathrm{T}})z = z^{\mathrm{T}}Sz$$

עבור כל $z \neq 0$ אנו מקבלים אפוא כי $z^{\mathrm{T}}Sz = x^{\mathrm{T}}Ax > 0$, ולכן S היא חיובית־לחלוטין. ■

מסקנה 31.16
פירוק LU של מטריצה סימטרית חיובית־לחלוטין לעולם אינו גורם לחלוקה ב־0.

הוכחה תהי A מטריצה סימטרית חיובית־לחלוטין. נוכיח טענה חזקה יותר מטענת הלמה: כל איבר ציר הוא חיובי ממש. איבר הציר הראשון הוא a_{11}. יהי e_1 וקטור היחידה הראשון, ומכך אנו מקבלים $a_{11} = e_1^{\mathrm{T}}Ae_1 > 0$. מכיוון שהצעד הראשון של פירוק LU יוצר את משלים־שור של A ביחס ל־$A_1 = (a_{11})$, למה 31.15 גוררת, באינדוקציה, שכל איברי הציר חיוביים. ■

קירוב על־ידי ריבועים פחותים

התאמת עקומים לקבוצות נתונות של נקודות נתונים היא יישום חשוב של מטריצות סימטריות חיוביות־לחלוטין. נניח שנתונה לנו קבוצה של m נקודות נתונים:

$$(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$$

כאשר ידוע שהקואורדינטות y_i מועדות לשגיאות מדידה. ברצוננו למצוא פונקציה $F(x)$ המקיימת, עבור $i = 1, 2, \ldots, m$:

$$y_i = F(x_i) + \eta_i \tag{31.31}$$

כאשר שגיאות הקירוב η_i הן קטנות. צורתה של הפונקציה F תלויה בבעיה הנתונה. כאן אנו מניחים שהפונקציה F היא סכום לינארי משוקלל:

$$F(x) = \sum_{j=1}^{n} c_j f_j(x)$$

כאשר מספר המחוברים n **ופונקציות הבסיס** f_j נבחרים על סמך ידיעת הבעיה שיש לפתור. בחירה מקובלת היא $f_j(x) = x^{j-1}$, שפירושה כי $F(x)$ הוא פולינום ממעלה $n-1$ ב־x:

$$F(x) = c_1 + c_2 x + c_3 x^2 + \cdots + c_n x^{n-1}$$

על־ידי בחירת $n = m$ ניתן לחשב *במדויק* כל y_i במשוואה (31.31). אבל F ממעלה כה גבוהה "מתאים לרעש" לא פחות מאשר לנתונים, ובדרך כלל מניב תוצאות עלובות כאשר משתמשים בו לניבוי ערכי y עבור ערכי x בלתי ידועים. בדרך כלל עדיף לבחור n הקטן במידה משמעותית מ־m ולקוות כי על־ידי בחירה מוצלחת של המקדמים c_j נוכל לקבל פונקציה F שתגלה תבניות משמעותיות בנקודות הנתונים בלי להתייחס יותר מדי לרעש. קיימים כמה עקרונות תיאורטיים לבחירת n, אולם הם חורגים מתחומו של ספר זה. בכל מקרה, ברגע שבחרנו את n, אנו מקבלים מערכת משוואות מוגדרת־יתר שאותה אנו רוצים לפתור כמיטב יכולתנו. עתה נראה כיצד ניתן לעשות זאת.

תהי:

$$A = \begin{pmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_m) & f_2(x_m) & \cdots & f_n(x_m) \end{pmatrix}.$$

מטריצת הערכים של פונקציות הבסיס בנקודות הנתונות; כלומר, $a_{ij} = f_j(x_i)$. יהי $c = (c_k)$ וקטור המקדמים המבוקש בגודל n. אזי:

$$\begin{aligned} Ac &= \begin{pmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_m) & f_2(x_m) & \cdots & f_n(x_m) \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} \\ &= \begin{pmatrix} F(x_1) \\ F(x_2) \\ \vdots \\ F(x_m) \end{pmatrix} \end{aligned}$$

הוא הווקטור בגודל m של "ערכי הניבוי" עבור y. לכן:

$$\eta = Ac - y$$

הוא הווקטור בגודל m של **שגיאות הקירוב** (approximation errors).

כדי למזער את שגיאות הקירוב, אנו בוחרים למזער את הנורמה של וקטור השגיאות η, ובכך מתקבל **פתרון על־ידי ריבועים פחותים** (least-squares solution), שכן:

$$\|\eta\| = \left(\sum_{j=1}^{m} \eta_i^2 \right)^{1/2}$$

מכיוון שמתקיים:

$$\|\eta\|^2 = \|Ac - y\|^2 = \sum_{i=1}^{m}\left(\sum_{j=1}^{n} a_{ij}c_j - y_i\right)^2$$

ניתן למזער את $\|\eta\|$ על-ידי כך שנוזרים את $\|\eta\|^2$ לפי כל c_k ומשווים כל נגזרת ל-0:

$$\frac{d\|\eta\|^2}{dc_k} = \sum_{i=1}^{m} 2\left(\sum_{j=1}^{n} a_{ij}c_j - y_i\right)a_{ik} = 0 \tag{31.32}$$

המשוואות (31.32), עבור $k = 1, 2, \ldots, n$, שקולות למשוואת המטריצות היחידה הבאה:

$$(Ac - y)^{\mathrm{T}}A = 0$$

או, לחלופין (תוך שימוש בתרגיל 31.1-3), הן שקולות למשוואה:

$$A^{\mathrm{T}}(Ac - y) = 0$$

שממנה נובע:

$$A^{\mathrm{T}}Ac = A^{\mathrm{T}}y \tag{31.33}$$

בסטטיסטיקה, משוואה זו נקראת **משוואה נורמלית** (normal equation). המטריצה $A^{\mathrm{T}}A$ סימטרית (על פי תרגיל 31.1-3), ואם A היא בעלת דרגת עמודות מלאה, אזי $A^{\mathrm{T}}A$ היא גם חיובית-לחלוטין. לכן, $(A^{\mathrm{T}}A)^{-1}$ קיימת, והפתרון למשוואה (31.33) הוא:

$$\begin{aligned} c &= ((A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}})y \\ &= A^{+}y \end{aligned} \tag{31.34}$$

כאשר המטריצה $A^{+} = ((A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}})$ נקראת המטריצה **הפסאודו-הופכית** (pseudoinverse) של המטריצה A. המטריצה הפסאודו-הופכית היא הכללה טבעית של מושג המטריצה ההופכית עבור המקרה שבו A איננה ריבועית. (השווה את משוואה (31.34), שהיא הפתרון המקורב ל-$Ac = y$, עם $A^{-1}b$ שהוא הפתרון המדויק ל-$Ax = b$.)

כדוגמה למציאת התאמה באמצעות ריבועים פחותים, נניח שנתונות לנו 5 נקודות נתונים:

$$(-1, 2), (1, 1), (2, 1), (3, 0), (5, 3)$$

המופיעות כנקודות שחורות באיור 31.3. ברצוננו להתאים לנקודות אלה פולינום ריבועי:

$$F(x) = c_1 + c_2x + c_3x^2$$

נתחיל במטריצת הערכים של פונקציות הבסיס:

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \\ 1 & x_5 & x_5^2 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 5 & 25 \end{pmatrix}$$

אשר המטריצה הפסאודו-הופכית שלה היא:

$$A^{+} = \begin{pmatrix} 0.500 & 0.300 & 0.200 & 0.100 & -0.100 \\ -0.388 & 0.093 & 0.190 & 0.193 & -0.088 \\ 0.060 & -0.036 & -0.048 & -0.036 & 0.060 \end{pmatrix}$$

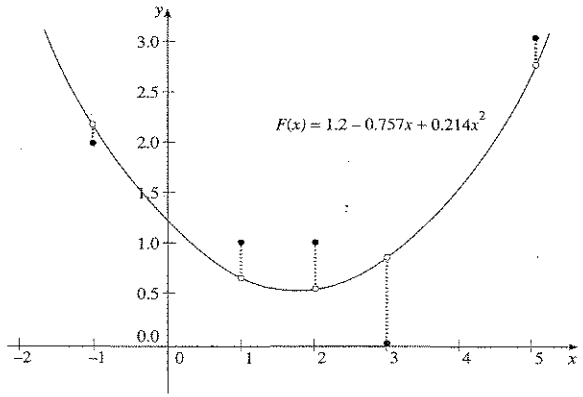
נכפול את y ב-A^{+}, ונקבל את וקטור המקדמים:

$$c = \begin{pmatrix} 1.200 \\ -0.757 \\ 0.214 \end{pmatrix}$$

היוצר את הפולינום הריבועי:

$$F(x) = 1.200 - 0.757x + 0.214x^2$$

שהוא הפולינום הריבועי הקרוב ביותר המתאים לנקודות הנתונות, במשמעות של ריבועים פחותים.



איור 31.3 ההתאמה בשיטת הריבועים הפחותים של פולינום ריבועי לקבוצת נקודות הנתונים $\{(-1, 2), (1, 1), (2, 1), (3, 0), (5, 3)\}$. הנקודות השחורות הן נקודות הנתונים, והנקודות הלבנות הן הערכים המקורבים שמנבא הפולינום $F(x) = 1.2 - 0.757x + 0.214x^2$. הפולינום הריבועי הממזער את סכום ריבועי השגיאות. השגיאה עבור כל נקודת נתונים מוצגת על-ידי קו מוצלל.

מסיבות מעשיות פותרים את המשוואה הנורמלית (31.33) על ידי כך שכופלים תחילה את y ב-A^{T}, ולאחר מכן מוצאים פירוק LU של $A^{\mathrm{T}}A$. אם A היא בעלת דרגה מלאה, מובטח לנו שהמטריצה $A^{\mathrm{T}}A$ היא רגולרית, משום שהיא סימטרית וחיובית-לחלוטין. (ראה תרגיל 31.1-3 ומשפט 31.6.)

תרגילים

31.6-1
הוכח שכל איבר באלכסון של מטריצה סימטרית וחיובית־לחלוטין הוא חיובי.

31.6-2
תהי $A = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$ מטריצה בגודל 2×2 סימטרית וחיובית־לחלוטין. הוכח שהדטרמיננטה שלה $ac - b^2$ היא חיובית באמצעות "השלמת הריבוע" בדרך דומה לזו שבה השתמשנו בהוכחת למה 31.15.

31.6-3
הוכח שהאיבר המקסימלי במטריצה סימטרית וחיובית־לחלוטין נמצא על האלכסון.

31.6-4
הוכח שהדטרמיננטה של כל תת־מטריצה מובילה של מטריצה סימטרית חיובית־לחלוטין היא חיובית.

31.6-5
נסמן ב־A_k את תת־מטריצה המובילה ה־k של מטריצה A סימטרית חיובית־לחלוטין. הוכח כי $\det(A_k)/\det(A_{k-1})$ הוא איבר הציר ה־k במהלך פירוק LU, כאשר נגדיר $\det(A_0) = 1$.

31.6-6
מצא את הפונקציה מהצורה:

$$F(x) = c_1 + c_2 x \lg x + c_3 e^x$$

שהיא ההתאמה הטובה ביותר בשיטת הריבועים הפחותים לנקודות הנתונים:

$$(1, 1), (2, 1), (3, 3), (4, 8)$$

31.6-7
הראה שהמטריצה הפסאודו־הופכית A^+ מקיימת את ארבע המשוואות הבאות:

$$AA^+A = A$$
$$A^+AA^+ = A^+$$
$$(AA^+)^{\mathrm{T}} = AA^+$$
$$(A^+A)^{\mathrm{T}} = A^+A$$

בעיות

31-1 האלגוריתם של שמיר (Shamir) לכפל מטריצות בוליאניות

בסעיף 31.3 הזכרנו שבאלגוריתם של שטראסן לכפל מטריצות לא ניתן להשתמש ישירות לכפל מטריצות בוליאניות, משום שהמעין־חוג הבוליאני $Q = (\{0, 1\}, \vee, \wedge, 0, 1)$ איננו חוג. אך משפט 3.10 הראה שאם משתמשים בפעולות אריתמטיות על מלים בנות $O(\lg n)$ סיביות, ניתן בכל זאת להשתמש בשיטת שטראסן כדי לכפול מטריצות בוליאניות בגודל $n \times n$ בזמן $O(n^{\lg 7})$.

בבעיה זו נחקור שיטה הסתברותית הערשה שימוש בפעולות על סיביות בלבד כדי להשיג חסם טוב כמעט כמו זה של שיטת שטראסן עם הסתברות קטנה לטעות.

א. הראה כי $R = (\{0, 1\}, \oplus, \wedge, 0, 1)$, כאשר $\oplus$ מציין את הפונקציה XOR (exclusive-or, או־מוציא), הוא חוג.

יהיו $A = (a_{ij})$ ו־$B = (b_{ij})$ מטריצות בוליאניות בגודל $n \times n$, ותהי $C = (c_{ij}) = AB$ במעין־חוג Q. ניצור את $A' = (a'_{ij})$ מ־A באמצעות השגרה האקראית הבאה:

- אם $a_{ij} = 0$, נציב $a'_{ij} = 0$.
- אם $a_{ij} = 1$, אזי בהסתברות $1/2$ נציב $a'_{ij} = 1$ ובהסתברות $1/2$ נציב $a'_{ij} = 0$. הבחירות האקראיות עבור כל תא הן בלתי תלויות.

ב. תהי $C' = (c'_{ij}) = A'B$ בחוג R. תראה כי $c_{ij} = 0$ גורר $c'_{ij} = 0$. הראה כי $c_{ij} = 1$ גורר $c'_{ij} = 1$ בהסתברות $1/2$.

ג. הראה שעבור כל $\epsilon > 0$, קיימת הסתברות של ϵ/n^2 לכל היותר ש־c'_{ij} נתון אינו מקבל את הערך c_{ij} באף אחת מ־$\lg(n^2/\epsilon)$ בחירות בלתי־תלויות של המטריצה A'. הראה שההסתברות שכל ה־c'_{ij} יקבלו את ערכיהם הנכונים לפחות פעם אחת היא לפחות $1 - \epsilon$.

ד. כתוב אלגוריתם אקראי שזמן ריצתו $O(n^{\lg 7} \lg n)$, המחשב את המכפלה של שתי מטריצות בגודל $n \times n$ במעין־חוג הבוליאני Q בהסתברות של $1 - 1/n^k$ לפחות עבור כל קבוע $k > 0$. הפעולות המותרות היחידות על איברי המטריצות הן $\wedge$, $\vee$ ו־$\oplus$.

31-2 מערכות תלת־אלכסוניות של משוואות לינאריות

נתונה מטריצה תלת־אלכסונית:

$$A = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}$$

א. מצא פירוק LU של A.

ב. פתור את המשוואה $Ax = (1\ 1\ 1\ 1\ 1)^{\mathrm{T}}$ באמצעות הצבות קדימה ואחורה.

ג. מצא את ההופכית של A.

ד. הראה שעבור כל מטריצה A בגודל $n \times n$ שהיא סימטרית, חיובית־לחלוטין ותלת־אלכסונית, ועבור כל וקטור b בגודל n, ניתן לפתור את המשוואה $Ax = b$ בזמן $O(n)$ על־ידי ביצוע פירוק LU. הראה שכל שיטה המבוססת על בניית A^{-1} תהיה יקרה יותר אסימפטוטית במקרה הגרוע.

ה. הראה שעבור כל מטריצה A בגודל $n \times n$, שהיא רגולרית ותלת־אלכסונית, ועבור כל וקטור b בגודל n, ניתן לפתור את המשוואה $Ax = b$ בזמן $O(n)$ על־ידי ביצוע פירוק LUP.

31-3 פונקציות ספליין

שיטה מעשית לביצוע אינטרפולציה של קבוצת נקודות באמצעות עקום היא להשתמש ב**פונקציות ספליין ממעלה שלישית** (cubic splines). נתונה לנו קבוצה $\{(x_i, y_i) : i = 0, 1, \ldots, n\}$ של $n + 1$ זוגות של שיעורי נקודות, כאשר $x_0 < x_1 < \cdots < x_n$. ברצוננו להתאים לנקודות עקום $f(x)$, המורכב מקטעים של עקומים ממעלה שלישית (פונקציית ספליין). כלומר, העקום

$f(x)$ מורכב מ־n פולינומים ממעלה שלישית; $f_i(x) = a_i + b_i x + c_i x^2 + d_i x^3$ עבור $i = 0, 1, \ldots, n-1$, כאשר, אם x נמצא בתחום $x_i \le x \le x_{i+1}$, אזי ערך העקום בנקודה x נתון על־ידי $f(x) = f_i(x - x_i)$. הנקודות x_i שבהן "מדביקים" זה לזה את הפולינומים מן המעלה השלישית נקראות **נקודות קשר** (knots). מטעמי פשטות, נניח כי $x_i = i$ עבור $i = 0, 1, \ldots, n$.

כדי להבטיח את רציפותה של $f(x)$ אנו דורשים כי עבור $i = 0, 1, \ldots, n-1$ יתקיים:

$$f(x_i) = f_i(0) = y_i$$
$$f(x_{i+1}) = f_i(1) = y_{i+1}$$

כדי להבטיח ש־$f(x)$ תהיה חלקה דיה, אנו דורשים גם רציפות של הנגזרת הראשונה בכל קשר:

$$f'(x_{i+1}) = f_i'(1) = f_{i+1}'(0)$$

עבור $i = 0, 1, \ldots, n-1$.

א. נניח שעבור $i = 0, 1, \ldots, n$, נתונים לנו לא רק זוגות שיעורי הנקודות $\{(x_i, y_i)\}$ אלא גם ערכי הנגזרות הראשונות $D_i = f'(x_i)$ בכל נקודת קשר. בטא כל אחד מהמקדמים a_i, b_i, c_i, ו־d_i במונחי הערכים y_i, y_{i+1}, D_i ו־D_{i+1}. (זכור כי $x_i = i$). באיזו מהירות ניתן לחשב את $4n$ המקדמים בהינתן זוגות שיעורי הנקודות וערכי הנגזרות הראשונות?

נותרת השאלה כיצד לבחור את הנגזרות הראשונות של $f(x)$ בנקודות הקשר. אחת השיטות היא לדרוש שהנגזרות השניות יהיו רציפות בנקודות הקשר:

$$f''(x_{i+1}) = f_i''(1) = f_{i+1}''(0)$$

עבור $i = 0, 1, \ldots, n-1$. בנקודות הקשר הראשונה והאחרונה, אנו מניחים כי $f''(x_0) = f_0''(0) = 0$ וכן $f''(x_n) = f_n''(1) = 0$; הנחות אלה הופכות את $f(x)$ לפונקציית ספליין **טבעית** מן המעלה השלישית.

ב. השתמש באילוצי הרציפות על הנגזרות השניות כדי להראות שעבור $i = 0, 1, \ldots, n-1$, מתקיים:

$$D_{i-1} + 4D_i + D_{i+1} = 3(y_{i+1} - y_{i-1}) \tag{31.35}$$

ג. הראה כי:

$$2D_0 + D_1 = 3(y_1 - y_0) \tag{31.36}$$

$$D_{n-1} + 2D_n = 3(y_n - y_{n-1}) \tag{31.37}$$

ד. כתוב מחדש את משוואות (31.35)-(31.37) כמשוואת מטריצות הכוללת את וקטור הנעלמים $D = \langle D_0, D_1, \ldots, D_n \rangle$. אילו מאפיינים יש למטריצה במשוואה שבנית?

ה. הראה כי ניתן לבצע אינטרפולציה של קבוצה בת $n+1$ זוגות של שיעורי נקודות באמצעות פונקציית ספליין טבעית מן המעלה השלישית בזמן $O(n)$ (ראה בעיה 31-2).

ו. הראה כיצד ניתן למצוא פונקציית ספליין טבעית מן המעלה השלישית לביצוע אינטרפולציה של קבוצה בת $n+1$ נקודות (x_i, y_i) המקיימות $x_0 < x_1 < \cdots < x_n$, גם כאשר x_i אינו בהכרח שווה ל־i. איזו משוואת מטריצות יש לפתור ומהו זמן הריצה של האלגוריתם שלך?

הערות ביבליוגרפיות לפרק 31

קיימים ספרים מצוינים רבים המתארים חישובים מדעיים ונומריים בפירוט רב בהרבה מזה שהתאפשר בספר זה. הספרים הבאים קריאים במיוחד: George & Liu [81], Golub & Van Loan [89], Press, Flannery, Teukolsky & Vetterling [162, 161] וכן Strang [181, 182].

פרסומו של האלגוריתם של שטראסן ב־1969 [183] עורר התרגשות רבה. קודם לכן, קשה היה להעלות על הדעת שניתן לשפר את האלגוריתם הנאיבי. מאז, שופר מאוד החסם העליון האסימפטוטי על מידת הקושי של כפל מטריצות. האלגוריתם היעיל ביותר אסימפטוטית המוכר כיום לכפל מטריצות בגודל $n \times n$ הוא של Coppersmith & Winograd [52] והוא רץ בזמן $O(n^{2.376})$. הייצוג הגרפי של האלגוריתם של שטראסן נעשה על־ידי Paterson [155]. Fischer & Meyer [67] התאימו את האלגוריתם של שטראסן למטריצות בוליאניות (משפט 31.10).

חילוץ גאוסיאני, שעליו מבוססים פירוקי LU ו־LUP, היה הדרך השיטתית הראשונה לפתרון מערכות של משוואות לינאריות. הוא היה גם אחד מן האלגוריתמים הנומריים המוקדמים ביותר. אף כי היה ידוע עוד קודם לכן, גילויו מיוחס בדרך כלל ל־C. F. Gauss (1777-1855). במאמרו המפורסם [183] גם הראה שטראסן שניתן למצוא את המטריצה ההופכית של מטריצה בגודל $n \times n$ בזמן $O(n^{\lg 7})$. Winograd [203] היה הראשון שהוכיח שכפל מטריצות אינו קשה יותר מהיפוך מטריצות; את הכיוון ההפוך הוכיחו Aho, Hopcroft & Ullman [4].

אצל Strang [182] מובאת הצגה מצוינת של מטריצות סימטריות וחיוביות־לחלוטין ושל אלגברה לינארית בכלל. בעמ' 334 מופיעה ההערה הבאה: "תלמידי שואלים תמיד על מטריצות *א־סימטריות חיוביות־לחלוטין*. אני לעולם איני משתמש במונח זה."

32 פולינומים והתמרת פוריה המהירה (FFT)

חיבור שני פולינומים ממעלה n בשיטה הישירה מתבצע בזמן $\Theta(n)$, אולם כפל שני פולינומים בשיטה הישירה מתבצע בזמן $\Theta(n^2)$. בפרק זה נראה כיצד התמרת פוריה המהירה (Fast Fourier Transform, או בקיצור FFT), יכולה לקצר ל־$\Theta(n \lg n)$ את הזמן לביצוע כפל פולינומים.

פולינומים

פולינום (polynomial) במשתנה x מעל שדה אלגברי F הוא פונקציה $A(x)$ שניתנת לייצוג באופן הבא:

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

את n אנו מכנים **החסם על מעלת הפולינום** (degree-bound) והערכים $a_0, a_1, \ldots, a_{n-1}$ נקראים **המקדמים** (coefficients) של הפולינום. המקדמים לקוחים מן השדה F, שהוא בדרך כלל קבוצת המספרים המרוכבים $\mathbf{C}$. אנו אומרים שפולינום $A(x)$ הוא ממעלה k אם המקדם הגבוה ביותר שלו השונה מאפס הוא a_k. המעלה של פולינום ממעלה החסומה על־ידי n, יכולה להיות כל מספר שלם בין 0 ל־$n-1$, כולל. ולהיפך, פולינום ממעלה k הוא פולינום ממעלה חסומה על־ידי n, עבור כל $n > k$.

קיים מגוון של פעולות שהיינו רוצים להגדיר עבור פולינומים. **חיבור פולינומים** (polynomial addition) מוגדר באופן הבא: אם $A(x)$ ו־$B(x)$ הם פולינומים ממעלה החסומה על־ידי n, אזי **הסכום** (sum) שלהם הוא פולינום $C(x)$, גם הוא ממעלה החסומה על־ידי n, המקיים $C(x) = A(x) + B(x)$ עבור כל x בשדה שמעליו מוגדרים הפולינומים. דהיינו, אם:

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

וכן:

$$B(x) = \sum_{j=0}^{n-1} b_j x^j$$

אזי:

$$C(x) = \sum_{j=0}^{n-1} c_j x^j$$

כאשר $c_j = a_j + b_j$ עבור $j = 0, 1, \ldots, n-1$. לדוגמה, אם:

$$A(x) = 6x^3 + 7x^2 - 10x + 9, \quad B(x) = -2x^3 + 4x - 5$$

אזי:

$$C(x) = 4x^3 + 7x^2 - 6x + 4$$

כפל פולינומים (polynomial multiplication) מוגדר באופן הבא: אם $A(x)$ ו־$B(x)$ הם פולינומים ממעלה החסומה על־ידי n, אזי **המכפלה** (product) שלהם היא הפולינום $C(x)$ ממעלה החסומה על־ידי $2n-1$, המקיים $C(x) = A(x)B(x)$ עבור כל x בשדה שמעליו מוגדרים הפולינומים. מן הסתם כבר כפלת פולינומים בעבר, על־ידי כך שכפלת כל איבר של $A(x)$ בכל איבר של $B(x)$, וחיברת איברים עם חזקות שוות. לדוגמה, ניתן לכפול את $A(x) = 6x^3 + 7x^2 - 10x + 9$ בפולינום $B(x) = -2x^3 + 4x - 5$ באופן הבא:

$$\begin{array}{r}
6x^3 + 7x^2 - 10x + 9 \\
-2x^3 + 4x - 5 \\
\hline
-30x^3 - 35x^2 + 50x - 45 \\
24x^4 + 28x^3 - 40x^2 + 36x \\
-12x^6 - 14x^5 + 20x^4 - 18x^3 \\
\hline
-12x^6 - 14x^5 + 44x^4 - 20x^3 - 75x^2 + 86x - 45
\end{array}$$

דרך אחרת לבטא את המכפלה $C(x)$ היא:

$$C(x) = \sum_{j=0}^{2n-2} c_j x^j \tag{32.1}$$

כאשר:

$$c_j = \sum_{k=0}^{j} a_k b_{j-k} \tag{32.2}$$

נשים לב כי מתקיים:

$$degree(C) = degree(A) + degree(B)$$

ומכאן נובע כי:

$$\begin{aligned} degree\text{-}bound(C) &= degree\text{-}bound(A) + degree\text{-}bound(B) - 1 \\ &\le degree\text{-}bound(A) + degree\text{-}bound(B) \end{aligned}$$

למרות זאת, נדבר על החסם על מעלתו של C כאילו היה סכום החסמים על מעלותיהם של A ו־B, שכן אם פולינום הוא ממעלה החסומה על־ידי k, אזי הוא גם ממעלה החסומה על־ידי $k+1$.

תמצית הפרק

סעיף 32.1 מציג שתי דרכים לייצוג פולינומים: ייצוג על־ידי מקדמים וייצוג בעזרת ערכי נקודות. כפל פולינומים בשיטות הישירות — משוואות (32.1) ו־(32.2) — מתבצע בזמן $\Theta(n^2)$ כאשר הפולינומים מיוצגים בעזרת מקדמים, אולם בזמן $\Theta(n)$ בלבד כאשר הם מיוצגים בעזרת ערכי נקודות. עם זאת, ניתן לכפול פולינומים המיוצגים בעזרת מקדמים בזמן $\Theta(n \lg n)$ בלבד

באמצעות המרות בין שתי שיטות הייצוג. אך כדי להיווכח מדוע ניתן לעשות זאת, עלינו ללמוד תחילה על השורשים המרוכבים של היחידה, וזאת נעשה בסעיף 32.2. לאחר מכן נשתמש בהתמרת פוריה המהירה ובהוכחתי שלה, אשר גם הם מתוארים בסעיף 32.2, לביצוע ההמרות בין שיטות הייצוג. בסעיף 32.3 נראה כיצד ניתן לממש במהירות את התמרת פוריה המהירה, הן במודלים סדרתיים והן במודלים מקביליים.

בפרק זה נעשה שימוש נרחב במספרים מרוכבים, ולכן נייחד את הסימן i לייצוג של $\sqrt{-1}$.

32.1 ייצוג פולינומים

ייצוגי פולינומים בעזרת מקדמים ובעזרת ערכי נקודות הם במובן מסוים ייצוגים שקולים; כלומר, לכל פולינום המיוצג בעזרת ערכי נקודות יש פולינום מקביל יחיד המיוצג בעזרת מקדמים. בסעיף זה נציג את שתי שיטות הייצוג ונראה כיצד ניתן לשלב ביניהן כדי לאפשר כפל של שני פולינומים ממעלה החסומה על־ידי n בזמן $\Theta(n \lg n)$.

ייצוג בעזרת מקדמים

ייצוג בעזרת מקדמים (coefficient representation) של פולינום $A(x) = \sum_{j=0}^{n-1} = a_j x^j$ ממעלה החסומה על־ידי n, הוא וקטור מקדמים $a = (a_0, a_1, \ldots, a_{n-1})$. במשוואות מטריצות שיופיעו בפרק זה נתייחס בדרך כלל לווקטורים כאל וקטורי עמודה.

הייצוג בעזרת מקדמים הוא נוח עבור פעולות מסוימות על פולינומים. לדוגמה, **מציאת הערך** של פולינום $A(x)$ בנקודה נתונה x_0 מתבצעת על־ידי חישוב הערך $A(x_0)$. מציאת הערך מתבצעת בזמן $\Theta(n)$ באמצעות **כלל הורנר** (Horner's rule):

$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-2} + x_0(a_{n-1}))\cdots))$$

באופן דומה, חיבור שני פולינומים המיוצגים בעזרת וקטורי המקדמים $a = (a_0, a_1, \ldots, a_{n-1})$ ו־$b = (b_0, b_1, \ldots, b_{n-1})$ מתבצע בזמן $\Theta(n)$: פשוט מחשבים את וקטור המקדמים $c = (c_0, c_1, \ldots, c_{n-1})$, שבו $c_j = a_j + b_j$ עבור $j = 0, 1, \ldots, n-1$.

עתה נתבונן בכפל של שני פולינומים $A(x)$ ו־$B(x)$ ממעלה החסומה על־ידי n המיוצגים בעזרת מקדמים. אם נשתמש בשיטה המתוארת על־ידי משוואות (32.1) ו־(32.2), יידרש זמן של $\Theta(n^2)$ לביצוע כפל הפולינומים, שכן יש לכפול כל אחד מהמקדמים בווקטור a בכל אחד מהמקדמים בווקטור b. הפעולה של כפל פולינומים המיוצגים בעזרת מקדמים נראית קשה בהרבה ממציאת ערכו של פולינום בנקודה או מחיבור שני פולינומים. וקטור המקדמים המתקבל c, הנתון במשוואה (32.2), נקרא גם **קונבולוציה** (convolution) של וקטורי הקלט a ו־b ומסומן על־ידי $c = a \otimes b$. מכיוון שכפל פולינומים וחישוב קונבולוציות הם בעיות חישוביות בסיסיות בעלות חשיבות מעשית ניכרת, מתמקד פרק זה באלגוריתמים יעילים לפתרונן.

ייצוג בעזרת ערכי נקודות

ייצוג בעזרת ערכי נקודות (point-value representation) של פולינום $A(x)$ ממעלה החסומה על־ידי n הוא קבוצה של n **זוגות של ערכי נקודות** (point-value pairs):

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$$

כך שכל ה־x_k שונים זה מזה, ועבור $k = 0, 1, \ldots, n-1$ מתקיים:

$$y_k = A(x_k) \tag{32.3}$$

לכל פולינום קיימים ייצוגים שונים רבים בעזרת ערכי נקודות, שכן כל קבוצה של n נקודות שונות זו מזו, $x_0, x_1, \ldots, x_{n-1}$, יכולה לשמש בסיס לייצוג.

חישוב ייצוג בעזרת ערכי נקודות של פולינום המיוצג בעזרת מקדמים הוא בעיקרון חישוב ישיר, שכן כל שעלינו לעשות הוא לבחור n נקודות שונות זו מזו $x_0, x_1, \ldots, x_{n-1}$, ואז למצוא את הערך של $A(x_k)$ עבור $k = 0, 1, \ldots, n-1$. בשיטת הורנר, חישוב ערכו של הפולינום ב־n נקודות מתבצע בזמן $\Theta(n^2)$. בהמשך נראה שאם בוחרים את הנקודות x_k בצורה נבונה, ניתן לקצר את זמן החישוב ל־$\Theta(n \lg n)$.

הפעולה ההפוכה למציאת ערכי הפולינום — חישוב הייצוג בעזרת מקדמים בהינתן ייצוג בעזרת ערכי נקודות — נקרא **אינטרפולציה** (interpolation). המשפט הבא מראה שאינטרפולציה מוגדרת היטב, בהנחה שהחסם על מעלתו של הפולינום המתקבל מאינטרפולציה שווה למספר הזוגות הנתונים של ערכי הנקודות.

משפט 32.1 (יחידותו של פולינום המתקבל מאינטרפולציה)
עבור כל קבוצה $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$ של n זוגות של ערכי נקודות, קיים פולינום יחיד $A(x)$ ממעלה החסומה על־ידי n המקיים $y_k = A(x_k)$ עבור $k = 0, 1, \ldots, n-1$.

הוכחה ההוכחה מבוססת על קיום המטריצה ההופכית של מטריצה מסוימת. משוואה (32.3) שקולה למשוואת המטריצות:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} \tag{32.4}$$

המטריצה השמאלית מסומנת על־ידי $V(x_0, x_1, \ldots, x_{n-1})$ וידועה בשם מטריצת ואנדרמונדה (Vandermonde). על פי תרגיל 31.1-10, הדטרמיננטה של מטריצה זו היא:

$$\prod_{j<k} (x_k - x_j)$$

ולכן, על פי משפט 31.5, המטריצה הפיכה (כלומר, רגולרית) אם ה־x_k שונים זה מזה. לכן, בהינתן הייצוג בעזרת ערכי נקודות, ניתן לחשב פתרון יחיד של המקדמים a_j:

$$a = V(x_0, x_1, \ldots, x_{n-1})^{-1} y \qquad \blacksquare$$

ההוכחה של משפט 32.1 מתארת אלגוריתם עבור אינטרפולציה המבוסס על פתרון מערכת המשוואות הלינאריות (32.4). באמצעות האלגוריתמים המבצעים פירוק־LU מפרק 31, ניתן לפתור משוואות אלה בזמן $O(n^3)$.

אלגוריתם מהיר יותר לביצוע אינטרפולציה של n נקודות מבוסס על **נוסחת לאגראנז'** (Lagrange's formula):

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)} \tag{32.5}$$

הקורא מוזמן לוודא שאגף ימין של משוואה (32.5) הוא פולינום ממעלה החסומה על־ידי n והוא מקיים $A(x_k) = y_k$ עבור כל k. בתרגיל 31.1-4 תתבקש למצוא כיצד ניתן לחשב את המקדמים של A באמצעות נוסחת לאגראנז' בזמן $\Theta(n^2)$.

אם כן, מציאת הערכים ב־n נקודות ואינטרפולציה הן שתי פעולות הופכיות מוגדרות היטב להמרה בין ייצוג של פולינום בעזרת מקדמים לבין ייצוג בעזרת ערכי נקודות[1]. האלגוריתמים שתוארו לעיל לפתרון בעיות אלה מתבצעים בזמן $\Theta(n^2)$.

הייצוג בעזרת ערכי נקודות הוא נוח למדי עבור פעולות רבות על פולינומים. עבור חיבור, אם $C(x) = A(x) + B(x)$, אזי $C(x_k) = A(x_k) + B(x_k)$ עבור כל נקודה x_k. ביתר דיוק, בהינתן ייצוג של A בעזרת ערכי נקודות:

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$$

וייצוג כזה של B:

$$\{(x_0, y_0'), (x_1, y_1'), \ldots, (x_{n-1}, y_{n-1}')\}$$

(יש לשים לב שערכיהם של A ושל B נתונים *באותן* n נקודות), אזי הייצוג של C בעזרת ערכי נקודות הוא:

$$\{(x_0, y_0 + y_0'), (x_1, y_1 + y_1'), \ldots, (x_{n-1}, y_{n-1} + y_{n-1}')\}$$

הזמן לחיבור שני פולינומים ממעלה החסומה על־ידי n המיוצגים בעזרת ערכי נקודות הוא אפוא $\Theta(n)$.

באופן דומה, הייצוג בעזרת ערכי נקודות הוא נוח עבור כפל פולינומים. אם $C(x) = A(x)B(x)$, אזי $C(x_k) = A(x_k)B(x_k)$ עבור כל נקודה x_k, וניתן לכפול נקודתית את הייצוג של A בעזרת ערכי נקודות בייצוג של B בעזרת ערכי נקודות, ולקבל ייצוג של C בעזרת ערכי נקודות. אולם, עומדת בפנינו הבעיה שהחסם על מעלתו של C שווה לסכום החסמים על מעלותיהם של A ושל B. ייצוג מקובל של A ושל B על־ידי ערכי נקודות מורכב מ־n זוגות של ערכי נקודות לכל פולינום. כפל הייצוגים נקודה נקודה נותן לנו n זוגות של ערכי נקודות עבור C, אולם מאחר שהחסם על מעלתו של C הוא $2n$, הרי שממשפט 32.1 נובע שאנו זקוקים ל־$2n$ זוגות של ערכי נקודות כדי לייצג את C בעזרת ערכי נקודות. לכן עלינו להתחיל עם ייצוגים "מורחבים" של A ושל B שכל אחד מהם מורכב מ־$2n$ זוגות של ערכי נקודות. בהינתן ייצוג מורחב של A בעזרת ערכי נקודות:

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{2n-1}, y_{2n-1})\}$$

וייצוג מורחב מתאים של B בעזרת ערכי נקודות:

$$\{(x_0, y_0'), (x_1, y_1'), \ldots, (x_{2n-1}, y_{2n-1}')\}$$

1. אינטרפולציה ידועה לשמצה בבעייתיות שלה מבחינת היציבות המספרית. אף כי הנושאות המתוארות כאן נכונות מבחינה מתמטית, הפרשים קטנים בקלטים או טעויות עיגול במהלך החישוב עלולים לגרום להפרשים גדולים בתוצאות.

אזי ייצוג בעזרת ערכי נקודות של C הוא:

$$\{(x_0, y_0 y_0'), (x_1, y_1 y_1'), \ldots, (x_{2n-1}, y_{2n-1} y_{2n-1}')\}$$

בהינתן כקלט שני פולינומים המיוצגים בייצוג מורחב בעזרת ערכי נקודות, אנו רואים כי הזמן לקבלת ייצוג בעזרת ערכי נקודות של המכפלה הוא $\Theta(n)$, זמן קצר בהרבה מזה הנדרש כדי לכפול פולינומים המיוצגים בעזרת מקדמים.

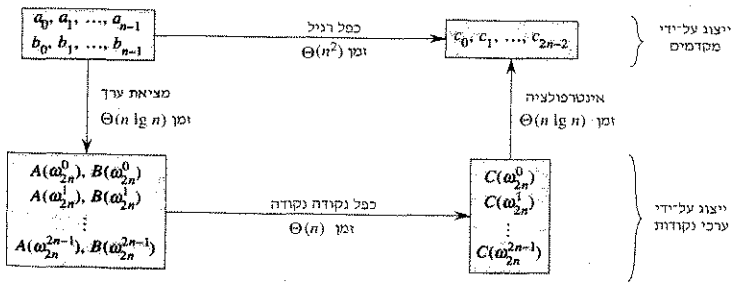
לבסוף נדון בשאלה כיצד מוצאים את ערכו של פולינום המיוצג בעזרת ערכי נקודות, בנקודה חדשה. עבור בעיה זו, נראה שאין דרך פשוטה יותר אלא לבצע תחילה המרה לייצוג בעזרת מקדמים, ואז לחשב את ערכו של הפולינום בנקודה החדשה.

כפל מהיר של פולינומים המיוצגים בעזרת מקדמים

האם נוכל להשתמש בשיטת הזמן הלינארי לכפל פולינומים המיוצגים בעזרת ערכי נקודות כדי להחיש כפל פולינומים המיוצגים בעזרת מקדמים? התשובה תלויה ביכולתנו להמיר פולינום במהירות משיטת הייצוג בעזרת מקדמים לשיטת הייצוג בעזרת ערכי נקודות (מציאת הערך) ולהיפך (אינטרפולציה).

אנו יכולים להשתמש בנקודות כלשהן כדי לייצג פולינום בשיטת ערכי הנקודות, אך בחירה נבונה של נקודות הייצוג תאפשר לבצע המרה בין הייצוגים בזמן $\Theta(n \lg n)$ בלבד. כפי שנראה בסעיף 32.2, אם נבחר את "שורשי היחידה המרוכבים" כנקודות הערכים, נוכל ליצור ייצוג בעזרת ערכי נקודות על־ידי הפעלת התמרת פוריה הבדידה (Discrete Fourier Transform, או בקיצור DFT) על וקטור מקדמים. את הפעולה ההפוכה, אינטרפולציה, ניתן לבצע על־ידי הפעלה של "DFT ההופכי" על זוגות של ערכי נקודות, ולקבל וקטור מקדמים. בסעיף 32.2 נראה כיצד FFT מבצע את DFT ואת DFT ההופכי בזמן $\Theta(n \lg n)$.

איור 32.1 מדגים בצורה גרפית את האסטרטגיה הזאת. פרט קטן אחד נוגע לחסמים על מעלות הפולינומים. המכפלה של שני פולינומים ממעלה החסומה על־ידי n היא פולינום ממעלה החסומה על־ידי $2n$. לכן, לפני מציאת ערכיהם של פולינומי הקלט A ו־B, אנו מכפילים תחילה את החסם על מעלתם ל־$2n$ על־ידי הוספת n מקדמים מסדר גבוה השווים ל־0. מכיוון שהווקטורים מכילים $2n$ איברים, אנו משתמשים בשורשי היחידה המרוכבים מסדר $2n$, המסומנים על־ידי הביטויים ω_{2n} באיור 32.1.



איור 32.1 תרשים גרפי של תהליך יעיל לכפל פולינומים. הייצוגים העליונים הם על־ידי מקדמים ואילו התחתונים — על־ידי ערכי נקודות. החיצים משמאל לימין מציינים פעולות כפל. הביטויים ω_{2n} הם שורשי היחידה המרוכבים מסדר $2n$.

בהינתן FFT, השגרה שלהלן, שזמן ריצתה $\Theta(n \lg n)$, כופלת שני פולינומים $A(x)$ ו־$B(x)$ ממעלה החסומה על־ידי n, כאשר הייצוגים של הקלט והפלט הם בשיטת המקדמים. אנו מניחים ש־n הוא חזקה של 2; את הדרישה הזאת ניתן תמיד למלא על־ידי הוספת מקדמים מסדר גבוה השווים ל־0.

1. *הכפל את החסם על המעלה:* יצירת ייצוגים בעזרת מקדמים של $A(x)$ ושל $B(x)$ כפולינומים ממעלה החסומה על־ידי $2n$ על־ידי הוספה של n מקדמים מסדר גבוה השווים לאפס לכל אחד מהם.
2. *מצא ערכים:* חישוב ייצוגים על־ידי ערכי נקודות באורך $2n$ של $A(x)$ ו־$B(x)$ באמצעות שתי הפעלות של FFT מסדר $2n$. ייצוגים אלה מכילים את ערכי שני הפולינומים בשורשי היחידה מסדר $2n$.
3. *כפול נקודתית:* חישוב ייצוג בעזרת ערכי נקודות של הפולינום $C(x) = A(x)B(x)$ על־ידי כפל ערכים אלה בכל נקודה. ייצוג זה מורכב מערכי $C(x)$ בכל אחד משורשי היחידה המרוכבים מסדר $2n$.
4. *בצע אינטרפולציה:* צור את הייצוג בעזרת מקדמים של הפולינום $C(x)$ על־ידי הפעלה יחידה של FFT על $2n$ הזוגות של ערכי נקודות לצורך חישוב DFT ההופכי.

צעדים (1) ו־(3) מתבצעים בזמן $\Theta(n)$, וצעדים (2) ו־(4) מתבצעים בזמן $\Theta(n \lg n)$. לכן, ברגע שנראה כיצד להשתמש ב־FFT, הרי שהוכחנו את המשפט הבא.

משפט 32.2
כפל שני פולינומים ממעלה החסומה על־ידי n ניתן לביצוע בזמן $\Theta(n \lg n)$ כאשר הייצוגים של הקלט והפלט הם בשיטת המקדמים. ■

תרגילים

32.1-1
כפול את הפולינומים $A(x) = 7x^3 - x^2 + x - 10$ ו־$B(x) = 8x^3 - 6x + 3$ באמצעות משוואות (32.1) ו־(32.2).

32.1-2
את מציאת ערכו של פולינום $A(x)$ ממעלה החסומה על־ידי n בנקודה נתונה x_0 ניתן לבצע גם על־ידי חלוקת $A(x)$ בפולינום $(x - x_0)$ לקבלת פולינום מנה $q(x)$ ממעלה החסומה על־ידי $n - 1$ ושארית r, המקיימים:

$$A(x) = q(x)(x - x_0) + r$$

אפשר לראות בבירור כי $A(x_0) = r$. הראה כיצד מחשבים את השארית r ואת המקדמים של $q(x)$ בזמן $\Theta(n)$, בהינתן x_0 והמקדמים של A.

32.1-3
מצא ייצוג בעזרת ערכי נקודות של $A^{rev}(x) = \sum_{j=0}^{n-1} = a_{n-1-j}x^j$ מתוך ייצוג בעזרת ערכי נקודות של $A(x) = \sum_{j=0}^{n-1} = a_j x^j$, בהנחה שכל הנקודות שונות מ־0.

32.1-4
הראה כיצד ניתן להשתמש במשוואה (32.5) לביצוע אינטרפולציה בזמן $\Theta(n^2)$. **רמז:** תחילה חשב את $\prod_j (x - x_j)$ ואז חלק ב־$(x - x_k)$ כנדרש עבור המונה של כל איבר. ראה תרגיל 31.1-1.

32.1-5
הסבר מה שגוי בגישה "המובנת מאליה" לחילוק פולינומים המיוצגים בעזרת ערכי נקודות. דון בנפרד במקרה שבו הפולינומים מתחלקים בדיוק ובמקרה שבו הם אינם מתחלקים בדיוק.

32.1-6
נתונות שתי קבוצות A ו־B, שכל אחת מהן מכילה n שלמים בתחום 0 עד $10n$. ברצוננו לחשב את **הסכום הקרטזי** (Cartesian sum) של A ו־B, המוגדר על־ידי:

$$C = \{x + y : y \in B \text{ ו־} x \in A\}$$

שים לב שהשלמים ב־C הם בתחום 0 עד $20n$. ברצוננו למצוא את איברי C ואת מספר הפעמים שכל איבר ב־C מתקבל כסכום איברים מ־A ומ־B. הראה שניתן לפתור את הבעיה בזמן $O(n \lg n)$. (**רמז:** ייצג את A ו־B כפולינומים ממעלה $10n$.)

32.2 DFT ו־FFT

בסעיף 32.1 טענו שאם משתמשים בשורשי היחידה המרוכבים, ניתן למצוא ערכי פולינום ולבצע אינטרפולציה בזמן $\Theta(n \lg n)$. בסעיף זה נגדיר שורשי היחידה המרוכבים ונלמד את תכונותיהם, נגדיר את ה־DFT, ונראה כיצד ה־FFT מחשב את ה־DFT ואת ההופכי שלו בזמן $\Theta(n \lg n)$ בלבד.

שורשי היחידה המרוכבים

שורש מרוכב מסדר n של היחידה (complex nth root of unity) הוא מספר מרוכב ω המקיים:

$$\omega^n = 1$$

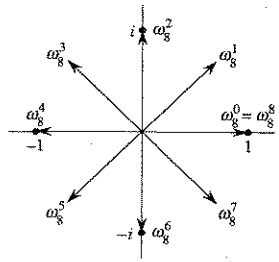
קיימים בדיוק n שורשים מרוכבים מסדר n של היחידה; אלה הם הערכים $e^{2\pi ik/n}$ עבור $k = 0, 1, \ldots, n-1$. כדי לפרש את הנוסחה הזאת, נשתמש בהגדרה של מעריך של מספר מרוכב:

$$e^{iu} = \cos(u) + i \sin(u)$$

כפי שניתן לראות באיור 32.2, n שורשי היחידה המרוכבים ממוקמים במרווחים שווים מסביב למעגל בעל רדיוס יחידה שמרכזו בראשיתו של המישור המרוכב. הערך:

$$\omega_n = e^{2\pi i/n} \quad (32.6)$$

נקרא **שורש היחידה הראשי מסדר n** (the principal nth root of unity); כל שאר שורשי היחידה המרוכבים מסדר n הם חזקות של ω_n.



איור 32.2 הערכים של $\omega_8^0, \omega_8^1, \ldots, \omega_8^7$ במישור המרוכב, כאשר $\omega_8 = e^{2\pi i/8}$ הוא שורש היחידה הראשי מסדר 8.

n שורשי היחידה המרוכבים מסדר n:

$$\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$$

יוצרים חבורה תחת כפל (ראה סעיף 33.3). חבורה זו היא בעלת אותו מבנה כמו החבורה החיבורית $(\mathbf{Z}_n, +)$ מודולו n, שכן $\omega_n^n = \omega_n^0 = 1$ גורר $\omega_n^j \omega_n^k = \omega_n^{j+k} = \omega_n^{(j+k) \bmod n}$. באופן דומה, $\omega_n^{-1} = \omega_n^{n-1}$. תכונות מהותיות של שורשי היחידה המרוכבים מסדר n נתונות בלמות הבאות.

למה 32.3 (למת הצמצום)
עבור שלמים כלשהם $n \geq 0$, $k \geq 0$ ו־$d > 0$, מתקיים:

$$\omega_{dn}^{dk} = \omega_n^k \tag{32.7}$$

הוכחה הלמה נובעת ישירות ממשוואה (32.6), שכן:

$$\begin{aligned} \omega_{dn}^{dk} &= (e^{2\pi i/dn})^{dk} \\ &= (e^{2\pi i/n})^k \\ &= \omega_n^k \end{aligned}$$

■

מסקנה 32.4
עבור כל מספר שלם זוגי $n > 0$, מתקיים:

$$\omega_n^{n/2} = \omega_2 = -1$$

הוכחה הקורא יתבקש להוכיח את הלמה בתרגיל 32.2-1. ■

למה 32.5 (למת המחצית)
אם $n > 0$ הוא מספר זוגי, אזי הריבועים של n שורשי היחידה המרוכבים מסדר n הם $n/2$ שורשי היחידה המרוכבים מסדר $n/2$.

הוכחה על פי למת הצמצום, $(\omega_n^k)^2 = \omega_{n/2}^k$ עבור כל שלם אי־שלילי k. נשים לב שאם נעלה בריבוע את כל שורשי היחידה המרוכבים מסדר n, אזי כל שורש מסדר $n/2$ יתקבל בדיוק פעמיים, שכן:

$$\begin{aligned} (\omega_n^{k+n/2})^2 &= \omega_n^{2k+n} \\ &= \omega_n^{2k}\omega_n^n \\ &= \omega_n^{2k} \\ &= (\omega_n^k)^2 \end{aligned}$$

הריבועים של ω_n^k ושל $\omega_n^{k+n/2}$ שווים אפוא. ניתן להוכיח תכונה זו גם באמצעות מסקנה 32.4, שכן $\omega_n^{n/2} = -1$ גורר $\omega_n^{k+n/2} = -\omega_n^k$, ולכן, $(\omega_n^{k+n/2})^2 = (\omega_n^k)^2$. ■

כפי שנראה, למת המחצית חיונית לגישת הפרד־ומשול שננקוט בה להמרה בין ייצוגי פולינומים בעזרת מקדמים ובעזרת ערכי נקודות, שכן היא מבטיחה שגודלן של התת־הבעיות הרקורסיביות יקטן למחצית.

למה 32.6 (למת הסכום)
עבור כל שלם $n \geq 1$ וכל שלם אי־שלילי k שאינו מתחלק ב־n, מתקיים:

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = 0$$

הוכחה מכיוון שמשוואה (3.3) מתקיימת עבור ערכים מרוכבים, אזי:

$$\begin{aligned} \sum_{j=0}^{n-1} (\omega_n^k)^j &= \frac{(\omega_n^k)^n - 1}{\omega_n^k - 1} \\ &= \frac{(\omega_n^n)^k - 1}{\omega_n^k - 1} \\ &= \frac{(1)^k - 1}{\omega_n^k - 1} \\ &= 0 \end{aligned}$$

הדרישה ש־k לא יתחלק ב־n מבטיחה שהמכנה שונה מאפס, שכן $\omega_n^k = 1$ רק כאשר k מתחלק ב־n. ■

ה־DFT

נזכור כי ברצוננו למצוא את ערכיו של פולינום:

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

ממעלה החסומה על־ידי n, בנקודות^1 $\omega_n^0, \omega_n^1, \omega_n^2, \ldots, \omega_n^{n-1}$ (כלומר ב־n שורשי היחידה המרוכבים מסדר n)^2. בלי הגבלת הכללות, נניח כי n הוא חזקה של 2, שכן תמיד ניתן להגדיל את החסם על המעלה — תמיד נוכל להוסיף מקדמים חדשים מסדר גבוה השווים לאפס — לפי הצורך. אנו מניחים כי A נתון בייצוג על־ידי מקדמים: $a = (a_0, a_1, \ldots, a_{n-1})$. הבה נגדיר את התוצאות y_k, עבור $k = 0, 1, \ldots, n-1$, על־ידי:

$$\begin{aligned} y_k &= A(\omega_n^k) \\ &= \sum_{j=0}^{n-1} a_j \omega_n^{kj} \end{aligned} \tag{32.8}$$

הווקטור $y = (y_0, y_1, \ldots, y_{n-1})$ הוא **התמרת פוריה הבדידה** (Discrete Fourier Transform, ובקיצור DFT) של וקטור המקדמים $a = (a_0, a_1, \ldots, a_{n-1})$. נסמן גם $y = \mathrm{DFT}_n(a)$.

ה־FFT

באמצעות שיטה הידועה בשם **התמרת פוריה המהירה** (Fast Fourier Transform, ובקיצור FFT), אשר מנצלת את תכונותיהם המיוחדות של שורשי היחידה המרוכבים, ניתן לחשב את $\mathrm{DFT}_n(a)$ בזמן $\Theta(n \lg n)$, לעומת הזמן $\Theta(n^2)$ הנדרש בשיטה הישירה.

שיטת ה־FFT נוקטת באסטרטגיית הפרד־ומשול; היא מפרידה בין המקדמים בעלי האינדקסים הזוגיים לבין המקדמים בעלי האינדקסים האי־זוגיים של $A(x)$, כדי להגדיר שני פולינומים חדשים ממעלה החסומה על־ידי $n/2$, והם $A^{[0]}(x)$ ו־$A^{[1]}(x)$:

$$\begin{aligned} A^{[0]}(x) &= a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{n/2-1} \\ A^{[1]}(x) &= a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{n/2-1} \end{aligned}$$

נשים לב כי $A^{[0]}$ מכיל את כל המקדמים בעלי האינדקסים הזוגיים של A (הייצוג הבינרי של האינדקס משתיים ב־0), ו־$A^{[1]}$ מכיל את כל המקדמים בעלי האינדקסים האי־זוגיים של A (הייצוג הבינרי של האינדקס משתיים ב־1). מכך נובע כי:

$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2) \tag{32.9}$$

כך שהבעיה של מציאת ערכי $A(x)$ בנקודות $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$ מצטמצמת לבעיה הבאה:

1. מציאת ערכי הפולינומים $A^{[0]}(x)$ ו־$A^{[1]}(x)$, שמעלתם חסומה על־ידי $n/2$, בנקודות:

$$(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2 \tag{32.10}$$

ואז:

2. צירוף התוצאות על פי משוואה (32.9).

על פי למת המחצית, רשימת הערכים (32.10) אינה מורכבת מ־n ערכים שונים אלא רק מ־$n/2$ שורשי היחידה המרוכבים מסדר $n/2$, כאשר כל שורש מופיע בדיוק פעמיים. לכן, מציאת ערכי הפולינומים $A^{[0]}$ ו־$A^{[1]}$, שמעלתם חסומה על־ידי $n/2$, ב־$n/2$ שורשי היחידה המרוכבים מסדר $n/2$, נעשית באופן רקורסיבי. תת־בעיות אלה הן בעלות מבנה זהה לחלוטין לזה של הבעיה המקורית, אולם גודלן הוא מחצית מגודלה. הצלחנו אפוא לחלק חישוב DFT_n על n איברים לשני חישובי $\mathrm{DFT}_{n/2}$ על $n/2$ איברים. פירוק זה הוא הבסיס לאלגוריתם ה־FFT הרקורסיבי שלהלן, המחשב את ה־DFT של וקטור בן n איברים $a = (a_0, a_1, \ldots, a_{n-1})$, כאשר n הוא חזקה של 2.

2. האורך n כאן הוא למעשה $2n$ במונחי סעיף 32.1, שכן אנו מכפילים את החסם על מעלתם של הפולינומים הנתונים לפני חישוב ערכיהם. בהקשר של כפל פולינומים, אנו עובדים למעשה עם שורשי היחידה המרוכבים מסדר $2n$.


```
RECURSIVE-FFT(a)
1   n ← length[a]                          ▷ n is a power of 2.
2   if n = 1
3       then return a
4   ω_n ← e^{2πi/n}
5   ω ← 1
6   a^[0] ← (a_0, a_2, ..., a_{n-2})
7   a^[1] ← (a_1, a_3, ..., a_{n-1})
8   y^[0] ← RECURSIVE-FFT(a^[0])
9   y^[1] ← RECURSIVE-FFT(a^[1])
10  for k ← 0 to n/2 − 1
11      do y_k ← y_k^[0] + ω y_k^[1]
12         y_{k+(n/2)} ← y_k^[0] − ω y_k^[1]
13         ω ← ω ω_n
14  return y                               ▷ y is assumed to be column vector.
```

השגרה RECURSIVE-FFT פועלת באופן הבא: שורות 2-3 מייצגות את בסיס הרקורסיה; ה־DFT של איבר אחד הוא האיבר עצמו, שכן במקרה זה:

$$\begin{aligned} y_0 &= a_0 \omega_1^0 \\ &= a_0 \cdot 1 \\ &= a_0 \end{aligned}$$

שורות 6-7 מגדירות את וקטורי המקדמים של הפולינומים $A^{[0]}$ ו־$A^{[1]}$. שורות 4, 5 ו־13 מבטיחות ש־ω יעודכן כהלכה, כך שבכל פעם שמבוצעות שורות 11-12, יתקיים $\omega = \omega_n^k$. (שמירת ערכו המעודכן של ω מאיטרציה לאיטרציה חוסכת את הזמן הנדרש לחישובו של ω_n^k מחדש בכל ביצוע של לולאת ה־**for**.) שורות 8-9 מבצעות את חישובי ה־$\mathrm{DFT}_{n/2}$ הרקורסיביים ומציבות, עבור $k = 0, 1, \ldots, n/2 - 1$:

$$\begin{aligned} y_k^{[0]} &= A^{[0]}(\omega_{n/2}^k) \\ y_k^{[1]} &= A^{[1]}(\omega_{n/2}^k) \end{aligned}$$

או, מכיוון ש־$\omega_{n/2}^k = \omega_n^{2k}$ על פי למת הצמצום:

$$\begin{aligned} y_k^{[0]} &= A^{[0]}(\omega_n^{2k}) \\ y_k^{[1]} &= A^{[1]}(\omega_n^{2k}) \end{aligned}$$

שורות 11-12 מצרפות את תוצאות חישובי ה־$\mathrm{DFT}_{n/2}$ הרקורסיביים. עבור $y_0, y_1, \ldots, y_{n/2-1}$, שורה 11 מניבה:

$$\begin{aligned} y_k &= y_k^{[0]} + \omega_n^k y_k^{[1]} \\ &= A^{[0]}(\omega_n^{2k}) + \omega_n^k A^{[1]}(\omega_n^{2k}) \\ &= A(\omega_n^k) \end{aligned}$$

כאשר השורה האחרונה בטיעון זה נובעת ממשוואה (32.9). עבור $y_{n/2}, y_{n/2+1}, \ldots, y_{n-1}$, כאשר $k = 0, 1, \ldots, n/2 - 1$, שורה 12 מניבה:

$$\begin{aligned} y_{k+(n/2)} &= y_k^{[0]} - \omega_n^k y_k^{[1]} \\ &= y_k^{[0]} + \omega_n^{k+(n/2)} y_k^{[1]} \\ &= A^{[0]}(\omega_n^{2k}) + \omega_n^{k+(n/2)} A^{[1]}(\omega_n^{2k}) \\ &= A^{[0]}(\omega_n^{2k+n}) + \omega_n^{k+(n/2)} A^{[1]}(\omega_n^{2k+n}) \\ &= A(\omega_n^{k+(n/2)}) \end{aligned}$$

השורה השנייה נובעת מן הראשונה מכיוון ש־$\omega_n^{k+(n/2)} = -\omega_n^k$. השורה הרביעית נובעת מן השלישית מכיוון ש־$\omega_n^n = 1$ גורר $\omega_n^{2k} = \omega_n^{2k+n}$. השורה האחרונה נובעת ממשוואה (32.9). לכן, הווקטור y שמחזירה RECURSIVE-FFT הוא אכן ה־DFT של וקטור הקלט a.

כדי למצוא את זמן הריצה של השגרה RECURSIVE-FFT, נשים לב שלבד מן הקריאות הרקורסיביות, כל ביצוע אורך זמן $\Theta(n)$, כאשר n הוא אורכו של וקטור הקלט. נוסחת הנסיגה עבור זמן הריצה היא אפוא:

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \lg n) \end{aligned}$$

לכן, בהינתן פולינום ממעלה החסומה על־ידי n, ניתן לחשב את ערכיו ב־n שורשי היחידה המורכבים מסדר n באמצעות התמרת פוריה המהירה בזמן $\Theta(n \lg n)$.

אינטרפולציה בשורשי היחידה המרוכבים

עתה נשלים את שיטת כפל הפולינומים על־ידי כך שנראה כיצד מבצעים אינטרפולציה של שורשי היחידה המרוכבים על־ידי פולינום, דבר שיאפשר לנו להמיר ייצוג בעזרת ערכי נקודות בחזרה לייצוג בעזרת מקדמים. אנו מבצעים אינטרפולציה על־ידי כתיבת ה־DFT כמשוואת מטריצות, ואז התבוננות בצורתה של המטריצה ההופכית.

על פי משוואה (32.4), ניתן לכתוב את ה־DFT כמכפלת המטריצות $y = V_n a$, כאשר V_n הינה מטריצת ואנדרמונדה המורכבת מן החזקות המתאימות של ω_n:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \omega_n^3 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \omega_n^6 & \cdots & \omega_n^{2(n-1)} \\ 1 & \omega_n^3 & \omega_n^6 & \omega_n^9 & \cdots & \omega_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \omega_n^{3(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

האיבר ה־(k, j) של V_n הוא ω_n^{kj}, עבור $j, k = 0, 1, \ldots, n-1$, והמעריכים של איברי V_n יוצרים לוח כפל.

את הפעולה ההפוכה, המשומנת על־ידי $a = \mathrm{DFT}_n^{-1}(y)$, אנו מבצעים על־ידי כפל y במטריצה V_n^{-1}, המטריצה ההופכית של V_n.

משפט 32.7
עבור $j, k = 0, 1, \ldots, n-1$, האיבר ה־(k, j) של V_n^{-1} הוא ω_n^{-kj}/n.

הוכחה אנו נראה כי $V_n^{-1}V_n = I_n$, כאשר I_n היא מטריצת היחידה בגודל $n \times n$. נתבונן באיבר ה־(j, j') של $V_n^{-1}V_n$:

$$\begin{aligned} [V_n^{-1}V_n]_{jj'} &= \sum_{k=0}^{n-1} (\omega_n^{-kj}/n)(\omega_n^{kj'}) \\ &= \sum_{k=0}^{n-1} \omega_n^{k(j'-j)}/n \end{aligned}$$

על פי למת הסכום (למה 32.6), סכום זה שווה ל־1 אם $j' = j$, ול־0 אחרת. יש לזכור שאנו מסתמכים על כך שמתקיים $-(n-1) < j' - j < n-1$, כך ש־$(j'-j)$ אינו מתחלק ב־n, וניתן להחיל את למת הסכום. ■

בהינתן המטריצה ההופכית V_n^{-1}, נקבל כי $\mathrm{DFT}_n^{-1}(y)$ נתון על־ידי:

$$a_j = \frac{1}{n} \sum_{k=0}^{n-1} y_k \omega_n^{-kj} \qquad (32.11)$$

עבור $j = 0, 1, \ldots, n-1$. מהשוואות המשוואות (32.8) ו־(32.11) עולה שניתן לחשב את ה־DFT ההופכי על־ידי הכנסת השינויים הבאים באלגוריתם ה־FFT: החלפת התפקידים של a ו־y, החלפת ω_n ב־ω_n^{-1}, וחלוקת כל איבר של התוצאה ב־n (ראה תרגיל 32.4). לכן, גם את DFT_n^{-1} ניתן לחשב בזמן $\Theta(n \lg n)$.

אם כן, באמצעות ה־FFT וה־FFT ההופכי נוכל לבצע המרות של פולינום ממעלה החסומה על־ידי n הלוך וחזור בין ייצוג על־ידי מקדמים לבין ייצוג על־ידי ערכי נקודות בזמן $\Theta(n \lg n)$. בהקשר של כפל פולינומים, הראינו בכך את הטענה שלהלן:

משפט 32.8 (משפט הקונבולוציה)
עבור כל שני וקטורים a ו־b באורך n, כאשר n הוא חזקה של 2, מתקיים:

$$a \otimes b = \mathrm{DFT}_{2n}^{-1}(\mathrm{DFT}_{2n}(a) \cdot \mathrm{DFT}_{2n}(b))$$

כאשר לווקטורים a ו־b מוסיפים אפסים עד לאורך $2n$, ו־$\cdot$ מציין מכפלה איבר איבר של שני הווקטורים בעלי $2n$ האיברים. ■

תרגילים

32.2-1
הוכח את מסקנה 32.4.

32.2-2
חשב את ה־DFT של הווקטור $(0, 1, 2, 3)$.

32.2-3
חזור על תרגיל 31.1-1 תוך שימוש בשיטה המתבצעת בזמן $\Theta(n \lg n)$.

32.2-4

כתוב פסאודו־קוד לחישוב DFT_n^{-1} בזמן $\Theta(n \lg n)$.

32.2-5

תאר הכללה של שיטת ה־FFT למקרה שבו n הוא חזקה של 3. מצא נוסחת נסיגה עבור זמן הריצה ופתור אותה.

* 32.2-6

נניח שבמקום לבצע FFT על n איברים מעל שדה המספרים המרוכבים (כאשר n זוגי), אנו משתמשים ב־Z_m, שהוא חוג השלמים מודולו m, כאשר $m = 2^{tn/2} + 1$ ו־t הוא שלם חיובי כלשהו. השתמש ב־$\omega = 2^t$ במקום ω_n כשורש היחידה הראשי מסדר n, מודולו m. הוכח שה־DFT וה־DFT ההופכי מוגדרים היטב במערכת זו.

32.2-7

בהינתן רשימת ערכים $z_0, z_1, \ldots, z_{n-1}$ (אולי עם חזרות), הראה כיצד מוצאים את מקדמי הפולינום $P(x)$ ממעלה החסומה על־ידי n שמקבל את הערך 0 רק בנקודות $z_0, z_1, \ldots, z_{n-1}$. השגרה שלך צריכה לרוץ בזמן $O(n \lg^2 n)$. (**רמז:** הפולינום $P(x)$ מקבל ערך 0 בנקודה z_j אם ורק אם $P(x)$ הוא כפולה של $(x - z_j)$.)

* 32.2-8

התמרת chirp (chirp transform) **של וקטור** $a = (a_0, a_1, \ldots, a_{n-1})$ היא וקטור $y = (y_0, y_1, \ldots, y_{n-1})$, כאשר $y_k = \sum_{j=0}^{n-1} a_j z^{kj}$ ו־z הוא מספר מרוכב כלשהו. ה־DFT הוא אפוא מקרה פרטי של התמרת chirp המתקבל כאשר $z = \omega_n$. הוכח שעבור כל מספר מרוכב z, ניתן לחשב את התמרת chirp בזמן $O(n \lg n)$. (**רמז:** השתמש במשוואה:

$$y_k = z^{k^2/2} \sum_{j=0}^{n-1} (a_j z^{j^2/2})\,(z^{-(k-j)^2/2})$$

כדי לראות את התמרת chirp כקונבולוציה.)

32.3 מימושים יעילים של ה־FFT

מכיוון שהיישומים המעשיים של ה־DFT, כגון עיבוד אותות, דורשים מהירות גדולה ביותר, נבחן בסעיף זה שני מימושים יעילים של ה־FFT. תחילה נבחן גרסה איטרטיבית של אלגוריתם ה־FFT שרצה בזמן $\Theta(n \lg n)$, אולם הקבוע החבוי בסימון Θ קטן מזה של המימוש הרקורסיבי מסעיף 32.2. לאחר מכן נשתמש ברעיונות שהובילו אותנו למימוש האיטרטיבי כדי לתכנן מעגל FFT מקבילי יעיל.

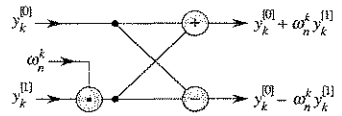
מימוש איטרטיבי של ה־FFT

תחילה נשים לב כי בלולאת ה־**for** שבשורות 10-13 של RECURSIVE-FFT מחשבים פעמיים את הערך $\omega_n^k y_k^{[1]}$. במונחי הידור (קומפילציה), ערך זה נקרא **תת־ביטוי משותף**

(common subexpression). ניתן לשנות את הלולאה כך שתחשב את הערך פעם אחת בלבד ותאחסן אותו במשתנה זמני t:

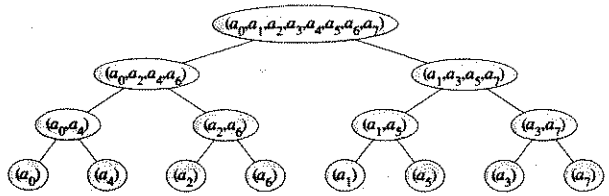
```
for k ← 0 to n/2 − 1
    do t ← ω y_k^[1]
       y_k ← y_k^[0] + t
       y_{k+(n/2)} ← y_k^[0] − t
       ω ← ω ω_n
```

הפעולה שמבצעת לולאה זו — כפל ω (ששווה ל־ω_n^k) ב־$y_k^{[1]}$, אחסון המכפלה ב־t, הוספת t ל־$y_k^{[0]}$ וחיסורו ממנו — ידועה בשם **פעולת פרפר** (butterfly operation) והיא מוצגת באופן סכמטי באיור 32.3.



איור 32.3 פעולת פרפר. שני ערכי הקלט נכנסים משמאל, ω_n^k מוכפל ב־$y_k^{[1]}$, והסכום וההפרש מתקבלים כפלטים מימין. את האיור ניתן לפרש כמעגל צירופי.

נראה עתה כיצד לשנות את אלגוריתם ה־FFT כך שיהיה בעל מבנה איטרטיבי ולא רקורסיבי. באיור 32.4 סידרנו את וקטורי הקלט לקריאות הרקורסיביות במהלך הרצה של RECURSIVE-FFT במבנה של עץ, כאשר הקריאה ההתחלתית היא עבור $n = 8$. העץ מכיל צומת אחד עבור כל קריאה רקורסיבית של השגרה, ובתוך כל צומת רשום וקטור הקלט המתאים. בכל קריאה ל־RECURSIVE-FFT מתבצעות שתי קריאות רקורסיביות, אלא אם כן וקטור הקלט הוא בן איבר אחד בלבד. הקריאה הרקורסיבית הראשונה היא הבן השמאלי, והקריאה השנייה היא הבן הימני.



איור 32.4 עץ וקטורי הקלט לקריאות הרקורסיביות של השגרה RECURSIVE-FFT. הקריאה ההתחלתית היא עבור $n = 8$.

התבוננות בעץ מלמדת שאילו יכולנו לסדר את איברי הווקטור ההתחלתי a בסדר שבו הם מופיעים בעלים, ניתן היה לחקות את ביצועה של השגרה RECURSIVE-FFT באופן הבא: תחילה, מחלקים את האיברים לזוגות, מחשבים את ה־DFT של כל זוג באמצעות פעולת פרפר אחת, ומחליפים את הזוג ב־DFT שלו. בשלב זה הווקטור מורכב מ־$n/2$ וקטורי DFT בני 2 איברים כל אחד. לאחר מכן, מחלקים את $n/2$ וקטורי DFT אלה לזוגות, מחשבים את ה־DFT של ארבעת איברי הווקטור שמהם נוצר כל זוג באמצעות שתי פעולות פרפר, ומחליפים כל שני וקטורי DFT בני 2 איברים בווקטור DFT אחד בן ארבעה איברים. הווקטור כולו מורכב בשלב זה מ־$n/4$ וקטורי DFT בני 4 איברים כל אחד. ממשיכים באופן דומה עד שהווקטור מכיל שני וקטורי DFT בני $n/2$ איברים כל אחד, ואותם מצרפים באמצעות $n/2$ פעולות פרפר ל־DFT הסופי בן n האיברים.

כדי להפוך תיאור זה לקוד, אנו משתמשים במערך $A[0..n-1]$ המכיל במצב ההתחלתי את איברי וקטור הקלט a בסדר שבו הם מופיעים בעלים של העץ מאיור 32.4. (בהמשך נראה כיצד קובעים את הסדר הזה.) מכיוון שהצירוף חייב להיעשות בכל רמה של העץ, אנו מוסיפים משתנה s למניית הרמות, מרמה 1 (בתחתית העץ, כאשר מצרפים זוגות איברים ליצירת וקטורי DFT בני 2 איברים) עד $\lg n$ (בראש העץ, כאשר מצרפים שני וקטורי DFT בני $n/2$ איברים לקבלת התוצאה הסופית). האלגוריתם הוא אפוא בעל המבנה הבא:

```
1  for s ← 1 to lg n
2      do for k ← 0 to n − 1 by 2^s
3             do combine the two 2^(s−1)-element DFT's in
                   A[k..k + 2^(s−1) − 1] and A[k + 2^(s−1)..k + 2^s − 1]
                   into one 2^s-element DFT in A[k..k + 2^s − 1]
```

את גוף הלולאה (שורה 3) נוכל לבטא על־ידי פסאודו־קוד מדויק יותר: נעתיק את לולאת ה־**for** מן השגרה RECURSIVE-FFT, תוך זיהוי $y^{[0]}$ עם $A[k..k + 2^{s-1} - 1]$ ו־$y^{[1]}$ עם $A[k + 2^{s-1}..k + 2^s - 1]$. הערך של ω בכל פעולת פרפר תלוי בערכו של s; אנו משתמשים ב־ω_m, כאשר $m = 2^s$ (את המשתנה m הוספנו רק כדי שהקוד יהיה קריא יותר). אנו מגדירים משתנה זמני נוסף u, המאפשר לנו לבצע את פעולת הפרפר בְּמָקוֹם. לאחר החלפת שורה 3 של המבנה הסכמטי בלולאה עצמה מקבלים את הפסאודו־קוד שלהלן, שהוא הבסיס לאלגוריתם FFT האיטרטיבי הסופי שלנו, כמו גם למימוש המקבילי שנציג מאוחר יותר.

```
FFT-BASE(a)
1  n ← length[a]                          ▷ n is a power of 2.
2  for s ← 1 to lg n
3      do m ← 2^s
4         ω_m ← e^(2πi/m)
5         for k ← 0 to n − 1 by m
6             do ω ← 1
7                for j ← 0 to m/2 − 1
8                    do t ← ω A[k + j + m/2]
9                       u ← A[k + j]
10                      A[k + j] ← u + t
11                      A[k + j + m/2] ← u − t
12                      ω ← ω ω_m
```

עתה נציג את הגרסה הסופית לקוד של אלגוריתם FFT האיטרטיבי שלנו, ההופכת את שתי הלולאות הפנימיות כדי לחסוך כמה חישובי אינדקסים, וכן משתמשת בשגרת עזר BIT-REVERSE-COPY(a, A) כדי להעתיק את הווקטור a למערך A בסדר ההתחלתי שבו דרושים לנו הערכים.


```
ITERATIVE-FFT(a)
1  BIT-REVERSE-COPY(a, A)
2  n ← length[a]                          ▷ n is a power of 2.
3  for s ← 1 to lg n
4      do m ← 2^s
5         ω_m ← e^(2πi/m)
6         ω ← 1
7         for j ← 0 to m/2 − 1
8             do for k ← j to n − 1 by m
9                    do t ← ω A[k + m/2]
10                      u ← A[k]
11                      A[k] ← u + t
12                      A[k + m/2] ← u − t
13                ω ← ω ω_m
14 return A
```

כיצד מסדרת BIT-REVERSE-COPY את איברי וקטור הקלט a במערך A בסדר הרצוי? הסדר שבו מופיעים העלים באיור 32.4 הוא הסדר המתקבל מהיפוך הסיביות בייצוג הבינרי. כלומר, אם נסמן ב־$\mathrm{rev}(k)$ את השלם בן $\lg n$ הסיביות אשר נוצר על־ידי היפוך סדר הסיביות בייצוג הבינרי של k, אזי ברצוננו לאחסן את האיבר a_k בתא $A[\mathrm{rev}(k)]$ במערך. באיור 32.4, לדוגמה, העלים מופיעים בסדר (משמאל לימין) 0, 4, 2, 6, 1, 5, 3, 7, ובכתיב בינרי (שוב, משמאל לימין): 000, 100, 010, 110, 001, 101, 011, 111. על־ידי היפוך סדר הסיביות בייצוג הבינרי של כל מספר מתקבלת הסדרה: 000, 001, 010, 011, 100, 101, 110, 111. כדי לראות כי גם במקרה הכללי ברצוננו לקבל סדר בינרי בהיפוך סיביות, נשים לב כי ברמה העליונה של העץ, אינדקסים בעלי סיבית אחרונה (הכי פחות משמעותית) השווה ל־0 ממוקמים בתת־עץ השמאלי, ואינדקסים שהסיבית האחרונה שלהם היא 1 ממוקמים בתת־עץ הימני. אם מסלקים את הסיבית הכי הפחות משמעותית בכל רמה וממשיכים בתהליך זה במורד העץ, מתקבל בעלים הסדר הבינרי בהיפוך סיביות.

מכיוון שהפונקציה $\mathrm{rev}(k)$ ניתנת לחישוב בנקל, ניתן לכתוב את השגרה BIT-REVERSE-COPY באופן הבא:

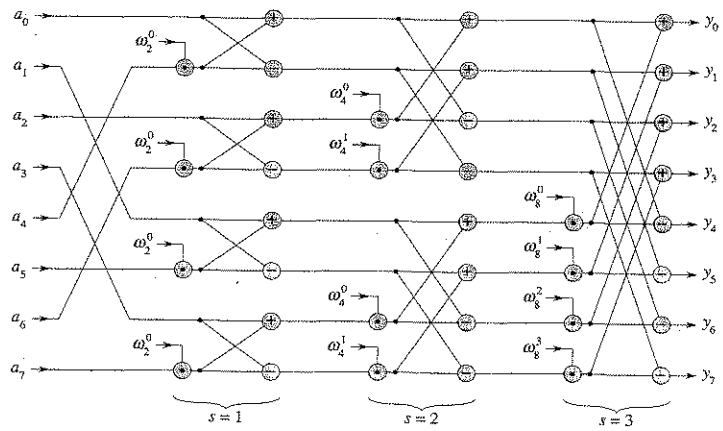
```
BIT-REVERSE-COPY(a, A)
1  n ← length[a]
2  for k ← 0 to n − 1
3      do A[rev(k)] ← a_k
```

המימוש האיטרטיבי של ה־FFT רץ בזמן $\Theta(n \lg n)$. הקריאה BIT-REVERSE-COPY(a, A) רצה בזמן $O(n \lg n)$, שכן מתבצעות n איטרציות, והיפוך סדר הסיביות של מספר שלם בין 0 ל־$n-1$, המיוצג על־ידי $\lg n$ סיביות, מתבצע בזמן $O(\lg n)$. (ביישומים מעשיים, ערכו ההתחלתי של n ידוע לנו בדרך כלל מראש, כך שניתן ליצור טבלה הממפה את k ל־$\mathrm{rev}(k)$ ולקצר בכך את זמן הריצה של BIT-REVERSE-COPY ל־$\Theta(n)$ עם קבוע חבוי קטן. לחלופין, ניתן להשתמש בשיטה המתוחכמת של המונה הבינרי בהיפוך סיביות מבעיה 1-18, המבוססת על ניתוח לשיעורין.) להשלמת ההוכחה שהשגרה ITERATIVE-FFT רצה בזמן $\Theta(n \lg n)$, נראה כי $L(n)$, מספר הפעמים שמתבצעת הלולאה הפנימית ביותר (שורות 9-21), הוא $\Theta(n \lg n)$:

$$
\begin{aligned}
L(n) &= \sum_{s=1}^{\lg n} \sum_{j=0}^{2^{s-1}-1} \frac{n}{2^s} \\
&= \sum_{s=1}^{\lg n} \frac{n}{2^s} \cdot 2^{s-1} \\
&= \sum_{s=1}^{\lg n} \frac{n}{2} \\
&= \Theta(n \lg n)
\end{aligned}
$$

מעגל FFT מקבילי

ניתן לנצל רבות מן התכונות שאפשרו לנו לממש אלגוריתם FFT איטרטיבי יעיל גם ליצירת אלגוריתם מקבילי יעיל עבור ה־FFT. (ראה תיאור של מודל המעגל הצירופי בפרק 29.) המעגל הצירופי PARALLEL-FFT המחשב את ה־FFT של n קלטים מוצג באיור 32.5 עבור $n = 8$. המעגל מתחיל בביצוע תמורת היפוך סיביות של הקלטים, ואחריה $\lg n$ שלבים, כאשר כל שלב מורכב מ־$n/2$ פעולות פרפר המתבצעות במקביל. עומקו של המעגל הוא אפוא $\Theta(\lg n)$.



איור 32.5 מעגל צירופי PARALLEL-FFT המחשב את ה־FFT מוצג כאן עבור $n = 8$ קלטים. השלבים של פעולות הפרפר מסומנים בהתאם לאיטרציות של הלולאה החיצונית ביותר בשגרה FFT-BASE. ניתן לחשב FFT של n קלטים באמצעות מעגל בעומק $\Theta(\lg n)$ המורכב מ־$\Theta(n \lg n)$ רכיבים צירופיים.

החלק השמאלי ביותר של המעגל PARALLEL-FFT מבצע את תמורת היפוך הסיביות, ושאר המעגל מחקה את השגרה האיטרטיבית FFT-BASE. אנו מנצלים את העובדה שכל איטרציה של לולאת ה־**for** החיצונית ביותר מבצעת $n/2$ פעולות פרפר בלתי־תלויות שאפשר לבצען במקביל. ערכו של s בכל איטרציה בשגרה FFT-BASE מתאים לשלב של פעולות פרפר במעגל שבאיור 32.5. כל שלב s, עבור $s = 1, 2, \ldots, \lg n$, מכיל $n/2^s$ קבוצות של פעולות פרפר (בהתאם לערכי k ב־FFT-BASE), כאשר כל קבוצה מכילה 2^{s-1} פעולות פרפר (בהתאם לערכי j ב־FFT-BASE). פעולות הפרפר שבאיור 32.5 מתאימות לפעולות הפרפר המבוצעות בלולאה הפנימית ביותר

(שורות 8-11) של FFT-BASE. יש לשים לב שערכי ω המשמשים בפעולות הפרפר במעגל מתאימים לאלה המשמשים ב־FFT-BASE: בשלב s משתמשים ב־$\omega_m^0, \omega_m^1, \ldots, \omega_m^{m/2-1}$, כאשר $m = 2^s$.

תרגילים

32.3-1

הראה כיצד ITERATIVE-FFT מחשבת את ה־DFT של וקטור הקלט $(0, 2, 3, -1, 4, 5, 7, 9)$.

32.3-2

הראה כיצד ניתן לממש אלגוריתם FFT שבו תמורת היפוך הסיביות מופיעה בסוף החישוב ולא בתחילתו. (**רמז:** בחן את ה־DFT ההופכי.)

32.3-3

כמה רכיבי חיבור, חיסור וכפל וכמה תילים נחוצים במעגל PARALLEL-FFT שתואר בסעיף זה כדי לחשב את DFT_n? (הנח שנחוץ תיל אחד בלבד להעברת מספר ממקום אחד לאחר.)

32.3-4 *

נניח שבסוכמים במעגל FFT חלה לעתים תקלה הגורמת לכך שהפלט שלהם הוא תמיד אפס, בלי תלות בקלטים. נניח שחלה תקלה כזאת בסוכם אחד בדיוק, אולם אינך יודע באיזה מהם. תאר כיצד תוכל לזהות את הסוכם הפגום על־ידי הזנת קלטים למעגל ה־FFT כולו ובחינת הפלטים המתקבלים. נסה לעשות זאת באופן יעיל.

בעיות

32-1 כפל בשיטת הפרד־ומשול

א. הראה כיצד ניתן לכפול שני פולינומים לינאריים $ax + b$ ו־$cx + d$ באמצעות שלוש פעולות כפל בלבד. (**רמז:** אחת מפעולות הכפל היא $(a + b) \cdot (c + d)$.)

ב. כתוב שני אלגוריתמי הפרד־ומשול אשר כופלים שני פולינומים ממעלה החסומה על־ידי n בזמן $\Theta(n^{\lg 3})$. האלגוריתם הראשון צריך לחלק את המקדמים של פולינומי הקלט למחצית מסדר גבוה ומחצית מסדר נמוך, והאלגוריתם השני צריך לחלק אותם למקדמים בעלי אינדקסים זוגיים ולמקדמים בעלי אינדקסים אי־זוגיים.

ג. הראה שניתן לכפול שני שלמים בני n סיביות ב־$O(n^{\lg 3})$ צעדים, כאשר בכל צעד פועלים על מספר קבוע לכל היותר של ערכים בני סיביות אחת.

32-2 מטריצות טפליץ (Toeplitz)

מטריצת טפליץ (Toeplitz matrix) היא מטריצה $A = (a_{ij})$ בגודל $n \times n$ אשר בה $a_{ij} = a_{i-1,j-1}$ עבור $i = 2, 3, \ldots, n$ ו־$j = 2, 3, \ldots, n$.

א. האם מטריצת הסכום של שתי מטריצות טפליץ היא בהכרח מטריצת טפליץ? מה בדבר מטריצת המכפלה?

ב. תאר ייצוג של מטריצת טפליץ המאפשר חיבור שתי מטריצות טפליץ בגודל $n \times n$ בזמן $O(n)$.

ג. כתוב אלגוריתם הכופל מטריצת טפליץ בגודל $n \times n$ בווקטור באורך n בזמן $O(n \lg n)$. השתמש בייצוג מסעיף (ב).

ד. כתוב אלגוריתם יעיל לכפל שתי מטריצות טפליץ בגודל $n \times n$. נתח את זמן הריצה של האלגוריתם שלך.

32-3 מציאת ערכי כל נגזרותיו של פולינום בנקודה

בהינתן פולינום $A(x)$ ממעלה החסומה על־ידי n, הנגזרת ה־t של הפולינום מוגדרת על־ידי:

$$A^{(t)}(x) = \begin{cases} A(x) & t = 0 \quad \text{אם} \\ \frac{d}{dx} A^{(t-1)}(x) & 1 \le t \le n-1 \quad \text{אם} \\ 0 & t \ge n \quad \text{אם} \end{cases}$$

בהינתן ייצוג של $A(x)$ על־ידי מקדמים $(a_0, a_1, \ldots, a_{n-1})$ ונקודה x_0, ברצוננו למצוא את $A^{(t)}(x_0)$ עבור $t = 0, 1, \ldots, n-1$.

א. בהינתן המקדמים $b_0, b_1, \ldots, b_{n-1}$ כך שמתקיים:

$$A(x) = \sum_{j=0}^{n-1} b_j (x - x_0)^j$$

הראה כיצד מחשבים את $A^{(t)}(x_0)$, עבור $t = 0, 1, \ldots, n-1$, בזמן $O(n)$.

ב. הסבר כיצד מוצאים את $b_0, b_1, \ldots, b_{n-1}$ בזמן $O(n \lg n)$, בהינתן $A(x_0 + \omega_n^k)$ עבור $k = 0, 1, \ldots, n-1$.

ג. הוכח כי:

$$A(x_0 + \omega_n^k) = \sum_{j=0}^{n-1} \left(\frac{\omega_n^{kr}}{r!} \sum_{j=0}^{n-1} f(j) g(r-j) \right)$$

כאשר $f(j) = a_j \cdot j!$ ו־$g(l)$ מוגדר על־ידי:

$$g(l) = \begin{cases} x_0^{-l}/(-l)! & -(n-1) \le l \le 0 \quad \text{אם} \\ 0 & 1 \le l \le (n-1) \quad \text{אם} \end{cases}$$

ד. הסבר כיצד מחשבים את $A(x_0 + \omega_n^k)$ עבור $k = 0, 1, \ldots, n-1$ בזמן $O(n \lg n)$. הסק שניתן למצוא את ערכן של כל הנגזרות הלא טריוויאליות של $A(x)$ בנקודה x_0 בזמן $O(n \lg n)$.

32-4 מציאת ערכו של פולינום בנקודות רבות

ראינו כי הבעיה של מציאת ערכו של פולינום ממעלה החסומה על־ידי $n-1$ בנקודה יחידה, ניתנת לפתרון בזמן $O(n)$ באמצעות כלל הורנר. כמו כן, גילינו שניתן למצוא את ערכיו של פולינום כזה בכל n השורשים המרוכבים של היחידה בזמן $O(n \lg n)$, באמצעות ה־FFT. עתה נראה כיצד ניתן למצוא את ערכיו של פולינום ממעלה החסומה על־ידי n ב־n נקודות כלשהן בזמן $O(n \lg^2 n)$.

כדי לעשות זאת, ננצל את העובדה שניתן לחשב את פולינום השארית המתקבל מחלוקת פולינום כזה בפולינום אחר כזה, בזמן $O(n \lg n)$. זוהי תוצאה שאנו מניחים ואיננו מוכיחים. לדוגמה,

השארית המתקבלת מחלוקת $3x^3 + x^2 - 3x + 1$ ב־$x^2 + x + 2$ היא:

$$(3x^3 + x^2 - 3x + 1) \bmod (x^2 + x + 2) = -7x + 5$$

בהינתן הייצוג על־ידי מקדמים של הפולינום $A(x) = \sum_{k=0}^{n-1} = a_k x^k$ ו־n נקודות $x_0, x_1, \ldots, x_{n-1}$, ברצוננו לחשב את n הערכים $A(x_0), A(x_1), \ldots, A(x_{n-1})$. עבור $0 \le i \le j \le n-1$, נגדיר את הפולינומים $P_{ij}(x) = \prod_{k=i}^{j} (x - x_k)$ ו־$Q_{ij} = A(x) \bmod P_{ij}(x)$. יש לשים לב כי Q_{ij} הוא פולינום ממעלה החסומה על־ידי $j - i$ לכל היותר.

א. הוכח כי $A(x) \bmod (x - z) = A(z)$ עבור כל נקודה z.

ב. הוכח כי $Q_{k,k}(x) = A(x_k)$ וכי $Q_{0,n-1}(x) = A(x)$.

ג. הוכח כי עבור $i \le k \le j$, מתקיים $Q_{ik}(x) = Q_{ij}(x) \bmod P_{ik}(x)$ וגם $Q_{kj}(x) = Q_{ij}(x) \bmod P_{kj}(x)$.

ד. כתוב אלגוריתם שרץ בזמן $O(n \lg^2 n)$ למציאת הערכים $A(x_0), A(x_1), \ldots, A(x_{n-1})$.

32-5 FFT באמצעות אריתמטיקה מודולרית

התמרת פוריה הבדידה, כפי שהוגדרה בפרק זה, דורשת שימוש במספרים מרוכבים, ודבר זה עלול לגרום לאובדן דיוק בשל טעויות עיגול. ישנן בעיות שעבורן ידוע כי התשובה מכילה מספרים שלמים בלבד, ולכן, כדי להבטיח חישוב מדויק של התוצאה, רצוי להשתמש בגרסה שונה של ה־FFT המבוססת על אריתמטיקה מודולרית. דוגמה לבעיה כזאת היא כפל שני פולינומים בעלי מקדמים שלמים. תרגיל 32.3-6 מציג גישה אחת לפתרון בעיה זו, העושה שימוש במודולוס (modulus) באורך $\Omega(n)$ סיביות כדי לטפל ב־DFT על n נקודות. בבעיה זו נציג גישה נוספת, העושה שימוש במודולוס באורך הסביר יותר, $O(\lg n)$; גישה זו דורשת הבנה של החומר המוצג בפרק 33. יהי n חזקה של 2.

א. נניח שאנו מחפשים את ה־k הקטן ביותר שעבורו $p = kn + 1$ הוא ראשוני. הבא נימוק היוריסטי פשוט מדוע אנו יכולים לצפות שערכו של k יהיה בקירוב $\lg n$. (ערכו של k עשוי להיות קטן בהרבה או גדול בהרבה, אבל סביר לצפות שנבחן בממוצע $O(\lg n)$ ערכים מועמדים עבור k.) מהו היחס בין תוחלת אורכו של p לבין אורכו של n?

יהי g יוצר של $\mathbf{Z}_p^*$ ויהי $\omega = g^k \bmod p$.

ב. הראה כי ה־DFT וה־DFT ההופכי הם פעולות הפוכות מוגדרות היטב מודולו p, כאשר ω משמש כשורש היחידה הראשי מסדר n.

ג. הראה כי ניתן לגרום לכך שה־FFT וההופכי שלו יפעלו מודולו p בזמן $O(\lg n)$, כאשר פעולות על מילים בנות $O(\lg n)$ סיביות מתבצעות ביחידת זמן אחת. הנח שהאלגוריתם מקבל את p ואת ω.

ד. חשב את DFT מודולו $p = 17$ של הווקטור $(0, 5, 3, 7, 7, 2, 1, 6)$. שים לב כי $g = 3$ הוא היוצר של $\mathbf{Z}_{17}^*$.

הערות ביבליוגרפיות לפרק 32

אצל Press, Flannery, Teukolsky & Vetterling [161, 162] מובא תיאור טוב של התמרת פוריה המהירה ושל יישומיה. אצל Oppenheim & Willsky [153] אפשר למצוא מבוא מצוין לעיבוד אותות, תחום יישום פופולרי של ה־FFT.

Cooley & Tukey [51] מוכרים בדרך כלל כמי שפיתחו בשנות ה־60' את ה־FFT. למעשה, ה־FFT פותח פעמים רבות קודם לכן, אולם חשיבותו לא הובנה במלואה לפני בניינם של המחשבים הספרתיים המודרניים. Press, Flannery, Teukolssky & Vetterling מייחסים את מקורות השיטה ל־Runge & König (1924).

33 אלגוריתמים בתורת המספרים

תורת המספרים נתפשה בעבר כנושא מתמטי טהור, יפהפה אך חסר תועלת מעשית. כיום נעשה שימוש נרחב באלגוריתמים המבוססים על תורת המספרים, בין השאר בשל המצאתן של שיטות הצפנה המבוססות על מספרים ראשוניים גדולים. השימושיות של שיטות אלה נובעת מיכולתנו למצוא בקלות מספרים ראשוניים גדולים, ואילו בטיחות השיטות נובעת מאי יכולתנו לפרק לגורמים מכפלה של מספרים ראשוניים גדולים. בפרק זה נציג כמה אלגוריתמים בתורת המספרים ואלגוריתמים קרובים, העומדים בבסיסם של יישומים כאלה.

בסעיף 33.1 נציג מושגים בסיסיים בתורת המספרים, כגון חלוקה ללא שארית, שקילות מודולרית ופירוק יחיד לגורמים. בסעיף 33.2 נלמד אחד מן האלגוריתמים העתיקים ביותר בעולם: האלגוריתם של אויקלידס למציאת המחלק המשותף הגדול ביותר של שני שלמים. בסעיף 33.3 נסקור מושגים באריתמטיקה מודולרית. בסעיף 33.4 נלמד את קבוצת הכפולות של מספר נתון a, מודולו n, ונראה כיצד מוצאים את כל הפתרונות של המשוואה $ax \equiv b \pmod{n}$ באמצעות האלגוריתם של אויקלידס. בסעיף 33.5 נציג את משפט השאריות הסיני ובסעיף 33.6 נבחן חזקות של מספר נתון a, מודולו n, ונציג אלגוריתם אשר, בהינתן a, b ו־n, מחשב ביעילות את $a^b \bmod n$ באמצעות העלאה חוזרת בריבוע. חישוב זה הוא לב לבו של מבחן ראשוניות יעיל. בסעיף 33.7 נתאר את מערכת ההצפנה RSA, המבוססת על מפתח ציבורי. בסעיף 33.8 נתאר מבחן ראשוניות אקראי שניתן להשתמש בו כדי למצוא ביעילות מספרים ראשוניים גדולים, שלב חיוני ביצירת מפתחות עבור שיטת ההצפנה RSA. לבסוף, בסעיף 33.9, נסקור היוריסטיקה פשוטה אך יעילה עבור פירוק לגורמים של מספרים שלמים קטנים. עובדה מעניינת היא, שפירוק לגורמים הוא אחת הבעיות שהיינו רוצים שתישאר בלתי פתירה, שכן בטיחותה של שיטת RSA תלויה בקושי לפרק לגורמים מספרים שלמים גדולים.

גודל קלטים והעלות של חישובים אריתמטיים

מאחר שבפרק זה נעבוד עם מספרים שלמים גדולים, עלינו לשנות במעט את דרך חשיבתנו על הגודל של קלט ועל עלותן של הפעולות האריתמטיות הבסיסיות.

בפרק זה, "קלט גדול" פירושו בדרך כלל קלט המורכב מ"שלמים גדולים" ולאו דווקא קלט הכולל "שלמים רבים" (כמו במיון, למשל). לכן נמדוד את גודלו של קלט במונחי *מספר הסיביות* הדרושות לייצוג הקלט, ולא נסתפק במספר השלמים שהקלט כולל. אלגוריתם שהקלטים שלו הם השלמים $a_1, a_2, \ldots, a_k$ הוא **אלגוריתם זמן־פולינומיאלי** (polynomial-time algorithm) אם זמן ריצתו הוא פולינום ב־$\lg a_1, \lg a_2, \ldots, \lg a_k$, כלומר, פולינום באורכי הייצוגים הבינריים של הקלטים.

ברובו של ספר זה נוח היה לנו לחשוב על הפעולות האריתמטיות היסודיות (כפל, חילוק, חישוב שארית) כפעולות יסוד שכל אחת מהן מתבצעת ביחידת זמן אחת. מניית מספר הפעולות האריתמטיות מסוג זה שמבצע אלגוריתם היתה הבסיס להערכה סבירה של זמן הריצה בפועל של

האלגוריתם על מחשב. אולם פעולות יסודיות יכולות לצרוך זמן רב כאשר הקלטים גדולים. משום כך נוח יותר למנות כמה **פעולות על סיביות** (bit operations) דורש אלגוריתם המבוסס על תורת המספרים. במודל זה, כפל שני שלמים שכל אחד מהם בן β סיביות בשיטה הרגילה מתבצע באמצעות $\Theta(\beta^2)$ פעולות על סיביות. באופן דומה, פעולת החילוק של שלם בן β סיביות בשלם קצר ממנו, או פעולת חישוב השארית המתקבלת בחלוקת שלם בן β סיביות בשלם קצר ממנו, ניתנות לביצוע בזמן $\Theta(\beta^2)$ באמצעות אלגוריתמים פשוטים. (ראה תרגיל 33.1-1.) ידועות שיטות מהירות יותר לביצוע חישובים אלה. לדוגמה, שיטת הפרד־ומשול פשוטה לכפל שני שלמים בני β סיביות רצה בזמן $\Theta(\beta^{\log_2 3})$, והשיטה המהירה ביותר המוכרת כיום רצה בזמן $\Theta(\beta \lg \beta \lg\lg \beta)$. אולם למטרות מעשיות, האלגוריתם שרץ בזמן $\Theta(\beta^2)$ הוא לעתים קרובות הטוב ביותר, ואנו נבסס את הניתוחים שלנו על חסם זה.

בפרק זה, ניתוחי האלגוריתמים ייערכו בדרך כלל במונחי מספר הפעולות האריתמטיות וגם במונחי מספר הפעולות על סיביות שדורשים האלגוריתמים.

33.1 מושגים בסיסיים בתורת המספרים

בסעיף זה נחזור בקצרה על מושגים בסיסיים בתורת המספרים הנוגעים לקבוצת השלמים $\mathbf{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ ולקבוצת המספרים הטבעיים $\mathbf{N} = \{0, 1, 2, \ldots\}$.

חלוקה ללא שארית ומחלקים

המושג של שלם אחד המתחלק ללא שארית בשלם אחר הוא מושג מרכזי בתורת המספרים. הסימון $d \mid a$ (קרי: "d **מחלק** את a", d divides a) פירושו שקיים שלם k שעבורו $a = kd$. כל שלם מחלק את 0. אם $a > 0$ ו־$d \mid a$, אזי $|d| \le |a|$. אם $d \mid a$, אנו אומרים גם ש־a הוא **כפולה** (multiple) של d. אם d אינו מחלק את a, אנו כותבים $d \nmid a$.

אם $d \mid a$ ו־$d \ge 0$, אנו אומרים ש־d הוא **מחלק** (divisor) של a. נשים לב ש־$d \mid a$ אם ורק אם $-d \mid a$, כך שניתן, בלי הגבלת הכלליות, להגדיר מחלקים כאי־שליליים, תוך ידיעה שהשלילי של כל מחלק של a מחלק גם הוא את a. המחלק של כל שלם a אינו קטן מ־1 ואינו גדול מ־$|a|$. לדוגמה, המחלקים של 24 הם 1, 2, 3, 4, 6, 8, 12 ו־24.

כל שלם a מתחלק **במחלקים הטריוויאליים** (trivial divisors) 1 ו־a. מחלקים לא טריוויאליים של a נקראים גם **גורמים** (factors) של a. לדוגמה, הגורמים של 20 הם 2, 4, 5 ו־10.

מספרים ראשוניים ומספרים פריקים

שלם $a > 1$ שהמחלקים היחידים שלו הם המחלקים הטריוויאליים 1 ו־a נקרא **מספר ראשוני**, או פשוט **ראשוני** (prime). מספרים ראשוניים הם בעלי תכונות מיוחדות רבות והם ממלאים תפקיד חיוני בתורת המספרים. המספר הראשוניים הקטנים, על פי סדרם, הם:

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, \ldots$$

בתרגיל 33.1-1 יתבקש הקורא להוכיח שקיימים אינסוף מספרים ראשוניים. שלם $a > 1$ שאינו ראשוני נקרא **מספר פריק**, או בקיצור **פריק** (composite). לדוגמה, 39 הוא פריק, שכן $3 \mid 39$.

השלם 1 נקרא **יחידה** (unit) והוא אינו ראשוני ואינו פריק. באופן דומה, השלם 0 וכל השלמים השליליים אינם ראשוניים ואינם פריקים.

משפט החילוק, שאריות ושקילות מודולרית

בהינתן שלם n, ניתן לחלק את השלמים לאלה שהם כפולות של n ולאלה שאינם כפולות של n. חלק בלתי מבוטל מתורת המספרים מבוסס על עידון של חלוקה זו, המתקבל על־ידי כך שמסווגים את השלמים שאינם כפולות של n על פי השארית המתקבלת מחלוקתם ב־n. המשפט שלהלן הוא הבסיס לעידון זה. את הוכחת המשפט לא נביא כאן (ראה, למשל, Niven & Zuckerman [151]).

משפט 33.1 (משפט החילוק)
עבור כל שלם a ושלם חיובי n, קיימים שלמים יחידים q ו־r המקיימים $0 \le r < n$ ו־$a = qn + r$. ■

הערך $q = \lfloor a/n \rfloor$ הוא **המנה** (quotient) של החילוק. הערך $r = a \bmod n$ הוא **השארית** (remainder, residue) של החילוק. מקבלים כי $n \mid a$ אם ורק אם $a \bmod n = 0$. מכך נובע כי:

$$a = \lfloor a/n \rfloor n + (a \bmod n) \tag{33.1}$$

או:

$$a \bmod n = a - \lfloor a/n \rfloor n \tag{33.2}$$

בהינתן מושג מוגדר היטב זה של השארית המתקבלת מחלוקת שלם אחד באחר, נוח לקבוע סימון מיוחד לציון שוויון שאריות. אם $(a \bmod n) = (b \bmod n)$, אנו כותבים $a \equiv b \pmod n$ ואומרים כי a **שקול** (equivalent) ל־b, מודולו n. במילים אחרות, $a \equiv b \pmod n$ אם a ו־b נותנים אותה שארית כאשר מחלקים אותם ב־n. לחלופין, $a \equiv b \pmod n$ אם ורק אם $n \mid (b - a)$. אנו כותבים $a \not\equiv b \pmod n$ אם a אינו שקול ל־b, מודולו n. לדוגמה, $61 \equiv 6 \pmod{11}$, וכן, $22 \equiv 2 \pmod 5$ ו־$-13 \equiv 2 \pmod 5$.

ניתן לחלק את השלמים ל־n מחלקות שקילות על פי השאריות המתקבלות מחלוקתם ב־n. **מחלקת השקילות מודולו n** (equivalence class modulo n) המכילה שלם a היא:

$$[a]_n = \{a + kn : k \in \mathbf{Z}\}$$

לדוגמה, $[3]_7 = \{\ldots, -11, -4, 3, 10, 17, \ldots\}$; סימונים אחרים של קבוצה זו הם, למשל, $[-4]_7$ או $[10]_7$. הסימון $a \in [b]_n$ מציין אותו דבר כמו $a \equiv b \pmod n$. קבוצת כל מחלקות השקילות מסוג זה היא:

$$\mathbf{Z}_n = \{[a]_n : 0 \le a \le n - 1\} \tag{33.3}$$

לעתים קרובות נתקלים בהגדרה:

$$\mathbf{Z}_n = \{0, 1, \ldots, n - 1\} \tag{33.4}$$

שיש להתייחס אליה כשקולה למשוואה (33.3), מתוך הבנה ש־0 מייצג את $[0]_n$, 1 מייצג את $[1]_n$, וכן הלאה; כל מחלקה מיוצגת על־ידי האיבר האי־שלילי הקטן ביותר שלה. עם זאת, תמיד

צריך לזכור שמדובר במחלקות שקילות. לדוגמה, כאשר מתייחסים אל -1 כאל איבר ב־$\mathbf{Z}_n$, פירוש הדבר שמתייחסים אל $[n-1]_n$, שכן $-1 \equiv n-1 \pmod n$.

מחלקים משותפים ומחלקים משותפים מקסימליים

אם d הוא מחלק של a וגם מחלק של b, אזי d הוא **מחלק משותף** (common divisor) של a ו־b. לדוגמה, המחלקים של 30 הם 1, 2, 3, 5, 6, 10, 15 ו־30, ולכן המחלקים המשותפים של 24 ו־30 הם 1, 2, 3 ו־6. נשים לב ש־1 הוא מחלק משותף של כל שני שלמים.

תכונה חשובה של מחלקים משותפים היא:

$$\text{אם } d \mid a \text{ וגם } d \mid b \text{ אזי } d \mid (a+b) \text{ וגם } d \mid (a-b) \tag{33.5}$$

ובאופן כללי יותר, מתקיים:

$$\text{אם } d \mid a \text{ וגם } d \mid b \text{ אזי } d \mid (ax+by) \tag{33.6}$$

עבור כל x ו־y שלמים. כמו כן, אם $a \mid b$, אזי או $|a| \le |b|$ או $b = 0$, ומכך נובע:

$$\text{אם } a \mid b \text{ וגם } b \mid a \text{ אזי } a = \pm b \tag{33.7}$$

המחלק המשותף המקסימלי (greatest common divisor) של שני שלמים a ו־b, אשר לפחות אחד מהם שונה מאפס, הוא הגדול ביותר מבין המחלקים המשותפים של a ו־b, והוא מסומן על־ידי $\gcd(a, b)$. לדוגמה, $\gcd(24, 30) = 6$, $\gcd(5, 7) = 1$ ו־$\gcd(0, 9) = 9$. אם a ו־b שניהם שונים מאפס, אזי $\gcd(a, b)$ הוא מספר שלם בין 1 ל־$\min(|a|, |b|)$. אנו מגדירים $\gcd(0, 0) = 0$; הגדרה זו נחוצה כדי שהתכונות שמקיימת הפונקציה gcd (למשל, משוואה (3.11) להלן) יהיו תקפות בכל המקרים.

להלן כמה תכונות בסיסיות של הפונקציה gcd:

$$\gcd(a, b) = \gcd(b, a) \tag{33.8}$$

$$\gcd(a, b) = \gcd(-a, b) \tag{33.9}$$

$$\gcd(a, b) = \gcd(|a|, |b|) \tag{33.10}$$

$$\gcd(a, 0) = |a| \tag{33.11}$$

$$\gcd(a, ka) = |a| \qquad \text{לכל } k \in \mathbf{Z} \tag{33.12}$$

משפט 33.2

אם a ו־b הם שלמים כלשהם, אשר לפחות אחד מהם שונה מאפס, אזי $\gcd(a, b)$ הוא האיבר החיובי הקטן ביותר בקבוצה $\{ax + by : x, y \in \mathbf{Z}\}$ של הצירופים הלינאריים של a ו־b.

הוכחה יהי s צירוף לינארי קטן ביותר כזה של a ו־b, כלומר, קיימים $x, y \in \mathbf{Z}$ כך ש־$s = ax + by$. יהי $q = \lfloor a/s \rfloor$. אזי ממשוואה (33.2) נובע:

$$\begin{aligned} a \bmod s &= a - qs \\ &= a - q(ax + by) \\ &= a(1 - qx) + b(-qy) \end{aligned}$$

ולכן $a \bmod s$ גם הוא צירוף לינארי של a ו־b. אולם, מכיוון ש־$a \bmod s < s$, אנו מקבלים כי $a \bmod s = 0$, משום ש־s הוא הצירוף הלינארי החיובי הקטן ביותר משוג זה. לכן, $a \mid s$,

ובאמצעות טיעון אנלוגי, $s \mid b$. לפיכך, s הוא מחלק משותף של a ו־b, ולכן $\gcd(a, b) \ge s$. משוואה (33.6) גוררת כי $s \mid \gcd(a, b)$, שכן $\gcd(a, b)$ מחלק גם את a וגם את b ו־s הוא צירוף לינארי של a ו־b. אבל $\gcd(a, b) \mid s$ ו־$s > 0$ גוררים כי $s \le \gcd(a, b)$. הצירוף של $s \ge \gcd(a, b)$ ו־$s \le \gcd(a, b)$ נותן $\gcd(a, b) = s$; אנו מסיקים אפוא כי s הוא המחלק המשותף המקסימלי של a ו־b. ■

מסקנה 33.3

עבור כל שני שלמים a ו־b, אם $d \mid a$ וגם $d \mid b$, אזי $d \mid \gcd(a, b)$.

הוכחה מסקנה זו נובעת ממשוואה (33.6), שכן על פי משפט 33.2, $\gcd(a, b)$ הוא צירוף לינארי של a ו־b. ■

מסקנה 33.4

עבור כל שני שלמים a ו־b ושלם אי־שלילי n כלשהו, מתקיים:

$$\gcd(an, bn) = n \gcd(a, b)$$

הוכחה אם $n = 0$, המסקנה טריוויאלית. אם $n > 0$, אזי $\gcd(an, bn)$ הוא האיבר החיובי הקטן ביותר בקבוצה $\{anx + bny\}$, ואיבר זה גדול פי n מן האיבר החיובי הקטן ביותר בקבוצה $\{ax + by\}$. ■

מסקנה 33.5

עבור כל n, a ו־b שלמים וחיוביים, אם $n \mid ab$ וגם $\gcd(a, n) = 1$, אזי $n \mid b$.

הוכחה הקורא יתבקש להוכיח את המסקנה בתרגיל 33.1-4. ■

שלמים זרים זה לזה

שני שלמים a ו־b נקראים **זרים זה לזה** (relatively prime) אם המחלק המשותף היחיד שלהם הוא 1, כלומר, אם $\gcd(a, b) = 1$. לדוגמה, 8 ו־15 הם זרים, שכן המחלקים של 8 הם 1, 2, 4 ו־8 ואילו המחלקים של 15 הם 1, 3, 5 ו־15. המשפט הבא קובע שאם כל אחד משני שלמים הוא זר לשלם p, אזי המכפלה שלהם זרה ל־p.

משפט 33.6

עבור כל a, b ו־p שלמים, אם $\gcd(a, p) = 1$ וגם $\gcd(b, p) = 1$, אזי $\gcd(ab, p) = 1$.

הוכחה ממשפט 33.2 נובע שקיימים שלמים x, y, x' ו־y', המקיימים:

$$ax + py = 1$$

$$bx' + py' = 1$$

מכפל משוואות אלה זו בזו וסידור האיברים מחדש אנו מקבלים:

$$ab(xx') + p(ybx' + y'ax + pyy') = 1$$

1 הוא אפוא צירוף לינארי חיובי של ab ו־p, ולכן ממשפט 33.2 נובע כי $\gcd(ab, p) = 1$. ■

אנו אומרים שהשלמים $n_1, n_2, \ldots, n_k$ **זרים בזוגות** (pairwise relatively prime), אם $i \ne j$ גורר $\gcd(n_i, n_j) = 1$.

פירוק יחיד לגורמים

המשפט הבא מביא עובדה בסיסית אך חשובה לגבי מחלקים ראשוניים.

משפט 33.7
עבור כל p ראשוני וכל a ו־b שלמים, אם $p \mid ab$ אזי $p \mid a$ או $p \mid b$.

הוכחה נניח, בדרך השלילה, כי $p \mid ab$ אבל $p \nmid a$ וגם $p \nmid b$. לכן, $\gcd(a, p) = 1$ וגם $\gcd(b, p) = 1$, שכן המחלקים היחידים של p הם 1 ו־p, ועל פי הנחתנו p אינו מחלק את a או את b. אזי, משפט 33.6 גורר כי $\gcd(ab, p) = 1$, בסתירה להנחתנו כי $p \mid ab$, שכן $\gcd(ab, p) = p$ גורר $p \mid ab$. סתירה זו משלימה את ההוכחה. ■

תוצאה של משפט 33.7 היא שלכל שלם יש פירוק יחיד לגורמים ראשוניים.

משפט 33.8 (פירוק יחיד לגורמים)
לשלם פריק a קיים ייצוג יחיד עד־כדי מכפלה מן הצורה:

$$a = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$$

כאשר ה־p_i ראשוניים, $p_1 < p_2 < \cdots < p_r$, וה־e_i הם שלמים חיוביים.

הוכחה הקורא יתבקש להוכיח את המשפט בתרגיל 33.1-10. ■

לדוגמה, הפירוק היחיד לגורמים של המספר 6000 הוא $2^4 \cdot 3 \cdot 5^3$.

תרגילים

33.1-1
הוכח שקיימים אינסוף מספרים ראשוניים. (**רמז:** הראה שאף אחד מן הראשוניים $p_1, p_2, \ldots, p_k$ אינו מחלק את $(p_1 p_2 \cdots p_k) + 1$.)

33.1-2
הוכח שאם $a \mid b$ וגם $b \mid c$, אזי $a \mid c$.

33.1-3
הוכח שאם p ראשוני ו־$0 < k < p$, אזי $\gcd(k, p) = 1$.

33.1-4
הוכח את מסקנה 33.5.

33.1-5
הוכח שאם p ראשוני ו־$0 < k < p$, אזי $p \mid \binom{p}{k}$. הסק שעבור כל a ו־b שלמים וכל p ראשוני, מתקיים:

$$(a + b)^p \equiv a^p + b^p \pmod{p}$$

33.1-6
הוכח שאם a ו־b הם שלמים כלשהם המקיימים $a \mid b$ ו־$b > 0$, אזי עבור כל x מתקיים:

$$(x \bmod b) \bmod a = x \bmod a$$

הוכח כי באותן הנחות, עבור כל שני שלמים x ו־y מתקיים:

$$x \equiv y \pmod{b} \implies x \equiv y \pmod{a}$$

33.1-7
עבור כל שלם $k > 0$, אנו אומרים ששלם n הוא **חזקה k־ית** (kth power) אם קיים שלם a כך ש־$a^k = n$. נאמר כי $n > 1$ הוא **חזקה לא טריוויאלית** (nontrivial power) אם n הוא חזקה k־ית עבור איזשהו $k > 1$. בהינתן שלם n בן β סיביות, הראה כיצד ניתן לקבוע בזמן פולינומאלי ב־β אם n הוא חזקה לא טריוויאלית.

33.1-8
הוכח את משוואות (33.8)-(33.12).

33.1-9
הראה שהאופרטור gcd הוא אסוציאטיבי. כלומר, הוכח שעבור כל a, b ו־c שלמים, מתקיים:

$$\gcd(a, \gcd(b, c)) = \gcd(\gcd(a, b), c)$$

* ***33.1-10***
הוכח את משפט 33.8.

33.1-11
כתוב אלגוריתמים יעילים עבור הפעולות הבאות: חלוקת שלם בן β סיביות בשלם קצר יותר, וחישוב השארית המתקבלת מחלוקת שלם בן β סיביות בשלם קצר יותר. האלגוריתמים שלך צריכים לרוץ בזמן $O(\beta^2)$.

33.1-12
כתוב אלגוריתם יעיל להמרת שלם הנתון בייצוג בינרי בן β סיביות לייצוג עשרוני. הראה שאם כפל או חילוק של שלמים שאורכם לכל היותר β מתבצעים בזמן $M(\beta)$, אזי ההמרה מייצוג בינרי לעשרוני יכולה להתבצע בזמן $\Theta(M(\beta)\lg\beta)$. (**רמז:** השתמש בגישת הפרד־ומשול, וחשב את המחצית העליונה והמחצית התחתונה של התוצאה באמצעות רקורסיות נפרדות.)

33.2 מחלק משותף מקסימלי

בסעיף זה נשתמש באלגוריתם של אוקלידס כדי לחשב ביעילות את המחלק המשותף המקסימלי של שני שלמים. מניתוח זמן הריצה עולה קשר מפתיע בין אלגוריתם זה לבין מספרי פיבונצ'י, אשר מהווים את הקלטים הגרועים ביותר לאלגוריתם של אוקלידס.

בסעיף זה נגביל עצמנו לשלמים אי־שליליים. ההצדקה להגבלה זו היא משוואה (33.10), הקובעת כי $\gcd(a, b) = \gcd(|a|, |b|)$.

בעיקרון, ניתן לחשב את $\gcd(a, b)$ עבור שלמים חיוביים a ו־b מתוך הפירוקים של a ו־b לגורמים ראשוניים. ואמנם, אם:

$$a = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r} \tag{33.13}$$

$$b = p_1^{f_1} p_2^{f_2} \cdots p_r^{f_r} \tag{33.14}$$

כאשר משתמשים בחזקות אפס כדי לקבל אותה סדרה של ראשוניים $p_1, p_2, \ldots, p_r$ עבור a ועבור b, אזי:

$$\gcd(a, b) = p_1^{\min(e_1, f_1)} p_2^{\min(e_2, f_2)} \cdots p_r^{\min(e_r, f_r)} \tag{33.15}$$

כפי שנראה בסעיף 33.9, האלגוריתמים הטובים ביותר המוכרים כיום לפירוק לגורמים אינם רצים בזמן פולינומיאלי. לפיכך, לא נראה שיש סיכוי כי גישה זו לחישוב מחלקים משותפים מקסימליים תניב אלגוריתם יעיל.

האלגוריתם של אויקלידס לחישוב מחלקים משותפים מקסימליים מבוסס על המשפט הבא.

משפט 33.9 (רקורסיביות ה־GCD)
עבור כל שלם אי־שלילי a וכל שלם חיובי b:

$$\gcd(a, b) = \gcd(b, a \bmod b)$$

הוכחה אנו נראה כי $\gcd(a, b)$ ו־$\gcd(b, a \bmod b)$ מחלקים זה את זה, ולכן, על פי משוואה (33.7), הם בהכרח שווים זה לזה (שכן שניהם אי־שליליים).

תחילה נראה כי $\gcd(a, b) \mid \gcd(b, a \bmod b)$. אם נסמן $d = \gcd(a, b)$, אזי $d \mid a$ ו־$d \mid b$. על פי משוואה (33.2), $(a \bmod b) = a - qb$, כאשר $q = \lfloor a/b \rfloor$ הוא אפוא צירוף לינארי של a ו־b, ולכן משוואה (33.6) גוררת כי $d \mid (a \bmod b)$. לכן, מאחר ש־$d \mid b$ וגם $d \mid (a \bmod b)$, מסקנה 33.3 גוררת כי $d \mid \gcd(b, a \bmod b)$, או, באופן שקול:

$$\gcd(a, b) \mid \gcd(b, a \bmod b) \tag{33.16}$$

בצורה דומה מאוד מראים גם כי $\gcd(b, a \bmod b) \mid \gcd(a, b)$. אם נסמן עתה $d = \gcd(b, a \bmod b)$, אזי $d \mid b$ וגם $d \mid (a \bmod b)$. מכיוון ש־$a = qb + (a \bmod b)$, כאשר $q = \lfloor a/b \rfloor$, הרי ש־a הוא צירוף לינארי של b ו־$(a \bmod b)$. על פי משוואה (33.6), נסיק כי $d \mid a$. מאחר ש־$d \mid b$ וגם $d \mid a$, נקבל על פי מסקנה 33.3 כי $d \mid \gcd(a, b)$, או, באופן שקול:

$$\gcd(b, a \bmod b) \mid \gcd(a, b) \tag{33.17}$$

שימוש במשוואה (33.7) כדי לצרף את משוואות (33.16) ו־(33.17) משלים את ההוכחה. ■

האלגוריתם של אויקלידס

האלגוריתם שלהלן למציאת מחלק משותף מקסימלי מתואר בספרו של אויקלידס *יסודות* (*Elements*, 300 לפנה"ס בערך), אם כי ייתכן שמקורו קדום אף יותר. האלגוריתם כתוב כתכנית רקורסיבית המבוססת ישירות על משפט 33.9. הקלטים a ו־b הם שלמים אי־שליליים כלשהם.

```
EUCLID(a, b)
1  if b = 0
2      then return a
3  else return EUCLID(b, a mod b)
```


כדוגמה להרצה של EUCLID, נתבונן בחישוב $\gcd(30, 21)$:

$$\begin{aligned} \text{EUCLID}(30, 21) &= \text{EUCLID}(21, 9) \\ &= \text{EUCLID}(9, 3) \\ &= \text{EUCLID}(3, 0) \\ &= 3 \end{aligned}$$

במהלך חישוב זה מתבצעות שלוש קריאות רקורסיביות ל־EUCLID.

הנכונות של EUCLID נובעת ממשפט 33.9 ומכך שאם האלגוריתם מחזיר a בשורה 2, אזי $b = 0$, ולכן משוואה (33.11) גוררת כי $\gcd(a, b) = \gcd(a, 0) = a$. האלגוריתם אינו יכול להמשיך ללא סוף בקריאות הרקורסיביות, שכן הארגומנט השני קטן ממש בכל קריאה רקורסיבית. לכן, האלגוריתם EUCLID עוצר תמיד עם התשובה הנכונה.

זמן הריצה של האלגוריתם של אויקלידס

נעתח את זמן הריצה של EUCLID במקרה הגרוע כפונקציה של גודלם של a ו־b. נניח, כמעט ללא הגבלת הכלליות, כי $a > b \geq 0$. נוכל להצדיק הנחה זו על־ידי ההבחנה הבאה: אם $b > a \geq 0$, אזי השגרה $\text{EUCLID}(a, b)$ מבצעת מייד את הקריאה הרקורסיבית $\text{EUCLID}(b, a)$. כלומר, אם הארגומנט הראשון קטן מן השני, השגרה מקדישה קריאה רקורסיבית אחת להחלפת הארגומנטים, ואז ממשיכה. באופן דומה, אם $b = a > 0$, השגרה עוצרת לאחר קריאה רקורסיבית אחת, שכן במקרה זה, $a \bmod b = 0$.

זמן הריצה הכולל של השגרה EUCLID נמצא ביחס ישר למספר הקריאות הרקורסיביות שהיא מבצעת. הניתוח שלנו נעזר במספרי פיבונאצ'י F_k, המוגדרים על־ידי נוסחת הנסיגה (2.13).

למה 33.10
אם $a > b \geq 0$ ובמהלך הרצת $\text{EUCLID}(a, b)$ מתבצעות $k \geq 1$ קריאות רקורסיביות, אזי $a \geq F_{k+2}$ ו־$b \geq F_{k+1}$.

הוכחה ההוכחה היא באינדוקציה על k. עבור בסיס האינדוקציה, יהי $k = 1$. אזי, $b \geq 1 = F_2$, ומכיוון ש־$a > b$, אזי בהכרח $a \geq 2 = F_3$. מכיוון ש־$b > (a \bmod b)$, אזי בכל קריאה רקורסיבית הארגומנט הראשון קטן ממש מן השני; ההנחה כי $a > b$ תקפה אפוא עבור כל אחת מן הקריאות הרקורסיביות.

נניח, באינדוקציה, שהלמה נכונה אם מתבצעות $k - 1$ קריאות רקורסיביות, ונוכיח שהיא נכונה עבור k קריאות רקורסיביות. מכיוון ש־$k > 0$, אזי $b > 0$, ו־$\text{EUCLID}(a, b)$ קוראת באופן רקורסיבי ל־$\text{EUCLID}(b, a \bmod b)$, וזו בתורה מבצעת $k - 1$ קריאות רקורסיביות. מהנחת האינדוקציה נובע אפוא כי $b \geq F_{k+1}$ (ובכך הוכחנו חלק מהלמה) וכי $(a \bmod b) \geq F_k$. אנו מקבלים:

$$\begin{aligned} b + (a \bmod b) &= b + (a - \lfloor a/b \rfloor b) \\ &\leq a \end{aligned}$$

שכן $a > b > 0$ גורר $\lfloor a/b \rfloor \geq 1$. ומכאן:

$$\begin{aligned} a &\geq b + (a \bmod b) \\ &\geq F_{k+1} + F_k \\ &= F_{k+2} \end{aligned}$$

■

המשפט הבא הוא מסקנה ישירה מן הלמה שלעיל.

משפט 33.11 (משפט לאמה (Lamé))
עבור כל שלם $k \geq 1$, אם $a > b \geq 0$ ו־$b < F_{k+1}$, אזי קריאה ל־$\text{EUCLID}(a, b)$ מבצעת פחות מ־k קריאות רקורסיביות. ■

ניתן להראות שהחסם העליון ממשפט 33.11 הוא הטוב ביותר האפשרי. זוג מספרי פיבונצ'י עוקבים הם הקלט הגרוע ביותר לשגרה EUCLID. מכיוון ש־$\text{EUCLID}(F_3, F_2)$ מבצעת בדיוק קריאה רקורסיבית אחת, ומכיוון שעבור $k \geq 2$ מתקיים $(F_{k+1} \bmod F_k) = F_{k-1}$, אזי:

$$\begin{aligned} \gcd(F_{k+1}, F_k) &= \gcd(F_k, (F_{k+1} \bmod F_k)) \\ &= \gcd(F_k, F_{k-1}) \end{aligned}$$

לכן, $\text{EUCLID}(F_{k+1}, F_k)$ מבצעת *בדיוק* $k - 1$ קריאות רקורסיביות, וזהו החסם העליון המופיע במשפט 33.11.

מכיוון ש־F_k שווה בקירוב ל־$\phi^k/\sqrt{5}$, כאשר ϕ הוא יחס הזהב $(1 + \sqrt{5})/2$ המוגדר על־ידי משוואה (2.14), אזי מספר הקריאות הרקורסיביות שמבצעת EUCLID הוא $O(\lg b)$. נראה חסם הדוק יותר בתרגיל 33.2-5. מכך נובע שאם נריץ את EUCLID על שני מספרים בני β סיביות, היא תבצע $O(\beta)$ פעולות אריתמטיות ו־$O(\beta^3)$ פעולות על סיביות (בהנחה שכפל וחילוק של מספרים בני β סיביות דורשים $O(\beta^2)$ פעולות על סיביות). בבעיה 33-2 יתבקש הקורא להראות שמספר הפעולות על סיביות חסום על־ידי $O(\beta^2)$.

הגרסה המורחבת של האלגוריתם של אויקלידס

נכתוב עתה מחדש את האלגוריתם של אויקלידס כך שיחשב מידע מועיל נוסף. בפרט, נרחיב את האלגוריתם כך שיחשב את המקדמים השלמים x ו־y המקיימים:

$$d = \gcd(a, b) = ax + by \tag{33.18}$$

נשים לב כי ערכם של x ו־y עשוי להיות אפס או שלילי. בהמשך נראה כי מקדמים אלה שימושיים בחישוב הופכיים כפליים מודולו n. השגרה EXTENDED-EUCLID מקבלת כקלט שני שלמים אי־שליליים כלשהם ומחזירה שלשה מהצורה (d, x, y) המקיימת את משוואה (33.18).

```
EXTENDED-EUCLID(a, b)
1  if b = 0
2     then return (a, 1, 0)
3  (d', x', y') ← EXTENDED-EUCLID(b, a mod b)
4  (d, x, y) ← (d', y', x' − ⌊a/b⌋y')
5  return (d, x, y)
```


איור 33.1 מדגים הרצה של EXTENDED-EUCLID לחישוב $\gcd(99, 78)$.

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1	3	−11	14
78	21	3	3	3	−11
21	15	1	3	−2	3
15	6	2	3	1	−2
6	3	2	3	0	1
3	0	—	3	1	0

איור 33.1 דוגמה לפעולתה של EXTENDED-EUCLID על הקלטים 99 ו־78. כל שורה מייצגת רמה אחת של הרקורסיה: ערכי הקלטים a ו־b, הערך המחושב $\lfloor a/b \rfloor$ וערכי d, x ו־y שמחזירה רמה זו. השלשה (d, x, y) שמחזירה רמה אחת הופכת לשלשה (d', x', y') המשמשת בחישובים המבוצעים ברמה שמעלית. הקריאה EXTENDED-EUCLID(99, 78) מחזירה $(3, -11, 14)$ ופירוש הדבר ש־$\gcd(99, 78) = 3 = 99(-11) + 78 \cdot 14$. וכן $\gcd(99, 78) = 3 = 99 \cdot (-11) + 78 \cdot 14$.

השגרה EXTENDED-EUCLID היא וריאציה של השגרה EUCLID. שורה 1 מקבילה לבדיקה "$b = 0$" בשורה 1 של EUCLID. אם $b = 0$, אזי EXTENDED-EUCLID מחזירה לא רק $d = a$ בשורה 2, אלא גם את המקדמים $x = 1$ ו־$y = 0$ שעבורם $a = ax + by$. אם $b \neq 0$, אזי EXTENDED-EUCLID מחשבת תחילה שלשה (d', x', y') המקיימת $d' = \gcd(b, a \bmod b)$ וכן:

$$d' = bx' + (a \bmod b)y' \tag{33.19}$$

בשגרה EUCLID אנו מקבלים במקרה זה $d = \gcd(a, b) = d' = \gcd(b, a \bmod b)$. כדי לקבל x ו־y כך ש־$d = ax + by$, נתחיל בכך שנכתוב מחדש את משוואה (33.19) בעזרת המשוואה $d = d'$ ומשוואה (33.2):

$$\begin{aligned} d &= bx' + (a - \lfloor a/b \rfloor b)y' \\ &= ay' + b(x' - \lfloor a/b \rfloor y') \end{aligned}$$

לכן, הבחירה $x = y'$ ו־$y = x' - \lfloor a/b \rfloor y'$ מספקת את המשוואה $d = ax + by$, ובכך הוכחה נכונותה של EXTENDED-EUCLID.

מאחר שמספר הקריאות הרקורסיביות שמבצעת EUCLID שווה למספר הקריאות הרקורסיביות שמבצעת EXTENDED-EUCLID, זמני הריצה של שתי השגרות שווים, עד כדי גורם קבוע. כלומר, עבור $a > b > 0$, מספר הקריאות הרקורסיביות הוא $O(\lg b)$.

תרגילים

33.2-1
הוכח כי משוואות (33.13)-(33.14) גוררות את משוואה (33.15).

33.2-2
חשב את הערכים (d, x, y) המתקבלים כפלט כאשר מבצעים את הקריאה EXTENDED-EUCLID(899, 493).

33.2-3
הוכח שעבור כל a, k ו־n שלמים מתקיים:

$$\gcd(a, n) = \gcd(a + kn, n)$$

33.2-4
כתוב מחדש את EUCLID כשגרה איטרטיבית המשתמשת בכמות קבועה בלבד של מקום בזיכרון (כלומר, מאחסנת מספר קבוע בלבד של ערכים שלמים).

33.2-5
הראה שאם $a > b \geq 0$, הקריאה EUCLID(a, b) מבצעת לכל היותר $1 + \log_\phi b$ קריאות רקורסיביות. שפר חסם זה ל־$1 + \log_\phi (b / \gcd(a, b))$.

33.2-6
מה מחזירה הקריאה EXTENDED-EUCLID(F_{k+1}, F_k)? הוכח את נכונות תשובתך.

33.2-7
אמת את נכונותו של הפלט (d, x, y) שמחזירה EXTENDED-EUCLID(a, b) על־ידי כך שתראה כי אם $d \mid a$, $d \mid b$ ו־$d = ax + by$, אזי $d = \gcd(a, b)$.

33.2-8
הגדר את הפונקציה gcd עבור יותר משני ארגומנטים באמצעות המשוואה הרקורסיבית $\gcd(a_0, a_1, \ldots, a_n) = \gcd(a_0, \gcd(a_1, \ldots, a_n))$. הראה כי gcd מחזירה אותה תשובה בלי תלות בסדר שבו כותבים את הארגומנטים. הראה כיצד מוצאים $x_0, x_1, \ldots, x_n$ שעבורם $\gcd(a_0, a_1, \ldots, a_n) = a_0 x_0 + a_1 x_1 + \cdots + a_n x_n$. הראה שמספר פעולות החילוק שמבצע האלגוריתם שלך הוא $O(n + \lg(\max_i a_i))$.

33.2-9
נסמן ב־$\operatorname{lcm}(a_1, a_2, \ldots, a_n)$ את **הכפולה המשותפת המינימלית** (least common multiple) של השלמים $a_1, a_2, \ldots, a_n$, כלומר, השלם האי־שלילי הקטן ביותר שהוא כפולה של כל אחד מן ה־a_i. הראה כיצד ניתן לחשב ביעילות את $\operatorname{lcm}(a_1, a_2, \ldots, a_n)$ תוך שימוש בפונקציה gcd (על שני ארגומנטים) כתת־שגרה.

33.2-10
הוכח כי n_1, n_2, n_3 ו־n_4 זרים בזוגות אם ורק אם:

$$\gcd(n_1 n_2, n_3 n_4) = \gcd(n_1 n_3, n_2 n_4) = 1$$

הוכח באופן כללי יותר כי $n_1, n_2, \ldots, n_k$ זרים בזוגות אם ורק אם קבוצה של $\lceil \lg k \rceil$ זוגות מספרים הנשלפים מקבוצת ה־n_i הם זרים זה לזה.

33.3 אריתמטיקה מודולרית

באופן בלתי פורמלי, אנו יכולים לחשוב על אריתמטיקה מודולרית כעל אריתמטיקה רגילה בשלמים, אלא שכאשר עובדים מודולו n, מחליפים כל תוצאה x באיבר מתוך $\{0, 1, \ldots, n-1\}$ השקול ל־x, מודולו n (דהיינו, מחליפים את x ב־$(x \bmod n)$). מודל בלתי פורמלי זה די בו אם מבצעים אך ורק פעולות חיבור, חיסור וכפל. להלן נציג מודל פורמלי יותר של אריתמטיקה מודולרית, שאותו ניתן לתאר בדרך הטובה ביותר במסגרת תורת החבורות.

חבורות סופיות

חבורה (group) $(S, \oplus)$ היא קבוצה S יחד עם פעולה בינרית $\oplus$ המוגדרת על S, כך שמתקיימות התכונות הבאות:

1. **סגירות** (closure): לכל $a, b \in S$, מתקיים $a \oplus b \in S$.
2. **איבר יחידה** (identity): קיים איבר $e \in S$ כך ש־$a \oplus e = e \oplus a = a$ לכל $a \in S$.
3. **אסוציאטיביות** (associativity): לכל $a, b, c \in S$, מתקיים
$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$
4. **איברים הופכיים** (inverses): לכל $a \in S$, קיים איבר יחיד $b \in S$ שעבורו מתקיים $a \oplus b = b \oplus a = e$.

לדוגמה, נתבונן בחבורה המוכרת $(\mathbf{Z}, +)$, חבורת השלמים $\mathbf{Z}$ תחת פעולת החיבור: 0 הוא איבר היחידה, והאיבר ההופכי של a הוא $-a$. אם חבורה $(S, \oplus)$ מקיימת את **חוק הקומוטטיביות**: $a \oplus b = b \oplus a$ לכל $a, b \in S$, אזי היא **חבורה אבלית** (abelian group). אם חבורה $(S, \oplus)$ מקיימת $|S| < \infty$, אזי היא **חבורה סופית** (finite group).

החבורות המוגדרות על־ידי חיבור וכפל מודולריים

ניתן ליצור שתי חבורות אבליות סופיות על־ידי שימוש בחיבור וכפל מודולו n, כאשר n הוא שלם חיובי. חבורות אלה מבוססות על מחלקות השקילות של השלמים מודולו n, שהוגדרו בסעיף 33.1.

כדי להגדיר חבורה על $\mathbf{Z}_n$, אנו זקוקים לפעולות בינריות מתאימות, ואת אלה מקבלים על־ידי הגדרה מחדש של פעולות החיבור והכפל הרגילות. קל להגדיר פעולות חיבור וכפל על $\mathbf{Z}_n$, מכיוון שמחלקות השקילות של שני שלמים קובעות באופן יחיד את מחלקת השקילות של סכומם או מכפלתם. כלומר, אם $a \equiv a' \pmod n$ ו־$b \equiv b' \pmod n$, אזי:

$$a + b \equiv a' + b' \pmod n$$
$$ab \equiv a'b' \pmod n$$

אנו מגדירים אפוא חיבור וכפל מודולו n, המסומנים על־ידי $+_n$ ו־$\cdot_n$, כדלקמן:

$$[a]_n +_n [b]_n = [a + b]_n$$
$$[a]_n \cdot_n [b]_n = [ab]_n$$

(באותו אופן ניתן להגדיר חיסור על $\mathbf{Z}_n$ על־ידי $[a]_n -_n [b]_n = [a - b]_n$, אבל פעולת החילוק מסובכת יותר, כפי שנראה בהמשך.) עובדות אלה מצדיקות את הנוהג המקובל והנוח של שימוש באיבר האי־שלילי הקטן ביותר של כל מחלקת שקילות כנציג של מחלקה זו בעת ביצוע חישובים ב־$\mathbf{Z}_n$. חיבור, חיסור וכפל מבוצעים כרגיל על הנציגים, אבל כל תוצאה x מוחלפת בנציג של המחלקה שלה (כלומר, ב־$(x \bmod n)$).

תוך שימוש בהגדרה זו של חיבור מודולו n, אנו מגדירים את **החבורה החיבורית מודולו n** (additive group modulo n) כ־$(\mathbf{Z}_n, +_n)$. גודלה של החבורה החיבורית מודולו n הוא $n = |\mathbf{Z}_n|$. באיור 33.2(i) מוצג לוח החיבור עבור החבורה $(\mathbf{Z}_6, +_6)$.

משפט 33.12
המערכת $(\mathbf{Z}_n, +_n)$ היא חבורה אבלית סופית.

הוכחה האסוציאטיביות והקומוטטיביות של $+_n$ נובעות מן האסוציאטיביות והקומוטטיביות של $+$:

$$\begin{aligned}([a]_n +_n [b]_n) +_n [c]_n &= [(a+b)+c]_n \\ &= [a+(b+c)]_n \\ &= [a]_n +_n ([b]_n +_n [c]_n)\end{aligned}$$

$$\begin{aligned}[a]_n +_n [b]_n &= [a+b]_n \\ &= [b+a]_n \\ &= [b]_n +_n [a]_n\end{aligned}$$

איבר היחידה של $(\mathbf{Z}_n, +_n)$ הוא 0 (כלומר, $[0]_n$). ההופכי (החיבורי) של איבר a (כלומר, $[a]_n$) הוא האיבר $-a$ (כלומר, $[-a]_n$ או $[n-a]_n$), שכן $[a]_n +_n [-a]_n = [a-a]_n = [0]_n$. ■

תוך שימוש בהגדרה של כפל מודולו n, אנו מגדירים את **החבורה הכפלית מודולו n** (multiplicative group modulo n) כ־$(\mathbf{Z}_n^*, \cdot_n)$. איברי החבורה הזאת הם הקבוצה $\mathbf{Z}_n^*$ של האיברים ב־$\mathbf{Z}_n$ שהם זרים ל־n:

$$\mathbf{Z}_n^* = \{[a]_n \in \mathbf{Z}_n : \gcd(a, n) = 1\}$$

כדי לראות ש־$\mathbf{Z}_n^*$ מוגדרת היטב, נשים לב כי עבור $0 \le a < n$ מתקיים $a \equiv (a + kn) \pmod n$ עבור כל השלמים k. לכן, על פי תרגיל 33.2-3, $\gcd(a, n) = 1$ גורר $\gcd(a + kn, n) = 1$ עבור כל השלמים k. מאחר ש־$[a]_n = \{a + kn : k \in \mathbf{Z}\}$, הרי שהקבוצה $\mathbf{Z}_n^*$ מוגדרת היטב. דוגמה לחבורה כזאת היא:

$$\mathbf{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$$

כאשר הפעולה על החבורה היא כפל מודולו 15. (כאן אנו מסמנים ב־a את איבר $[a]_{15}$.) באיור 33.2(ii) מוצגת החבורה $(\mathbf{Z}_{15}^*, \cdot_{15})$. לדוגמה, כאשר עובדים ב־$\mathbf{Z}_{15}^*$, $8 \cdot 11 \equiv 13 \pmod{15}$. איבר היחידה עבור חבורה זו הוא 1.

$+_6$	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

(i)

$\cdot_{15}$	1	2	4	7	8	11	13	14
1	1	2	4	7	8	11	13	14
2	2	4	8	14	1	7	11	13
4	4	8	1	13	2	14	7	11
7	7	14	13	4	11	2	1	8
8	8	1	2	11	4	13	14	7
11	11	7	14	2	13	1	8	4
13	13	11	7	1	14	8	4	2
14	14	13	11	8	7	4	2	1

(ii)

איור 33.2 שתי חבורות סופיות. מחלקות שקילות מיוצגות על־ידי האיברים הנציגים שלהן. **(i)** החבורה $(\mathbf{Z}_6, +_6)$. **(ii)** החבורה $(\mathbf{Z}_{15}^*, \cdot_{15})$.

משפט 33.13
המערכת $(\mathbf{Z}_n^*, \cdot_n)$ היא חבורה אבלית סופית.

הוכחה הסגירות של $(\mathbf{Z}_n^*, \cdot_n)$ נובעת ממשפט 33.6. את האסוציאטיביות והקומוטטיביות של $\cdot_n$ ניתן להוכיח באותו אופן שבו הוכחנו אותן עבור $+_n$ במשפט 33.12. איבר היחידה הוא $[1]_n$. כדי להוכיח את קיומם של איברים הופכיים, יהי a איבר ב־$\mathbf{Z}_n^*$, ויהי (d, x, y) הפלט של EXTENDED-EUCLID(a, n). אזי $d = 1$, שכן $a \in \mathbf{Z}_n^*$, וכן:

$$ax + ny = 1$$

או, באופן שקול:

$$ax \equiv 1 \pmod n$$

לכן, $[x]_n$ הוא ההופכי הכפלי של $[a]_n$, מודולו n. ההוכחה כי ההופכיים מוגדרים באופן יחיד נדחית עד למסקנה 33.26. ■

מכאן ואילך, כאשר נעבוד בשארית הפרק עם החבורות $(\mathbf{Z}_n, +_n)$ ו־$(\mathbf{Z}_n^*, \cdot_n)$, נקוט בנוהג הנוח של ציון מחלקות שקילות על־ידי האיברים הנציגים שלהן, וציון הפעולות $+_n$ ו־$\cdot_n$ על־ידי הסימונים האריתמטיים הרגילים $+$ ו־$\cdot$ (או כתיבת גורמי המכפלה בסמיכות), בהתאמה. כמו כן, ניתן לפרש שקילויות מודולו n כמשוואות ב־$\mathbf{Z}_n$. לדוגמה, שתי הטענות שלהלן שקולות זו לזו:

$$ax \equiv b \pmod n$$

$$[a]_n \cdot_n [x]_n = [b]_n$$

כדי להקל עוד יותר את הכתיבה, נציין לעתים את החבורה $(S, \oplus)$ פשוט על־ידי S כאשר הפעולה ברורה מן ההקשר. אנו עשויים אפוא להתייחס אל החבורות $(\mathbf{Z}_n, +_n)$ ו־$(\mathbf{Z}_n^*, \cdot_n)$ כאל $\mathbf{Z}_n$ ו־$\mathbf{Z}_n^*$, בהתאמה.

ההופכי (הכפלי) של איבר a מסומן על־ידי $(a^{-1} \bmod n)$. חילוק ב־$\mathbf{Z}_n^*$ מוגדר על־ידי המשוואה $a/b \equiv ab^{-1} \pmod n$. לדוגמה, בחבורה $\mathbf{Z}_{15}^*$, נקבל $7^{-1} \equiv 13 \pmod{15}$, שכן $7 \cdot 13 \equiv 91 \equiv 1 \pmod{15}$, ולפיכך $4/7 \equiv 4 \cdot 13 \equiv 7 \pmod{15}$.

הגודל של $\mathbf{Z}_n^*$ מסומן על־ידי $\phi(n)$. פונקציה זו, הידועה בשם **פונקציית פי של אוילר** (Euler's phi function), מקיימת את המשוואה:

$$\phi(n) = n \prod_{p \mid n} \left(1 - \frac{1}{p}\right) \tag{33.20}$$

כאשר p עובר על פני כל הראשוניים המחלקים את n (כולל n עצמו, אם n ראשוני). לא נוכיח כאן נוסחה זו. אינטואיטיבית, מתחילים עם רשימה של n השאריות $\{0, 1, \ldots, n-1\}$ ואז, עבור כל p ראשוני המחלק את n, מוחקים מן הרשימה את כל הכפולות של p. לדוגמה, מאחר שהמחלקים הראשוניים של 45 הם 3 ו־5, נקבל:

$$\begin{aligned}\phi(45) &= 45\left(1 - \frac{1}{3}\right)\left(1 - \frac{1}{5}\right) \\ &= 45\left(\frac{2}{3}\right)\left(\frac{4}{5}\right) \\ &= 24\end{aligned}$$

אם p ראשוני, אזי $\mathbf{Z}_p^* = \{1, 2, \ldots, p-1\}$, ולכן:

$$\phi(p) = p - 1 \tag{33.21}$$

אם n פריק, אזי $\phi(n) < n - 1$.

תת־חבורות

אם $(S, \oplus)$ היא חבורה, $S' \subseteq S$, ו־$(S', \oplus)$ גם היא חבורה, נאמר כי $(S', \oplus)$ היא **תת־חבורה** (subgroup) של $(S, \oplus)$. לדוגמה, השלמים הזוגיים מהווים תת־חבורה של השלמים תחת פעולת החיבור. המשפט הבא מספק כלי שימושי לזיהוי תת־חבורות.

משפט 33.14 (תת־קבוצה סגורה לא ריקה של חבורה סופית היא תת־חבורה)
אם $(S, \oplus)$ היא חבורה סופית ו־S' היא תת־קבוצה לא ריקה כלשהי של S כך ש־$a \oplus b \in S'$ לכל $a, b \in S'$, אזי $(S', \oplus)$ היא תת־חבורה של $(S, \oplus)$.

הוכחה הקורא יתבקש להוכיח את המשפט בתרגיל 33.3-2. ■

לדוגמה, הקבוצה $\{0, 2, 4, 6\}$ יוצרת תת־חבורה של $\mathbf{Z}_8$, מאחר שהיא סגורה תחת הפעולה $+$ (כלומר, היא סגורה תחת $+_8$).

המשפט הבא מציב הגבלה שימושית ביותר על גודלה של תת־חבורה.

משפט 33.15 (משפט לאגראנז' (Lagrange))
אם $(S, \oplus)$ היא חבורה סופית ו־$(S', \oplus)$ היא תת־חבורה של $(S, \oplus)$, אזי $|S'|$ הוא מחלק של $|S|$. ■

תת־חבורה S' של חבורה S נקראת **תת־חבורה ממש** (proper subgroup) אם $S' \neq S$. המסקנה הבאה תשמש אותנו בעת ניתוח מבחן הראשוניות של מילר ורבין (Miller-Rabin primality test) בסעיף 33.8.

מסקנה 33.16
אם S' היא תת־חבורה ממש של חבורה סופית S, אזי $|S'| \leq |S|/2$. ■

תת־חבורות הנוצרות על־ידי איבר

משפט 33.14 מספק דרך מעניינת ליצירת תת־חבורה של חבורה סופית $(S, \oplus)$: בוחרים איבר a ולוקחים את כל האיברים שניתן ליצור מ־a באמצעות הפעולה $\oplus$. ובפירוט, נגדיר $a^{(k)}$, עבור $k \geq 1$, על־ידי:

$$a^{(k)} = \bigoplus_{i=1}^{k} a = \underbrace{a \oplus a \oplus \cdots \oplus a}_{k}$$

לדוגמה, אם נבחר $a = 2$ בחבורה $\mathbf{Z}_6$, אזי הסדרה $a^{(1)}, a^{(2)}, \ldots$ היא:

$$2, 4, 0, 2, 4, 0, 2, 4, 0, \ldots$$

בחבורה $\mathbf{Z}_n$ מתקיים $a^{(k)} = ka \bmod n$, ובחבורה $\mathbf{Z}_n^*$ מתקיים $a^{(k)} = a^k \bmod n$.
התת־חבורה הנוצרת על־ידי a (the subgroup generated by a) מסומנת על־ידי $\langle a \rangle$ או $(\langle a \rangle, \oplus)$ ומוגדרת על־ידי:

$$\langle a \rangle = \{a^{(k)} : k \geq 1\}$$

אנו אומרים ש־a **יוצר** (generates) את התת־חבורה $\langle a \rangle$ או ש־a הוא **האיבר היוצר** (generator) של $\langle a \rangle$. מאחר ש־S סופית, $\langle a \rangle$ היא תת־קבוצה סופית של S, אולי S כולה. מאחר שהאסוציאטיביות של $\oplus$ גוררת:

$$a^{(i)} \oplus a^{(j)} = a^{(i+j)}$$

הרי ש־$\langle a \rangle$ סגורה, ולכן, על פי משפט 33.14, $\langle a \rangle$ היא תת־חבורה של S. לדוגמה, ב־$\mathbf{Z}_6$ נקבל:

$$\langle 0 \rangle = \{0\}$$
$$\langle 1 \rangle = \{0, 1, 2, 3, 4, 5\}$$
$$\langle 2 \rangle = \{0, 2, 4\}$$

באופן דומה, ב־$\mathbf{Z}_7^*$ נקבל:

$$\langle 1 \rangle = \{1\}$$
$$\langle 2 \rangle = \{1, 2, 4\}$$
$$\langle 3 \rangle = \{1, 2, 3, 4, 5, 6\}$$

הסדר (order) של a (בחבורה S) המסומן על־ידי $\mathrm{ord}(a)$, מוגדר כאיבר הקטן ביותר $t > 0$ כך ש־$a^{(t)} = e$.

משפט 33.17
עבור כל חבורה סופית $(S, \oplus)$ וכל $a \in S$, הסדר של איבר שווה לגודל התת־חבורה שהוא יוצר: $\mathrm{ord}(a) = |\langle a \rangle|$.

הוכחה יהי $t = \mathrm{ord}(a)$. מאחר ש־$a^{(t)} = e$ ומתקיים $a^{(t+k)} = a^{(t)} \oplus a^{(k)} = a^{(k)}$ עבור k, אזי עבור $k \geq 1$, קיים $j < i$ שעבורו $a^{(i)} = a^{(j)}$. לכן, אחרי $a^{(t)}$ לא נוצרים יותר איברים חדשים, ומתקיים $\langle a \rangle = \{a^{(1)}, a^{(2)}, \ldots, a^{(t)}\}$. כדי להראות כי $t = |\langle a \rangle|$, נניח בדרך השלילה כי $a^{(i)} = a^{(j)}$ עבור איזשהם i ו־j המקיימים $1 \leq j < i \leq t$. אזי, $a^{(i+k)} = a^{(j+k)}$ עבור $k \geq 0$. אבל מכך נובע כי $a^{(i+(t-j))} = a^{(j+(t-j))}$, וזוהי סתירה, שכן $i + (t - j) < t$ אבל t הוא הערך החיובי הקטן ביותר שעבורו $a^{(t)} = e$. לכן, האיברים $a^{(1)}, a^{(2)}, \ldots, a^{(t)}$ שונים כולם זה מזה, ו־$t = |\langle a \rangle|$. ■

מסקנה 33.18
הסדרה $a^{(1)}, a^{(2)}, \ldots$ היא מחזורית עם מחזור $t = \mathrm{ord}(a)$; כלומר, $a^{(i)} = a^{(j)}$ אם ורק אם $i \equiv j \pmod{t}$. ■

באופן קונסיסטנטי עם המסקנה שלעיל, נוכל להגדיר את $a^{(0)}$ כ־e ואת $a^{(i)}$ כ־$a^{(i \bmod t)}$ עבור כל השלמים i.

מסקנה 33.19
אם $(S, \oplus)$ היא חבורה סופית עם איבר יחידה e, אזי לכל $a \in S$ מתקיים:

$$a^{(|S|)} = e$$

הוכחה משפט לאגראנז' גורר כי $\mathrm{ord}(a) \mid |S|$, ולכן $|S| \equiv 0 \pmod{t}$, כאשר $t = \mathrm{ord}(a)$. ■

תרגילים

33.3-1

רשום את לוחות הפעולה עבור החבורות $(\mathbf{Z}_4, +_4)$ ו־$(\mathbf{Z}_5^*, \cdot_5)$. הראה שהחבורות הללו איזומורפיות על־ידי הצגת התאמה חד־חד־ערכית α בין איברי החבורות, המקיימת $\alpha(a) \cdot \alpha(b) \equiv \alpha(c) \pmod 5$ אם ורק אם $a + b \equiv c \pmod 4$.

33.3-2

הוכח את משפט 33.14.

33.3-3

הראה שאם p ראשוני ו־e הוא שלם חיובי, אזי:

$$\phi(p^e) = p^{e-1}(p-1)$$

33.3-4

הראה שעבור כל $n > 1$ ועבור כל $a \in \mathbf{Z}_n^*$, הפונקציה $f_a : \mathbf{Z}_n^* \to \mathbf{Z}_n^*$ המוגדרת על־ידי $f_a(x) = ax \bmod n$ היא תמורה של $\mathbf{Z}_n^*$.

33.3-5

ערוך רשימה של כל התת־חבורות של $\mathbf{Z}_9$ ושל $\mathbf{Z}_{13}^*$.

33.4 פתרון משוואות לינאריות מודולריות

נתבונן עתה בבעיה של מציאת פתרונות למשוואה:

$$ax \equiv b \pmod n \tag{33.22}$$

כאשר $n > 0$. בעיה זו היא בעלת חשיבות מעשית רבה. אנו מניחים כי a, b ו־n נתונים, ועלינו למצוא את ה־x־ים, מודולו n, המספקים את משוואה (33.22). ייתכנו אפס, אחד או יותר פתרונות כאלה.

נסמן ב־$\langle a \rangle$ את התת־חבורה של $\mathbf{Z}_n$ הנוצרת על־ידי a. מכיוון שמתקיים $\langle a \rangle = \{a^{(x)} : x > 0\} = \{ax \bmod n : x > 0\}$, למשוואה (33.22) יש פתרון אם ורק אם $b \in \langle a \rangle$. משפט לאגראנז' (משפט 33.15) קובע ש־$|\langle a \rangle|$ הוא בהכרח מחלק של n. המשפט הבא נותן אפיון מדויק של $\langle a \rangle$.

משפט 33.20

עבור שלמים חיוביים כלשהם a ו־n, אם $d = \gcd(a, n)$, אזי:

$$\langle a \rangle = \langle d \rangle = \{0, d, 2d, \ldots, ((n/d) - 1)d\} \tag{33.23}$$

ולכן:

$$|\langle a \rangle| = n/d$$

הוכחה נתחיל בכך שנראה כי $d \in \langle a \rangle$. נשים לב שהפלטים של EXTENDED-EUCLID(a, n) הם שלמים x' ו־y' כך ש־$ax' + ny' = d$. לכן, $ax' \equiv d \pmod n$, ומכאן ש־$d \in \langle a \rangle$.

מאחר ש־$d \in \langle a \rangle$, נובע מכך שכל כפולה של d שייכת ל־$\langle a \rangle$, משום שכפולה של כפולה של a היא כפולה של a. לכן, $\langle a \rangle$ מכילה כל איבר של $\{0, d, 2d, \ldots, ((n/d) - 1)d\}$, דהיינו, $\langle d \rangle \subseteq \langle a \rangle$.

עתה נראה כי $\langle a \rangle \subseteq \langle d \rangle$. אם $m \in \langle a \rangle$, אזי קיים שלם x שעבורו $m = ax \bmod n$, ולכן קיים שלם y שעבורו $m = ax + ny$. אולם $d \mid a$ ו־$d \mid n$, ולכן $d \mid m$ על פי משוואה (33.6). לכן, $m \in \langle d \rangle$.

נצרף את התוצאות הללו ונקבל כי $\langle d \rangle = \langle a \rangle$. כדי לראות כי $|\langle a \rangle| = n/d$, נשים לב שבין 0 ל־$(n-1)$, כולל, קיימות בדיוק n/d כפולות של d. ■

מסקנה 33.21

המשוואה $ax \equiv b \pmod n$ פתירה עבור הנעלם x אם ורק אם $\gcd(a, n) \mid b$. ■

מסקנה 33.22

למשוואה $ax \equiv b \pmod n$ יש d פתרונות שונים מודולו n, כאשר $d = \gcd(a, n)$, או שאין לה פתרון כלל.

הוכחה אם למשוואה $ax \equiv b \pmod n$ יש פתרון, אזי $b \in \langle a \rangle$. על פי מסקנה 33.18, הסדרה $ai \bmod n$, עבור $i = 0, 1, \ldots$, היא מחזורית עם מחזור של $|\langle a \rangle| = n/d$. אם $b \in \langle a \rangle$, אזי b מופיע בדיוק d פעמים בסדרה $ai \bmod n$, עבור $i = 0, 1, \ldots, n-1$, שכן כאשר i גדל מ־0 ל־$(n-1)$, גוש הערכים $\langle a \rangle$, שאורכו n/d, חוזר על עצמו בסדרה בדיוק d פעמים. האינדקסים x של d המיקומים שבהם מופיע b בסדרה הם הפתרונות של המשוואה $ax \equiv b \pmod n$. ■

משפט 33.23

יהי $d = \gcd(a, n)$, ונניח כי $d = ax' + ny'$ עבור איזשהם שלמים x' ו־y' (למשל, אלה המחושבים על־ידי השגרה EXTENDED-EUCLID). אם $d \mid b$, אזי אחד הפתרונות למשוואה $ax \equiv b \pmod n$ הוא הערך x_0 שעבורו:

$$x_0 = x'(b/d) \bmod n$$

הוכחה מאחר ש־$ax' \equiv d \pmod n$, אזי:

$$\begin{aligned} ax_0 &\equiv ax'(b/d) \pmod n \\ &\equiv d(b/d) \pmod n \\ &\equiv b \pmod n \end{aligned}$$

ולכן x_0 הוא פתרון של המשוואה $ax \equiv b \pmod n$. ■

משפט 33.24

נניח שהמשוואה $ax \equiv b \pmod n$ פתירה (כלומר, $d \mid b$ כאשר $d = \gcd(a, n)$) ונניח כי x_0 הוא פתרון כלשהו של המשוואה. אזי, למשוואה זו יש בדיוק d פתרונות, מודולו n, הנתונים על־ידי $x_i = x_0 + i(n/d)$ עבור $i = 1, 2, \ldots, d-1$.

הוכחה מאחר ש-$n/d > 0$ וכן $0 \le i(n/d) < n$ עבור $i = 0, 1, 2, \ldots, d-1$, הערכים $x_0, x_1, \ldots, x_{d-1}$ שונים כולם זה מזה, מודולו n. על פי מחזוריות הסדרה $ai \bmod n$ (משפט 33.18), אם x_0 הוא פתרון של $ax \equiv b \pmod n$, אזי כל x_i הוא פתרון. על פי משפט 33.22, קיימים בדיוק d פתרונות, ולכן $x_0, x_1, \ldots, x_{d-1}$ הם בהכרח כל הפתרונות. ■

בכך סיימנו את פיתוח המתמטיקה הדרושה לפתרון המשוואה $ax \equiv b \pmod n$; האלגוריתם שלהלן מדפיס את כל הפתרונות למשוואה זו. הקלטים a ו-b הם שלמים כלשהם, ו-n הוא שלם חיובי כלשהו.

```
MODULAR-LINEAR-EQUATION-SOLVER(a, b, n)
1  (d, x', y') ← EXTENDED-EUCLID(a, n)
2  if d | b
3     then x_0 ← x'(b/d) mod n
4          for i ← 0 to d − 1
5              do print (x_0 + i(n/d)) mod n
6  else print "no solutions"
```

כדוגמה לפעולתה של השגרה, נתבונן במשוואה $14x \equiv 30 \pmod{100}$ (כאן, $a = 14$, $b = 30$ ו-$n = 100$). הקריאה ל-EXTENDED-EUCLID בשורה 1 מחזירה $(d, x, y) = (2, -7, 1)$. מכיוון ש-$2 \mid 30$, מבוצעות שורות 3-5. בשורה 3 מחשבים את $x_0 = (-7)(15) \bmod 100 = 95$. הלולאה בשורות 4-5 מדפיסה את שני הפתרונות: 95 ו-45.

השגרה MODULAR-LINEAR-EQUATION-SOLVER פועלת באופן הבא: בשורה 1 מחשבים את $d = \gcd(a, n)$ ואת שני הערכים x' ו-y' שעבורם $d = ax' + ny'$, ובכך מראים כי x' הוא פתרון של המשוואה $ax' \equiv d \pmod n$. אם d אינו מחלק את b, אזי, על פי מסקנה 33.21, למשוואה $ax \equiv b \pmod n$ אין פתרון. שורה 2 בודקת אם $d \mid b$; אם לא, שורה 6 מדווחת שלמשוואה אין פתרון. אחרת, שורה 3 מחשבת פתרון x_0 למשוואה (33.22), בהתאם למשפט 33.23. בהינתן פתרון אחד, משפט 33.24 קובע כי ניתן לקבל את $d - 1$ הפתרונות האחרים על-ידי הוספת כפולות של (n/d), מודולו n. לולאת ה-**for** בשורות 4-5 מדפיסה את d הפתרונות, החל מ-x_0 ובמרווחים של (n/d), מודולו n.

זמן הריצה של MODULAR-LINEAR-EQUATION-SOLVER הוא $O(\lg n + \gcd(a, c))$ פעולות אריתמטיות, שכן EXTENDED-EUCLID מבצעת $O(\lg n)$ פעולות אריתמטיות וכל איטרציה של לולאת ה-**for** בשורות 4-5 מבצעת מספר קבוע של פעולות אריתמטיות.

המסקנות הבאות ממשפט 33.24 מציגות מקרים פרטיים שיש בהם עניין מיוחד.

מסקנה 33.25
עבור כל $n > 1$, אם $\gcd(a, n) = 1$, אזי למשוואה $ax \equiv b \pmod n$ יש פתרון יחיד מודולו n. ■

אם $b = 1$, מקרה שכיח שיש בו עניין לא מבוטל, אזי ה-x המבוקש הוא **הופכי כפלי** (multiplicative inverse) של a, מודולו n.

מסקנה 33.26
לכל $n > 1$, אם $\gcd(a, n) = 1$, אזי למשוואה:

$$ax \equiv 1 \pmod n \tag{33.24}$$

יש פתרון יחיד מודולו n. אחרת, למשוואה אין פתרון.

מסקנה 33.26 מאפשרת לנו להשתמש בסימון $(a^{-1} \bmod n)$ לציון ההופכי הכפלי (היחיד) של a, מודולו n, כאשר a ו-n זרים זה לזה. אם $\gcd(a, n) = 1$, אזי אחד הפתרונות למשוואה $ax \equiv 1 \pmod n$ הוא השלם x שמחזירה EXTENDED-EUCLID, שכן המשוואה:

$$\gcd(a, n) = 1 = ax + ny$$

גוררת $ax \equiv 1 \pmod n$. לכן, באמצעות השגרה EXTENDED-EUCLID ניתן לחשב ביעילות את $(a^{-1} \bmod n)$.

תרגילים

33.4-1
מצא את כל הפתרונות של המשוואה $35x \equiv 10 \pmod{50}$.

33.4-2
הוכח שכאשר $\gcd(a, n) = 1$, המשוואה $ax \equiv ay \pmod n$ גוררת $x \equiv y \pmod n$. הראה שהתנאי $\gcd(a, n) = 1$ הוא הכרחי על-ידי הצגת דוגמה נגדית שבה $\gcd(a, n) > 1$.

33.4-3
נניח שאנו מבצעים את השינוי הבא בשורה 3 של MODULAR-LINEAR-EQUATION-SOLVER:

```
3     then x_0 ← x'(b/d) mod (n/d)
```

האם השגרה תפעל כהלכה לאחר השינוי? הסבר מדוע או מדוע לא.

33.4-4 *
יהי $f(x) \equiv f_0 + f_1 x + \cdots + f_t x^t \pmod p$ פולינום ממעלה t עם מקדמים f_i הלקוחים מ-$\mathbf{Z}_p$, כאשר p ראשוני. אנו אומרים ש-$a \in \mathbf{Z}_p$ הוא **אפס** של f אם $f(a) \equiv 0 \pmod p$. הוכח שאם a הוא אפס של f, אזי קיים פולינום $g(x)$ ממעלה $t - 1$ שעבורו $f(x) \equiv (x - a)g(x) \pmod p$. הוכח באינדוקציה על t שלפולינום $f(x)$ ממעלה t יכולים להיות לכל היותר t אפסים שונים מודולו p ראשוני.

33.5 משפט השאריות הסיני

בשנת 100 לספירה לערך פתר המתמטיקאי הסיני סון-טסו (Sun-Tsu) את הבעיה של מציאת אותם שלמים x אשר השאריות המתקבלות מחלוקתם ב-3, 5 ו-7 הן 2, 3 ו-2, בהתאמה. אחד הפתרונות הוא $x = 23$; כל הפתרונות הם מהצורה $105k + 23$, עבור k שלם כלשהו. "משפט השאריות הסיניי" קובע התאמה בין מערכת של משוואות מודולו סדרת שלמים זרים זה לזה (3, 5 ו-7, למשל) לבין משוואה מודולו המכפלה של מספרים אלה (105, למשל).

למשפט השאריות הסיני שני שימושים עיקריים. יהי $n = n_1 n_2 \cdots n_k$ שלם, כאשר הגורמים n_i זרים בזוגות. ראשית, משפט השאריות הסיני הוא "משפט מבנה" תיאורי: הוא מתאר את המבנה של $\mathbf{Z}_n$ כזהה לזה של המכפלה הקרטזית $\mathbf{Z}_{n_1} \times \mathbf{Z}_{n_2} \times \cdots \times \mathbf{Z}_{n_k}$, עם חיבור וכפל נקודתיים עבור כל קואורדינטה בנפרד, המתבצעים מודולו n_i בקואורדינטה ה-i. שנית, תיאור זה יכול להניב

לעתים קרובות אלגוריתמים יעילים, שכן עבודה בכל אחת מן המערכות $\mathbf{Z}_{n_i}$ עשויה להיות יעילה יותר (במונחי פעולות על סיביות) מאשר עבודה מודולו n.

משפט 33.27 (משפט השאריות הסיני)
יהי $n = n_1 n_2 \cdots n_k$, כאשר ה־n_i זרים בזוגות. נתבונן בהתאמה:

$$a \leftrightarrow (a_1, a_2, \ldots, a_k) \tag{33.25}$$

כאשר $a \in \mathbf{Z}_n$, $a_i \in \mathbf{Z}_{n_i}$, ועבור $i = 1, 2, \ldots, k$ מתקיים:

$$a_i = a \bmod n_i$$

אזי, המיפוי (33.25) הוא התאמה חד־חד ערכית בין $\mathbf{Z}_n$ והמכפלה הקרטזית $\mathbf{Z}_{n_1} \times \mathbf{Z}_{n_2} \times \cdots \times \mathbf{Z}_{n_k}$. פעולות המבוצעות על איברי $\mathbf{Z}_n$ ניתנות לביצוע באופן שקול על ה־k־יות המתאימות, על־ידי כך שמבצעים את הפעולות בכל קואורדינטה בנפרד במערכת המתאימה. כלומר, אם:

$$a \leftrightarrow (a_1, a_2, \ldots, a_k)$$
$$b \leftrightarrow (b_1, b_2, \ldots, b_k)$$

אזי:

$$(a + b) \bmod n \leftrightarrow ((a_1 + b_1) \bmod n_1, \ldots, (a_k + b_k) \bmod n_k) \tag{33.26}$$

$$(a - b) \bmod n \leftrightarrow ((a_1 - b_1) \bmod n_1, \ldots, (a_k - b_k) \bmod n_k) \tag{33.27}$$

$$(ab) \bmod n \leftrightarrow (a_1 b_1 \bmod n_1, \ldots, a_k b_k \bmod n_k) \tag{33.28}$$

הוכחה ההמרות בין שני הייצוגים ישירות למדי. המעבר מ־a ל־$(a_1, a_2, \ldots, a_k)$ דורש k פעולות חילוק בלבד. חישוב a מתוך הקלטים $(a_1, a_2, \ldots, a_k)$ קל כמעט באותה מידה, ומתבצע בעזרת הנוסחה שלהלן. יהי $m_i = n/n_i$ עבור $i = 1, 2, \ldots, k$. נשים לב כי $m_i = n_1 n_2 \cdots n_{i-1} n_{i+1} \cdots n_k$, כך ש־$m_i \equiv 0 \pmod{n_j}$ עבור כל $j \neq i$. לכן, אם נסמן:

$$c_i = m_i (m_i^{-1} \bmod n_i) \tag{33.29}$$

עבור $i = 1, 2, \ldots, k$, נקבל:

$$a \equiv (a_1 c_1 + a_2 c_2 + \cdots + a_k c_k) \pmod{n} \tag{33.30}$$

משוואה (33.29) מוגדרת היטב שכן m_i ו־n_i זרים זה לזה (על פי משפט 33.6), ולכן מסקנה 33.26 גוררת כי $(m_i^{-1} \bmod n_i)$ מוגדר. כדי לאמת את משוואה (33.30), נשים לב כי $c_j \equiv m_j \equiv 0 \pmod{n_i}$ אם $j \neq i$, וכי $c_i \equiv 1 \pmod{n_i}$. לכן מתקבלת ההתאמה:

$$c_i \leftrightarrow (0, 0, \ldots, 0, 1, 0, \ldots, 0)$$

בין c_i לווקטור שבו מופיע 1 בקואורדינטה ה־i ו־0 בכל הקואורדינטות האחרות. ה־c_i יוצרים אפוא במובן מסוים "בסיס" לייצוג. לכן לכל i מתקיים:

$$\begin{aligned} a &\equiv a_i c_i && \pmod{n_i} \\ &\equiv a_i m_i (m_i^{-1} \bmod n_i) && \pmod{n_i} \\ &\equiv a_i && \pmod{n_i} \end{aligned}$$

מאחר שניתן לבצע המרה בשני הכיוונים, ההתאמה היא חד־חד ערכית. משוואות (33.26)-(33.28) נובעות ישירות מתרגיל 33.1-6, שכן $x \bmod n_i = (x \bmod n) \bmod n_i$ עבור כל x ו־$i = 1, 2, \ldots, k$. ■

המסקנות הבאות ישמשו אותנו בשלב מאוחר יותר בפרק זה.

מסקנה 33.28
אם $n_1, n_2, \ldots, n_k$ זרים בזוגות ו־$n = n_1 n_2 \cdots n_k$, אזי עבור כל סדרת שלמים $a_1, a_2, \ldots, a_k$, מערכת המשוואות:

$$x \equiv a_i \pmod{n_i}$$

עבור $i = 1, 2, \ldots, k$, היא בעלת פתרון יחיד מודולו n עבור הנעלם x. ■

מסקנה 33.29
אם $n_1, n_2, \ldots, n_k$ זרים בזוגות ו־$n = n_1 n_2 \cdots n_k$, אזי לכל x ו־a שלמים מתקיים:

$$x \equiv a \pmod{n_i}$$

עבור $i = 1, 2, \ldots, k$, אם ורק אם:

$$x \equiv a \pmod{n}$$

■

כדוגמה למשפט השאריות הסיני, נניח כי נתונות לנו שתי המשוואות:

$$a \equiv 2 \pmod 5$$
$$a \equiv 3 \pmod{13}$$

שבהן $a_1 = 2$, $n_1 = m_2 = 5$, $a_2 = 3$ ו־$n_2 = m_1 = 13$, וברצוננו לחשב את $a \bmod 65$, שכן $n = 65$. מכיוון ש־$13^{-1} \equiv 2 \pmod 5$ וכן $5^{-1} \equiv 8 \pmod{13}$, נקבל:

$$c_1 = 13\,(2 \bmod 5) = 26$$
$$c_2 = 5\,(8 \bmod 13) = 40$$

וכן:

$$\begin{aligned} a &\equiv 2 \cdot 26 + 3 \cdot 40 && \pmod{65} \\ &\equiv 52 + 120 && \pmod{65} \\ &\equiv 42 && \pmod{65} \end{aligned}$$

באיור 33.3 מודגם משפט השאריות הסיני מודולו $n = 65$.

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	40	15	55	30	5	45	20	60	35	10	50	25
1	26	1	41	16	56	31	6	46	21	61	36	11	51
2	52	27	2	42	17	57	32	7	47	22	62	37	12
3	13	53	28	3	43	18	58	33	8	48	23	63	38
4	39	14	54	29	4	44	19	59	34	9	49	24	64

איור 33.3 הדוגמה של משפט השאריות הסיני עבור $n_1 = 5$ ו־$n_2 = 13$. בדוגמה זו, $c_1 = 26$ ו־$c_2 = 40$. בתא שבשורה i ובעמודה j מופיע ערכו של a, מודולו 65, שמקיים $a \bmod 5 = i$ ו־$a \bmod 13 = j$. נשים לב כי התא שבשורה 0 ובעמודה 0 מכיל 0. באופן דומה, התא שבשורה 4 ובעמודה 12 מכיל 64 (השקול ל־(−1)). מאחר ש־$c_1 = 26$, כאשר נעים שורה אחת כלפי מטה, ערכו של a גדל ב־26. באופן דומה, $c_2 = 40$ פירושו שכאשר נעים טור אחד ימינה, a גדל ב־40. הגדלת a ב־1 מתאימה לתנועה באלכסון למטה וימינה, כאשר מגיעים לשורה האחרונה ממשיכים בטור הבא ימינה בשורה העליונה, וכאשר מגיעים לטור האחרון ממשיכים בשורה הבאה בטור הראשון משמאל.

משפט 33.34
אם p הוא ראשוני אי־זוגי ו־$e \geq 1$, אזי למשוואה:

$$x^2 \equiv 1 \pmod{p^e} \tag{33.34}$$

יש רק שני פתרונות, והם $x = 1$ ו־$x = -1$.

הוכחה יהי $n = p^e$. משפט 33.32 גורר כי $\mathbf{Z}_n^*$ מכילה שורש פרימיטיבי g. את משוואה (33.34) ניתן לכתוב בצורה:

$$(g^{\text{ind}_{n,g}(x)})^2 \equiv g^{\text{ind}_{n,g}(1)} \pmod{n} \tag{33.35}$$

אם נשים לב כי $\text{ind}_{n,g}(1) = 0$, נראה שמשפט 33.33 גורר כי משוואה (33.35) שקולה למשוואה:

$$2 \cdot \text{ind}_{n,g}(x) \equiv 0 \pmod{\phi(n)} \tag{33.36}$$

כדי לפתור משוואה זו עבור הנעלם $\text{ind}_{n,g}(x)$, נשתמש בשיטות שתוארו בסעיף 33.4. יהי $d = \gcd(2, \phi(n)) = \gcd(2, (p-1)p^{e-1}) = 2$. אם נזכור כי $d \mid 0$, הרי שעל פי משפט 33.24, למשוואה (33.36) יש בדיוק $d = 2$ פתרונות. לכן, למשוואה (33.34) יש בדיוק שני פתרונות, והם $x = 1$ ו־$x = -1$, כפי שניתן לוודא על־ידי הצבה. ■

מספר x הוא **שורש ריבועי לא טריוויאלי של 1, מודולו n** (nontrivial square root of 1 modulo n) אם הוא מספק את המשוואה $x^2 \equiv 1$ אבל x אינו שקול, מודולו n, לאף אחד משני השורשים הריבועיים "הטריוויאליים" 1 או 1–. לדוגמה, 6 הוא שורש ריבועי לא טריוויאלי של 1, מודולו 35. המסקנה שלהלן ממשפט 33.34 תשתמש אותנו בהוכחת נכונותה של השגרה לבדיקת ראשוניות של מילר ורבין, בסעיף 33.8.

מסקנה 33.35
אם קיים שורש ריבועי לא טריוויאלי של 1, מודולו n, אזי n פריק.

הוכחה ממשפט 33.34 נובע שאם קיים שורש ריבועי לא טריוויאלי של 1, מודולו n, אזי n אינו יכול להיות ראשוני או חזקה של ראשוני. יתר על כן, אם קיים שורש ריבועי לא טריוויאלי של 1, מודולו n, אזי בהכרח $n > 1$. לכן, בהכרח n פריק. ■

העלאה בחזקה באמצעות העלאה חוזרת בריבוע

אחת הפעולות השכיחות בעת ביצוע חישובים בתורת המספרים היא העלאת מספר בחזקה, מודולו מספר אחר, פעולה הידועה בשם **העלאה מודולרית בחזקה** (modular exponentiation). ביתר דיוק, ברצוננו למצוא דרך יעילה לחישוב $a^b \bmod n$, כאשר a ו־b הם שלמים אי־שליליים ו־n הוא שלם חיובי. העלאה מודולרית בחזקה היא גם פעולה חיונית בשגרות רבות לבדיקת ראשוניות וכן במערכת ההצפנה RSA, המבוססת על מפתח ציבורי. השיטה של **העלאה חוזרת בריבוע** (repeated squaring) פותרת ביעילות בעיה זו תוך שימוש בייצוג הבינרי של b.

יהי $\langle b_k, b_{k-1}, \ldots, b_1, b_0 \rangle$ הייצוג הבינרי של b. (כלומר, הייצוג הבינרי של b הוא בן $k+1$ סיביות, b_k היא הסיבית המשמעותית ביותר ו־b_0 — הסיבית הכי פחות משמעותית.) השגרה שלהלן מחשבת את $a^c \bmod n$, כאשר c גדל מ־0 עד b באמצעות הכפלות פי שניים והוספות של 1.


```
MODULAR-EXPONENTIATION(a, b, n)
1  c ← 0
2  d ← 1
3  let ⟨b_k, b_{k-1}, ..., b_0⟩ be the binary representation of b
4  for i ← k downto 0
5      do c ← 2c
6         d ← (d · d) mod n
7         if b_i = 1
8            then c ← c + 1
9                 d ← (d · a) mod n
10 return d
```

כל מעריך המחושב במהלך השגרה כפול בערכו מן המעריך הקודם או גדול ממנו ב־1; הייצוג הבינרי של b נקרא משמאל לימין, וערכה של הסיבית b_i קובע אילו פעולות יש לבצע. כל איטרציה של הלולאה משתמשת באחת משתי הזהויות:

$$a^{2c} \bmod n = (a^c)^2 \bmod n$$
$$a^{2c+1} \bmod n = a \cdot (a^c)^2 \bmod n$$

והדבר תלוי בכך אם $b_i = 0$ או $b_i = 1$, בהתאמה. השימוש המרכזי בהעלאה בריבוע בכל איטרציה מסביר את השם "העלאה חוזרת בריבוע". מיד לאחר הקריאה והעיבוד של סיבית b_i, ערכו של c שווה לרישא $\langle b_k, b_{k-1}, \ldots, b_i \rangle$ של הייצוג הבינרי של b. לדוגמה, עבור $a = 7$, $b = 560$ ו־$n = 561$, האלגוריתם מחשב את סדרת הערכים מודולו 561 המוצגים באיור 33.4; סדרת המעריכים המשמשים בחישוב מופיעה בשורה c בטבלה.

i	9	8	7	6	5	4	3	2	1	0
b_i	1	0	0	0	1	1	0	0	0	0
c	1	2	4	8	17	35	70	140	280	560
d	7	49	157	526	160	241	298	166	67	1

איור 33.4 התוצאות שמפיקה MODULAR-EXPONENTIATION בעת חישוב $a^b \pmod n$, כאשר $a = 7$, $\langle 1000110000 \rangle = b = 560$ ו־$n = 561$. הערכים המוצגים הם אלה המתקבלים לאחר כל איטרציה של לולאת ה־**for**. התוצאה הסופית היא 1.

המשתנה c אינו נחוץ בעצם לאלגוריתם, והוא נכלל כאן לצורך ההסבר בלבד: האלגוריתם משמר את האינוואריאנטה $d = a^c \bmod n$ כאשר הוא מגדיל את c על־ידי הכפלות פי 2 ותוספות של 1 עד אשר $c = b$. אם הקלטים a, b ו־n הם מספרים בני β סיביות, אזי המספר הכולל של פעולות אריתמטיות המבוצעות במהלך השגרה הוא $O(\beta)$ והמספר הכולל של פעולות על סיביות הוא $O(\beta^3)$.

תרגילים

33.6-1

ערוך טבלה המראה את הסדר של כל איבר ב־$\mathbf{Z}_{11}^*$. בחר את השורש הפרימיטיבי הקטן ביותר g, וחשב את טבלת הערכים $\text{ind}_{11,g}(x)$ עבור כל $x \in \mathbf{Z}_{11}^*$.

33.6-2
כתוב אלגוריתם להעלאה מודולרית בחזקה הקורא את הסיביות של b מימין לשמאל במקום משמאל לימין.

33.6-3
בהנחה ש־$\phi(n)$ ידוע, הסבר כיצד מחשבים את $a^{-1} \bmod n$ עבור $a \in \mathbf{Z}_n^*$ כלשהו באמצעות השגרה MODULAR-EXPONENTIATION.

33.7 מערכת ההצפנה RSA – הצפנה באמצעות מפתח ציבורי

באמצעות מערכת הצפנה המבוססת על מפתח ציבורי ניתן לקודד הודעות הנשלחות בין שני צדדים כך שמצותת המאזין להודעות המוצפנות לא יוכל לפענח אותן. מערכת הצפנה המבוססת על מפתח ציבורי גם מאפשרת לשולח להוסיף בסופה של הודעה אלקטרונית "חתימה דיגיטלית" שאינה ניתנת לזיוף. חתימה כזאת היא הגרסה האלקטרונית של חתימה בכתב יד על מסמך כתוב. כל אחד יכול לבדוק אותה, איש אינו יכול לזייף אותה, אבל היא מאבדת את תקפותה אם שונתה אפילו סיבית אחת בהודעה. החתימה מאמתת אפוא הן את זהותו של החותם והן את תוכן ההודעה החתומה. זהו הכלי המושלם עבור חוזים עסקיים הנחתמים באופן אלקטרוני, המחאות אלקטרוניות, טופסי הזמנה לקניות אלקטרוניות ותקשורות אלקטרוניות אחרות שיש צורך באימותן.

מערכת ההצפנה באמצעות מפתח ציבורי RSA מבוססת על ההבדל הדרמטי בין הקלות שבה ניתן למצוא מספרים ראשוניים גדולים לבין הקושי לפרק לגורמים את המכפלה של שני ראשוניים גדולים. סעיף 33.8 מתאר שגרה יעילה למציאת מספרים ראשוניים גדולים, וסעיף 33.9 דן בבעיית הפירוק לגורמים של שלמים גדולים.

מערכות הצפנה המבוססות על מפתח ציבורי

במערכת הצפנה המבוססת על מפתח ציבורי (public-key cryptosystem), כל משתתף מקבל **מפתח ציבורי** (public key) וגם **מפתח סודי** (secret key). כל מפתח הוא פריט מידע. לדוגמה, במערכת ההצפנה RSA, כל מפתח מורכב מזוג שלמים. אנו נדגים את פעולתן של מערכות הצפנה באמצעות שני משתתפים: "איה" ו"בועז", ונסמן את המפתחות הציבוריים והסודיים שלהם על־ידי P_A ו־S_A עבור איה, ו־P_B ו־S_B עבור בועז.

כל משתתף יוצר בעצמו את המפתח הציבורי ואת המפתח הסודי שלו. כל אחד שומר בסוד את המפתח הסודי שלו, אבל יכול לגלות לכל אדם את המפתח הציבורי שלו ואף לפרסם אותו. למעשה, לעתים קרובות נוח להניח שניתן למצוא את המפתחות הציבוריים של כל המשתתפים במדריך ציבורי, כך שכל משתתף יכול לאתר בקלות את המפתח הציבורי של כל משתתף אחר.

המפתחות, הציבורי והסודי, מגדירים פונקציות שניתן להפעילן על כל הודעה. נסמן ב־$\mathcal{D}$ את קבוצת ההודעות המותרות. לדוגמה, $\mathcal{D}$ עשויה להיות קבוצת כל הסדרות באורך סופי של סיביות. אנו דורשים שהמפתח הציבורי והמפתח הסודי יגדירו פונקציות חד־חד־ערכיות מ־$\mathcal{D}$ לעצמה. הפונקציה המתאימה למפתח הציבורי P_A של איה מסומנת על־ידי $P_A()$, והפונקציה המתאימה

למפתח הסודי שלה S_A מסומנת על־ידי $S_A()$. הפונקציות $P_A()$ ו־$S_A()$ הן אפוא תמורות של $\mathcal{D}$. אנו מניחים שהפונקציות $P_A()$ ו־$S_A()$ ניתנות לחישוב ביעילות בהינתן המפתחות המתאימים P_A ו־S_A.

המפתח הציבורי והמפתח הסודי של כל משתתף הם "זוג תואם", במובן זה שהפונקציות שהם מגדירים הופכיות זו לזו. כלומר, עבור כל הודעה $M \in \mathcal{D}$:

$$M = S_A(P_A(M)) \tag{33.37}$$

$$M = P_A(S_A(M)) \tag{33.38}$$

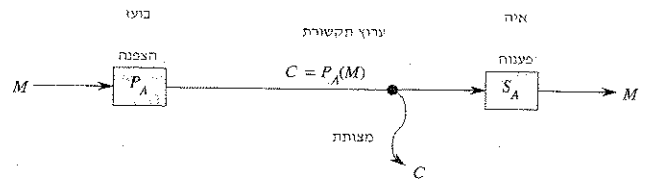
אם מפעילים על M את שני המפתחות P_A ו־S_A בזה אחר זה, בסדר כלשהו, מקבלים בחזרה את M.

במערכת הצפנה הפועלת באמצעות מפתח ציבורי, חיוני שאיש פרט לאיה לא יהיה מסוגל לחשב את הפונקציה $S_A()$ בזמן סביר. פרטיות הדואר המוצפן הנשלח לאיה ואימות החתימה הדיגיטלית של איה מסתמכים על ההנחה שרק איה מסוגלת לחשב את $S_A()$. זוהי הדרישה שבגללה איה שומרת בסוד את S_A; אם לא כן, היא תאבד את ייחודיותה ומערכת ההצפנה שוב לא תוכל לספק לה יכולות ייחודיות. ההנחה שרק איה מסוגלת לחשב את $S_A()$ חייבת להתקיים אף על פי שכולם מכירים את P_A ויכולים לחשב ביעילות את $P_A()$, הפונקציה ההופכית ל־S_A. הקושי העיקרי בתכנון מערכת הצפנה מבוססת מפתח ציבורי הפועלת היטב הוא למצוא כיצד ליצור מערכת שבה ניתן לחשוף טרנספורמציה $P_A()$ בלי לחשוף בכך כיצד לחשב את הטרנספורמציה ההופכית המתאימה $S_A()$.

במערכת הצפנה המבוססת על מפתח ציבורי, ההצפנה מתבצעת כמתואר להלן. נניח שבועז רוצה לשלוח לאיה הודעה M, מוצפנת כך שתיראה לכל מצותת כקשקוש בלתי קריא. משלוח ההודעה מתנהל על פי התרחיש הבא:

- בועז מקבל את המפתח הציבורי P_A של איה (ממדריך ציבורי או ישירות מאיה).
- בועז מחשב את **כתב־הצופן** (ciphertext) $C = P_A(M)$ של ההודעה M ושולח את C לאיה.
- כאשר איה מקבלת את כתב הצופן C, היא מפעילה עליו את המפתח הסודי שלה S_A ומקבלת את ההודעה המקורית: $M = S_A(C)$.

תהליך זה מודגם באיור 33.5. איה יכולה לחשב את M מתוך C מכיוון ש־$S_A()$ ו־$P_A()$ הן פונקציות הופכיות. מכיוון שרק איה יכולה לחשב את $S_A()$, רק היא יכולה לחשב את M מתוך C. הצפנת M באמצעות $P_A()$ מגנה על M מפני גילוי על־ידי כל אדם אחר פרט לאיה.

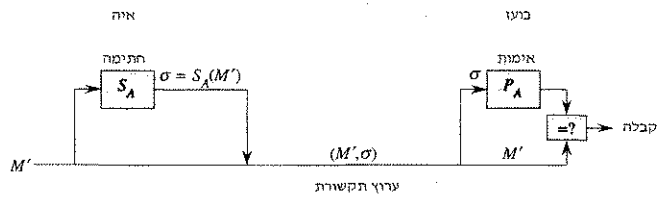


איור 33.5 הצפנה במערכת המבוססת על מפתח ציבורי. בועז מקודד את ההודעה M באמצעות המפתח הציבורי P_A של איה ושולח לאיה את כתב־הצופן המתקבל $C = P_A(M)$. מצותת הקולט את כתב־הצופן אינו מקבל ממנו שום מידע על M. איה מקבלת את C, מפענחת אותו באמצעות המפתח הסודי שלה ומקבלת את ההודעה המקורית $M = S_A(C)$.

באופן דומה ניתן לממש גם חתימות דיגיטליות במערכת הצפנה המבוססת על מפתח ציבורי. נניח עתה שאיה רוצה לשלוח לבועז תשובה M' חתומה באופן דיגיטלי. החתימה הדיגיטלית מתבצעת על פי התרחיש שלהלן:

- איה מחשבת את **החתימה הדיגיטלית** (digital signature) שלה σ עבור ההודעה M' באמצעות המפתח הסודי שלה S_A והמשוואה $\sigma = S_A(M')$.
- איה שולחת לבועז את הזוג (M', σ) של ההודעה והחתימה.
- כאשר בועז מקבל את (M', σ), הוא יכול לאמת שההודעה נשלחה על־ידי איה באמצעות אימות המשוואה $M' = P_A(\sigma)$ תוך שימוש במפתח הציבורי של איה. (אנו מניחים ש־M' מכילה את שמה של איה כך שבועז יודע באיזה מפתח ציבורי להשתמש.) אם המשוואה מתקיימת, בועז מסיק שההודעה M' אכן נחתמה על־ידי איה. אם המשוואה אינה מתקיימת, בועז מסיק שההודעה M' או החתימה נפגמו בשל שגיאות תמסורת או שהזוג (M', σ) הוא ניסיון זיוף.

תהליך זה מודגם באיור 33.6. מכיוון שחתימה דיגיטלית מאמתת הן את זהותו של החותם והן את תוכן ההודעה החתומה, היא מקבילה לחתימה בכתב יד בסופו של מסמך כתוב.



איור 33.6 חתימות דיגיטליות במערכת המבוססת על מפתח ציבורי. איה חותמת על ההודעה M' על־ידי כך שהיא מוסיפה לחודעה את החתימה הדיגיטלית שלה $\sigma = S_A(M')$. היא שולחת לבועז את הזוג (M', σ) של החודעה והחתימה, ובועז מאמת אותו על־ידי בדיקת המשוואה $M' = P_A(\sigma)$. אם המשוואה מתקיימת, הוא מקבל את (M', σ) כהודעה שנחותמה על־ידי איה.

תכונה חשובה של חתימה דיגיטלית היא היכולת לאמת אותה על־ידי כל מי שיש לו גישה למפתח הציבורי של החותם. אדם המקבל הודעה חתומה יכול לאמת אותה, ואז להעביר אותה הלאה למכותבים אחרים שגם הם יכולים לאמת את החתימה. לדוגמה, ההודעה עשויה להיות המחאה אלקטרונית הנשלחת מאיה לבועז. לאחר שבועז מאמת את חתימתה של איה על ההמחאה, הוא יכול להעביר את ההמחאה לבנק שלו, והבנק יכול גם הוא לאמת את החתימה ולבצע את העברת הכספים המתאימה.

נשים לב כי הודעה חתומה אינה מוצפנת; ההודעה גלויה לעין ואינה מוגנת מחשיפה. על־ידי הרכבה של פרוטוקולי ההצפנה והחתימה שתוארו לעיל, ניתן ליצור הודעות חתומות ומוצפנות כאחת. החותם מוסיף תחילה את חתימתו הדיגיטלית בסוף ההודעה, ואז מקודד את הזוג הודעה/חתימה המתקבל באמצעות המפתח הציבורי של הנמען המיועד. הנמען מפענח את ההודעה המתקבלת באמצעות המפתח הסודי שלו, ומקבל את ההודעה המקורית וגם את החתימה הדיגיטלית המצורפת אליה. לאחר מכן, הוא יכול לאמת את החתימה באמצעות המפתח הציבורי של החותם. תהליך משולב זה מקביל לחתימה בכתב יד על מסמך כתוב והכנסת המסמך למעטפה סגורה הנפתחת רק על־ידי הנמען המיועד.

מערכת ההצפנה RSA

במערכת ההצפנה RSA (RSA public-key cryptosystem), כל משתתף יוצר את המפתח הציבורי והמפתח הסודי שלו באמצעות האלגוריתם הבא:

1. בחר באקראי שני מספרים ראשוניים גדולים, p ו־q. הראשוניים p ו־q יהיו, נאמר, בני 100 ספרות עשרוניות כל אחד.
2. חשב את n באמצעות המשוואה $n = pq$.
3. בחר שלם אי־זוגי קטן e שהוא זר ל־$\phi(n)$. על פי משוואה (33.20), $\phi(n)$ שווה ל־$(p-1)(q-1)$.
4. חשב את d, ההופכי הכפלי של e, מודולו $\phi(n)$. (מסקנה 33.26 מבטיחה ש־d קיים ושהוא יחיד.)
5. פרסם את הזוג $P = (e, n)$ כ**מפתח RSA הציבורי** שלך (RSA public-key).
6. שמור בסוד את הזוג $S = (d, n)$ כ**מפתח RSA הסודי** שלך (RSA secret-key).

עבור מערכת זו, התחום $\mathcal{D}$ הוא הקבוצה $\mathbf{Z}_n$. הטרנספורמציה של הודעה M המתקבלת מהפעלת מפתח ציבורי $P = (e, n)$ היא:

$$P(M) = M^e \pmod{n} \tag{33.39}$$

הטרנספורמציה של כתב־צופן C באמצעות מפתח סודי $S = (d, n)$ היא:

$$S(C) = C^d \pmod{n} \tag{33.40}$$

משוואות אלה מתאימות להצפנה וגם לחתימה. ליצירת חתימה, החותם מפעיל את המפתח הסודי שלו על ההודעה שעליה הוא רוצה לחתום, ולא על כתב־צופן. כדי לאמת חתימה, מפעילים על החתימה, ולא על הודעה המיועדת להצפנה, את המפתח הציבורי של החותם.

פעולת המפתח הציבורי ופעולת המפתח הסודי ניתנות למימוש באמצעות השגרה MODULAR-EXPONENTIATION שתוארה בסעיף 33.6. כדי לנתח את זמן הריצה של פעולות אלה, נניח שהמפתח הציבורי (e, n) והמפתח הסודי (d, n) מקיימים $\lg_2 e = O(1)$ וכן $\lg_2 d = \lg_2 n = \beta$. אזי, הפעלת מפתח ציבורי דורשת $O(1)$ פעולות כפל מודולרי המשתמשות ב־$O(\beta^2)$ פעולות על סיביות. הפעלת מפתח סודי דורשת $O(\beta)$ פעולות כפל מודולרי המשתמשות ב־$O(\beta^3)$ פעולות על סיביות.

משפט 33.36 (נכונות RSA)
משוואות (33.39) ו־(33.40) במערכת RSA מגדירות טרנספורמציות הופכיות זו לזו ב־$\mathbf{Z}_n$ המקיימות את משוואות (33.37) ו־(33.38).

הוכחה ממשוואות (33.39) ו־(33.40) נקבל כי לכל $M \in \mathbf{Z}_n$:

$$P(S(M)) = S(P(M)) = M^{ed} \pmod{n}$$

מכיוון ש־e ו־d הם הופכיים כפליים מודולו $\phi(n) = (p-1)(q-1)$, אזי קיים שלם k שעבורו:

$$ed = 1 + k(p-1)(q-1)$$

אבל אז, אם $M \not\equiv 0 \pmod{p}$, נקבל (בעזרת משפט 33.31):

$$\begin{aligned} M^{ed} &\equiv M\,(M^{p-1})^{k(q-1)} \pmod p \\ &\equiv M\,(1)^{k(q-1)} \pmod p \\ &\equiv M \pmod p \end{aligned}$$

כמו כן, אם $M \equiv 0 \pmod p$, אזי $M^{ed} \equiv M \pmod p$. לכן, לכל M מקיימת:

$$M^{ed} \equiv M \pmod p$$

באופן דומה, לכל M מקיימת:

$$M^{ed} \equiv M \pmod q$$

לכן, על פי מסקנה 33.29 של משפט השאריות הסיני, לכל M מקיימת:

$$M^{ed} \equiv M \pmod n$$

■

בטיחותה של מערכת ההצפנה RSA תלויה במידה רבה בקושי לפרק לגורמים שלמים גדולים. אם יריב יכול לפרק לגורמים את המודולוס n במפתח ציבורי, אזי מידיעת הגורמים p ו־q הוא יכול לחשב את המפתח הסודי מן המפתח הציבורי באותו אופן שעשה זאת יוצר המפתח הציבורי. לכן, אם קל לפרק לגורמים שלמים גדולים, אזי קל גם לפצח את מערכת ההצפנה RSA. הטענה ההפוכה, שאם קשה לפרק לגורמים שלמים גדולים אזי קשה לפצח את RSA, לא הוכחה. עם זאת, לאחר עשור של מחקר, לא נמצאה דרך קלה יותר לפיצוח מערכת ההצפנה RSA המבוססת על מפתח ציבורי, מאשר פירוק לגורמים של המודולוס n. כפי שנראה בסעיף 33.9, פירוק לגורמים של שלמים גדולים קשה להפליא. על־ידי בחירה אקראית של שני ראשוניים בני 100 ספרות כל אחד והכפלתם זה בזה, ניתן ליצור מפתח ציבורי שאינו ניתן לפיצוח בתוך זמן סביר באמצעות הטכנולוגיה הקיימת כיום. ללא פריצת דרך משמעותית בתכנון אלגוריתמים בתורת המספרים, מערכת ההצפנה RSA מסוגלת לספק דרגת בטיחות גבוהה ביישומים.

אולם, כדי להשיג בטיחות במערכת ההצפנה RSA, יש לעבוד עם שלמים בני $100 - 200$ ספרות, שכן פירוק לגורמים של שלמים קטנים יותר אינו בלתי מעשי. ובמיוחד, עלינו להיות מסוגלים למצוא ביעילות ראשוניים גדולים, כדי ליצור מפתחות באורך הדרוש. בבעיה זו יעסוק סעיף 33.8.

מטעמי יעילות, משתמשים לעתים קרובות במערכת RSA יחד עם מערכות הצפנה מהירות שאינן משתמשות במפתח ציבורי, בשיטת "כלאיים" או "ניהול מפתחות". במערכת כזאת, מפתחות ההצפנה והפענוח זהים. אם איה רוצה לשלוח לבועז הודעה פרטית M, היא בוחרת באקראי מפתח K ומשתמשת בו כדי להצפין את M בשיטת ההצפנה המהירה, שאינה מבוססת על מפתח ציבורי, ולקבל את כתב־הצופן C. כאן, C שווה באורכו ל־M אבל K קצר למדי. לאחר מכן מצפינה איה את K באמצעות מפתח RSA הציבורי של בועז. מכיוון ש־K קצר, החישוב של $P_B(K)$ מהיר (מהיר בהרבה מהחישוב של $P_B(M)$). אחר כך איה שולחת לבועז את $(C, P_B(K))$, ובועז מפענח את $P_B(K)$ ומקבל את K, ואז משתמש ב־K לפיענח C ומקבל את M.

גישת כלאיים דומה משמשת לעתים קרובות לחתימות דיגיטליות יעילות. בגישה זו משלבים את RSA עם **פונקציית גיבוב חד־כיוונית** (one-way hash function) h – פונקציה ציבורית, קלה לחישוב, אך כזאת שעבורה אין אפשרות חישובית מעשית למצוא שתי הודעות M ו־M' המקיימות $h(M) = h(M')$. הערך $h(M)$ הוא "טביעת־אצבע" קצרה (נאמר, בת 128 סיביות) של ההודעה M. אם איה רוצה לחתום על הודעה M, היא מפעילה עליה תחילה את h, לקבלת טביעת־האצבע $h(M)$, ואז היא חותמת עליה באמצעות המפתח הסודי שלה. היא שולחת לבועז את

$(M, S_A(h(M)))$ כגרסה החתומה של M. בועז יכול לאמת את החתימה על־ידי כך שהוא מחשב את $h(M)$ ואז מפעיל את P_A על $S_A(h(M))$ ומוודא שהתוצאה המתקבלת היא אכן $h(M)$. מאחר שלא ניתן ליצור שתי הודעות עם אותה טביעת־אצבע, הרי שלא ניתן לשנות את ההודעה החתומה ועדיין לשמור על תקפות החתימה.

לבסוף, נשים לב ששימוש **באישורים** (certificates) מקל מאוד את הפצתם של מפתחות ציבוריים. לדוגמה, נניח שקיימת "רשות מוסמכת" T שהמפתח הציבורי שלה ידוע לכולם. איה יכולה לקבל מ־T הודעה חתומה (האישור שלה) הקובעת כי "המפתח הציבורי של איה הוא P_A". אישור זה ניתן לאימות בפני עצמו, שכן כל אחד מכיר את P_T. איה יכולה לכלול את האישור שלה בהודעות החתומות שהיא שולחת, כך שמקבל ההודעה יקבל גם את המפתח הציבורי שלה ויוכל לאמת מיד את חתימתה. מאחר שהמפתח שלה נחתם על־ידי T, מקבל ההודעה יודע שהמפתח של איה הוא אכן שלה.

תרגילים

33.7-1

נתון זוג מפתחות RSA עם $p = 11$, $q = 29$, $n = 319$ ו־$e = 3$. באיזה ערך של d יש להשתמש במפתח הסודי? מהו כתב־הצופן של ההודעה $M = 100$?

33.7-2

הוכח שאם מעריך החזקה הציבורי e של איה הוא 3, ויריב משיג את מעריך החזקה הסודי של איה, d, אזי היריב יכול לפרק לגורמים את המודולוס n של איה בזמן פולינומיאלי במספר הסיביות ב־n. (אמנם אינך מתבקש להוכיח זאת, אבל אולי תתעניין לדעת שתוצאה זו עדיין נכונה גם כאשר מסירים את התנאי $e = 3$. ראה Miller [147].)

* 33.7-3

הוכח ש־RSA היא מערכת "כפלית" במובן זה שהיא מקיימת:

$$P_A(M_1)P_A(M_2) \equiv P_A(M_1 M_2) \pmod n$$

השתמש בעובדה זו כדי להוכיח שאם בידי יריב מצויה שגרה המפענחת ביעילות אחוז אחד של הודעות שנבחרו באופן אקראי מ־$\mathbf{Z}_n^*$ והוצפנו עם P_A, אזי הוא יכול להפעיל אלגוריתם הסתברותי אשר בהסתברות גבוהה יפענח כל הודעה שהוצפנה עם P_A.

* 33.8 בדיקת ראשוניות

בסעיף זה נבחן את בעיית מציאתם של מספרים ראשוניים גדולים. נפתח בדיון בצפיפותם של הראשוניים, לאחר מכן נבחן גישה סבירה (אולם בלתי שלמה) לבדיקת ראשוניות, ולבסוף נציג את מבחן הראשוניות האקראי היעיל של מילר ורבין.

צפיפות המספרים הראשוניים

ביישומים רבים (כגון הצפנה) יש למצוא מספרים ראשוניים גדולים "אקראיים". למרבה המזל, מספרים ראשוניים גדולים אינם נדירים מדי, כך שלא נדרש זמן ארוך מדי לבדיקת שלמים אקראיים בגודל המתאים עד שמוצאים ביניהם מספר ראשוני. **פונקציית התפלגות הראשוניים** (prime distribution function) $\pi(n)$ מציינת את מספר הראשוניים הקטנים מ־n או שווים לו. לדוגמה, $\pi(10) = 4$, שכן קיימים 4 ראשוניים הקטנים מ־10 או שווים לו: 2, 3, 5 ו־7. משפט המספרים הראשוניים מציג קירוב שימושי ל־$\pi(n)$.

משפט 33.37 (משפט המספרים הראשוניים)

$$\lim_{n \to \infty} \frac{\pi(n)}{n/\ln n} = 1$$

■

הקירוב $n/\ln n$ נותן אומדנים מדויקים למדי של $\pi(n)$ אפילו עבור n־ים קטנים. לדוגמה, קיימת סטייה קטנה מ־6% עבור $n = 10^9$, שכן $\pi(n) = 50{,}847{,}534$ ואילו $n/\ln n = 48{,}254{,}942$. (לאיש תורת המספרים, 10^9 הוא מספר קטן.)

ניתן להשתמש במשפט המספרים הראשוניים כדי לאמוד כי ההסתברות ששלם n אשר נבחר באקראי יהיה ראשוני היא $1/\ln n$. לכן, כדי למצוא ראשוני באורך השווה לאורכו של n, יהיה עלינו לבחון בקירוב $\ln n$ שלמים הקרובים ל־n שנבחרו באקראי. לדוגמה, כדי למצוא ראשוני בן 100 ספרות עלולה להידרש בדיקה של כ־$\ln 10^{100} \approx 230$ שלמים בני 100 ספרות שנבחרו באקראי. (ניתן לצמצם למחצית את מספר הבדיקות אם בוחרים רק שלמים אי־זוגיים.)

ביתרתו של סעיף זה נדון בבעיה של בדיקת ראשוניות, דהיינו, כיצד קובעים אם שלם גדול אי־זוגי n הוא ראשוני. מטעמי נוחות, נסמן את הפירוק של n לגורמים ראשוניים באופן הבא:

$$n = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r} \tag{33.41}$$

כאשר $r \geq 1$ ו־$p_1, p_2, \ldots, p_r$ הם הגורמים הראשוניים של n. כמובן, n הוא ראשוני אם ורק אם $r = 1$ ו־$e_1 = 1$.

גישה פשוטה אחת לבעיה של בדיקת ראשוניות היא ביצוע **ניסיונות חילוק** (trial division): מנסים לחלק את n בכל אחד מהשלמים $2, 3, \ldots, \lfloor \sqrt{n} \rfloor$. (שוב, ניתן לדלג על שלמים זוגיים גדולים מ־2.) קל לראות כי n הוא ראשוני אם ורק אם אף אחד מהמחלקים הללו אינו מחלק את n. בהנחה שכל חילוק כזה מתבצע בזמן קבוע, הרי שזמן הריצה במקרה הגרוע הוא $\Theta(\sqrt{n})$, דהיינו, אקספוננציאלי באורכו של n. (נזכור כי אם n מקודד בשיטה הבינרית ואורכו β סיביות, אזי $\beta = \lceil \lg(n+1) \rceil$, ולכן $\sqrt{n} = \Theta(2^{\beta/2})$.) לפיכך, גישה זו של ניסיונות חילוק פועלת היטב רק אם n קטן מאוד או שבמקרה יש לו גורם ראשוני קטן. כאשר היא פועלת, לגישת ניסיונות החילוק יש היתרון שלא זו בלבד שהיא קובעת אם n ראשוני או פריק, אלא גם מוצאת את אחד הגורמים הראשוניים כאשר n פריק.

בסעיף זה אנו מעוניינים רק לבדוק אם מספר נתון n הוא ראשוני; אם n פריק, איננו מעוניינים למצוא את גורמיו הראשוניים. כפי שנראה בסעיף 33.9, מציאת הגורמים הראשוניים יקרה מבחינה חישובית. אולי מפתיע הדבר שקל הרבה יותר לקבוע אם מספר נתון הוא ראשוני או לא מאשר למצוא את הגורמים הראשוניים אם הוא אינו ראשוני.

בדיקת פסאודו־ראשוניות

נבחן עתה שיטה לבדיקת ראשוניות אשר "כמעט פועלת", ולמעשה, היא טובה דיה עבור יישומים מעשיים רבים. שיפור של שיטה זו המסלק את הפגם הקטן בה יוצג בהמשך. נסמן ב־$\mathbf{Z}_n^+$ את איברי $\mathbf{Z}_n$ השונים מאפס:

$$\mathbf{Z}_n^+ = \{1, 2, \ldots, n-1\}$$

אם n ראשוני, אזי $\mathbf{Z}_n^+ = \mathbf{Z}_n^*$.

אנו אומרים ש־n הוא **פסאודו־ראשוני בבסיס a** (base-a pseudoprime) אם n פריק ומקיים:

$$a^{n-1} \equiv 1 \pmod{n} \tag{33.42}$$

ממשפט פרמה (משפט 33.31) נובע שאם n ראשוני, אזי n מקיים את משוואה (33.42) עבור כל a השייך ל־$\mathbf{Z}_n^+$. לכן, אם נוכל למצוא $a \in \mathbf{Z}_n^+$ כלשהו שעבורו n *אינו* מקיים את משוואה (33.42), אזי n בוודאי פריק. למרבה ההפתעה, הטענה ההפוכה מתקיימת *כמעט* תמיד, כך שקריטריון זה מהווה מבחן ראשוניות כמעט מושלם. בודקים אם n מקיים את משוואה (33.42) עבור $a = 2$. אם לא, מכריזים על n כעל מספר פריק. אחרת, הפלט הוא ניחוש ש־n ראשוני (כאשר למעשה, כל שאנו יודעים הוא ש־n הוא אחד מהשניים: ראשוני או פסאודו־ראשוני בבסיס 2).

השגרה הבאה מתיימרת לבדוק באופן זה את ראשוניותו של n. היא משתמשת בשגרה MODULAR-EXPONENTIATION מסעיף 33.6. ההנחה היא שהקלט n הוא שלם גדול מ־2.

```
PSEUDOPRIME(n)
1  if MODULAR-EXPONENTIATION(2, n − 1, n) ≠ 1 (mod n)
2      then return COMPOSITE          ▷ Definitely
3      else return PRIME              ▷ We hope!
```

שגרה זו עלולה לשגות, אולם השגיאות הן מסוג אחד בלבד. כלומר, אם היא קובעת ש־n פריק, אזי היא תמיד צודקת. אבל אם היא קובעת ש־n ראשוני, אזי היא טועה רק אם n הוא פסאודו־ראשוני בבסיס 2.

באיזו תדירות שוגה השגרה? לעתים נדירות להפליא. עבור n קטן מ־10,000, קיימים רק 22 ערכים שעבורם הוא שוגה; ארבעת הראשונים בהם הם: 341, 561, 645 ו־1105. ניתן להראות שההסתברות שתכניות זו תשגה עבור מספר בן β סיביות שנבחר באקראי שואפת לאפס כאשר β שואף לאינסוף. על סמך אומדנים מדויקים יותר של מספר המספרים בגודל נתון שהם פסאודו־ראשוניים בבסיס 2 שמצא Pomerance [157], ניתן להעריך כי הסיכוי שמספר בן 50 ספרות שנבחר באקראי יוכרז כראשוני על־ידי השגרה שלעיל, כאשר למעשה הוא פסאודו־ראשוני בבסיס 2, הוא 1 למיליון; והסיכוי שמספר בן 100 ספרות שנבחר באקראי יוכרז כראשוני על־ידי השגרה למרות שהוא פסאודו־ראשוני בבסיס 2, הוא 1 ל־10^{13}.

לרוע המזל, איננו יכולים למנוע את כל הטעויות פשוט על־ידי בדיקת משוואה (33.42) עבור בסיס נוסף, נאמר $a = 3$, משום שקיימים שלמים פריקים n המקיימים את משוואה (33.42) עבור כל $a \in \mathbf{Z}_n^*$. שלמים אלה ידועים כ**מספרי קארמייקל** (Carmichael numbers). שלושת מספרי קארמייקל הראשונים הם 561, 1105 ו־1729. מספרי קארמייקל נדירים ביותר; לדוגמה, קיימים רק 255 מספרי קארמייקל הקטנים מ־100,000,000. תרגיל 33.8-2 עוזר להבין מדוע הם נדירים כל כך.

עתה נראה כיצד נוכל לשפר את מבחן הראשוניות שלנו כך שלא יוטעה על־ידי מספרי קארמייקל.

מבחן הראשוניות האקראי של מילר ורבין

מבחן הראשוניות של מילר ורבין (Miller-Rabin) מתגבר על הבעיות של המבחן הפשוט באמצעות שני שינויים: PSEUDOPRIME

- הוא מנסה כמה ערכי בסיס a שנבחרים באקראי במקום ערך בסיס אחד בלבד.
- בעת ההעלאה המודולרית בחזקה, הוא בודק אם התגלה שורש ריבועי לא טריוויאלי של 1, מודולו n. אם התגלה שורש כזה, המבחן עוצר ומפיק כפלט הודעה ש־n פריק. מסקנה 33.35 מצדיקה איתור מספרים פריקים באופן זה.

להלן הפסאודו־קוד עבור מבחן הראשוניות של מילר ורבין. הקלט $n > 2$ הוא המספר האי־זוגי שאת ראשוניותו יש לבדוק, והקלט s הוא מספר ערכי הבסיס הנבחרים באקראי מתוך $\mathbf{Z}_n^+$ כדי לבחון את n. הקוד משתמש במחולל המספרים האקראיים RANDOM מסעיף 8.3: RANDOM$(1, n-1)$ מחזיר שלם a שנבחר באקראי ומקיים $1 \le a \le n-1$. הקוד משתמש גם בשגרת עזר WITNESS. הקריאה WITNESS(a, n) מחזירה TRUE אם ורק אם a הוא "עד" (witness) לפריקותו של n — כלומר, אם ניתן להוכיח באמצעות a (בדרך שנראה להלן) ש־n פריק. המבחן WITNESS(a, n) דומה למבחן:

$$a^{n-1} \not\equiv 1 \pmod{n}$$

שעליו (עבור $a = 2$) התבססה השגרה PSEUDOPRIME, אך הוא יעיל יותר. תחילה נציג ונצדיק את המבנה של WITNESS, ולאחר מכן נראה כיצד משתמש בה מבחן הראשוניות של מילר ורבין.

```
WITNESS(a, n)
1  let ⟨b_k, b_{k-1}, ..., b_0⟩ be the binary representation of n − 1
2  d ← 1
3  for i ← k downto 0
4      do x ← d
5      d ← (d · d) mod n
6          if d = 1 and x ≠ 1 and x ≠ n − 1
7              then return TRUE
8          if b_i = 1
9              then d ← (d · a) mod n
10 if d ≠ 1
11     then return TRUE
12 return FALSE
```

הפסאודו־קוד עבור WITNESS מבוסס על הפסאודו־קוד של MODULAR-EXPONENTIATION. שורה 1 קובעת את הייצוג הבינרי של $n-1$, אשר ישמש להעלאת a בחזקת $(n-1)$. שורות 2-9 מחשבות את d כ־$a^{n-1} \bmod n$. שיטת החישוב זהה לשיטה שהשתמשנו בה בשגרה MODULAR-EXPONENTIATION. אבל בכל פעם שבשורה 5 מתבצעת העלאה בריבוע, שורות 6-7 בודקות אם התגלה זה עתה שורש ריבועי לא טריוויאלי של 1. אם התגלה שורש כזה, האלגוריתם עוצר ומחזיר TRUE. שורות 10-11 מחזירות TRUE אם הערך $a^{n-1} \bmod n$ שחושב שונה מ־1, בדיוק כשם ש־PSEUDOPRIME מחזירה COMPOSITE במקרה זה.

עתה אנו טוענים שאם WITNESS(a, n) מחזירה TRUE, אזי ניתן לבנות הוכחה לכך ש־n פריק, תוך שימוש ב־a.

אם WITNESS מחזירה TRUE משורה 11, אזי היא גילתה כי $d = a^{n-1} \bmod n \neq 1$. אבל אם n ראשוני, אזי על פי משפט פרמה (משפט 33.31), מתקיים $a^{n-1} \equiv 1 \pmod{n}$ עבור כל $a \in \mathbf{Z}_n^+$. לכן, n אינו יכול להיות ראשוני, והמשוואה $a^{n-1} \bmod n \neq 1$ היא הוכחה לכך.

אם WITNESS מחזירה TRUE משורה 7, אזי היא גילתה כי x הוא שורש ריבועי לא טריוויאלי של 1, מודולו n, שכן $x \not\equiv \pm 1 \pmod{n}$ אבל $x^2 \equiv 1 \pmod{n}$. מסקנה 33.35 קובעת ששורש ריבועי לא טריוויאלי של 1 מודולו n קיים רק אם n פריק, כך שקיומו של x כזה מהווה הוכחה לפריקותו של n.

בכך נשלמה הוכחת נכונותה של WITNESS. אם הקריאה WITNESS(a, n) מחזירה TRUE, אזי n בוודאי פריק, וניתן להוכיח בקלות את פריקותו תוך שימוש ב־a ו־n. עתה נתבונן במבחן הראשוניות של מילר ורבין, המבוסס על שימוש ב־WITNESS.

```
MILLER-RABIN(n, s)
1  for j ← 1 to s
2      do a ← RANDOM(1, n − 1)
3          if WITNESS(a, n)
4              then return COMPOSITE        ▷ Definitely.
5  return PRIME                             ▷ Almost surely.
```

השגרה MILLER-RABIN מהווה חיפוש הסתברותי אחר הוכחה לפריקותו של n. הלולאה הראשית (המתחילה בשורה 1) בוחרת s ערכים אקראיים של a מתוך $\mathbf{Z}_n^+$ (שורה 2). אם אחד מערכי a שנבחרו הוא עד לפריקותו של n, אזי MILLER-RABIN מחזירה COMPOSITE בשורה 4. מנכונותה של WITNESS נובע שפלט כזה הוא תמיד נכון. אם ב־s הניסיונות לא נמצא עד לפריקותו של n, MILLER-RABIN מניחה כי הסיבה לכך היא שלא קיימים עדים כאלה, ולכן n ראשוני. אנו נראה שאם s גדול דיו, אזי סביר שפלט זה נכון, אולם קיים סיכוי קטן שהשגרה היתה חסרת מזל בבחירת ערכי a, ועדים כאלה אכן קיימים, למרות שהשגרה לא מצאה אף אחד מהם.

להדגמת פעולתה של MILLER-RABIN, נבחר כ־n את מספר קארמייקל 561. נניח כי $a = 7$ נבחר כבסיס. איור 33.4 מראה כי WITNESS מגלה שורש ריבועי לא טריוויאלי של 1 בהעלאה האחרונה בריבוע, שכן $a^{280} \equiv 67 \pmod{n}$ ו־$a^{560} \equiv 1 \pmod{n}$. לכן, $a = 7$ הוא עד לפריקותו של n, WITNESS$(7, n)$ מחזירה TRUE, ו־MILLER-RABIN מחזירה COMPOSITE.

אם n הוא מספר בן β סיביות, ל־MILLER-RABIN נדרשות $O(s\beta)$ פעולות אריתמטיות ו־$O(s\beta^3)$ פעולות על סיביות, שכן אסימפטוטית, נדרשות לא יותר מאשר s העלאות מודולריות בחזקה.

שיעור השגיאות במבחן הראשוניות של מילר ורבין

אם MILLER-RABIN מחזירה PRIME, אזי קיים סיכוי קטן שזוהי שגיאה. אולם שלא כמו ב־PSEUDOPRIME, הסיכוי לשגיאה אינו תלוי ב־n; לא קיימים קלטים גרועים עבור שגרה זו. הסיכוי לשגיאה תלוי כאן בגודלו של s ובמזל הטוב בבחירת ערכי הבסיס a. כמו כן, מאחר שכל בדיקה שמבצעת השגרה היא מחמירה יותר מאשר בדיקה פשוטה של משוואה (33.42), הרי שעקרונית ניתן לצפות ששיעור השגיאה יהיה קטן עבור שלמים n הנבחרים באקראי. המשפט הבא מציג טיעון מדויק יותר.

משפט 33.38
אם n הוא מספר אי־זוגי פריק, אזי מספר העדים לפריקותו של n הוא לפחות $(n-1)/2$.

הוכחה ההוכחה מראה שמספר המספרים שאינם עדים לפריקותו של n אינו עולה על $(n-1)/2$, ומכאן נובעת טענת המשפט.

תחילה נשים לב שכל מספר שאינו עד לפריקותו של n שייך בהכרח ל־$\mathbf{Z}_n^*$, שכן כל מספר a שאינו עד מקיים $a^{n-1} \equiv 1 \pmod n$; אבל אם $\gcd(a, n) = d > 1$, אזי על פי מסקנה 33.21, לא קיימים פתרונות x למשוואה $ax \equiv 1 \pmod n$. (ובמיוחד, $x = a^{n-2}$ אינו פתרון.) לכן, כל איבר ב־$\mathbf{Z}_n^* - \mathbf{Z}_n$ הוא עד לפריקותו של n.

להשלמת ההוכחה נראה להלן שכל המספרים שאינם עדים מוכלים בתת־חבורה ממש B של $\mathbf{Z}_n^*$. על פי מסקנה 33.16, נקבל אז $|\mathbf{Z}_n^*|/2 \geq |B|$. מאחר ש־$|B| \leq n-1$, נקבל $|B| \leq (n-1)/2$. לכן, קיימים לכל היותר $(n-1)/2$ מספרים שאינם עדים לפריקותו של n, ומכאן שמספר העדים חייב להיות לפחות $(n-1)/2$.

עתה נראה כיצד מוצאים תת־חבורה ממש B של $\mathbf{Z}_n^*$ המכילה את כל המספרים שאינם עדים. נחלק את ההוכחה לשני מקרים:

מקרה 1: קיים $x \in \mathbf{Z}_n^*$ המקיים:

$$x^{n-1} \not\equiv 1 \pmod n \tag{33.43}$$

תהי $B = \{b \in \mathbf{Z}_n^* : b^{n-1} \equiv 1 \pmod n\}$. מכיוון ש־$B$ סגורה תחת כפל מודולו n, אזי, על פי משפט 33.14, B היא תת־חבורה של $\mathbf{Z}_n^*$. נשים לב שכל מספר שאינו עד שייך ל־B, שכן מספר a שאינו עד מקיים $a^{n-1} \equiv 1 \pmod n$. מכיוון ש־$x \in \mathbf{Z}_n^* - B$, מקבלים כי B היא תת־חבורה ממש של $\mathbf{Z}_n^*$.

מקרה 2: לכל $x \in \mathbf{Z}_n^*$ מתקיים:

$$x^{n-1} \equiv 1 \pmod n \tag{33.44}$$

במקרה זה, n אינו יכול להיות חזקה של מספר ראשוני. כדי לראות מדוע, יהי $n = p^e$, כאשר p הוא ראשוני אי־זוגי ו־$e > 1$. משפט 33.32 גורר כי $\mathbf{Z}_n^*$ מכילה איבר g שעבורו $\operatorname{ord}_n(g) = |\mathbf{Z}_n^*| = \phi(n) = (p-1)p^{e-1}$. אבל אז משוואה (33.44) ומשפט הלוגריתם הבדיד (משפט 33.33, עבור $y = 0$) גוררים כי $n - 1 \equiv 0 \pmod{\phi(n)}$, או:

$$(p-1)p^{e-1} \mid p^e - 1$$

תנאי זה אינו מתקיים עבור $e > 1$, משום שאז אגף שמאל מתחלק ב־p ואילו אגף ימין אינו מתחלק ב־p. לכן, n איננו חזקה של מספר ראשוני.

מאחר ש־n איננו חזקה של מספר ראשוני, נפרק אותו למכפלה $n_1 n_2$, כאשר n_1 ו־n_2 גדולים מ־1 וזרים זה לזה. (עשויות להיות דרכים אחדות לעשות זאת, ואין זה משנה איזו דרך נבחר. לדוגמה, אם $n = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$, אזי נוכל לבחור $n_1 = p_1^{e_1}$ ו־$n_2 = p_2^{e_2} p_3^{e_3} \cdots p_r^{e_r}$.)

נגדיר t ו־u שעבורם מתקיים $n - 1 = 2^t u$, כאשר $t \geq 1$ ו־u אי־זוגי. לכל $a \in \mathbf{Z}_n^+$, נתבונן בסדרה:

$$\langle a^u, a^{2u}, a^{2^2 u}, \ldots, a^{2^t u} \rangle \tag{33.45}$$

שבה כל האיברים מחושבים מודולו n. מכיוון ש־$2^t \mid (n-1)$, הייצוג הבינרי של $n-1$ מסתיים ב־t אפסים, ואיברי $\hat{a}$ הם $t+1$ ערכי d האחרונים המחושבים על־ידי WITNESS במהלך החישוב של $a^{n-1} \pmod n$; t הפעולות האחרונות הן העלאות בריבוע.

עתה נמצא $j \in \{0, 1, \ldots, t\}$ שעבורו קיים $v \in \mathbf{Z}_n^*$ המקיים $v^{2^j u} \equiv -1 \pmod n$; צריך להיות גדול ככל האפשר. j כזה בוודאי קיים, שכן u אי־זוגי: ניתן לבחור $v = -1$ ו־$j = 0$. נבחר v המקיים את התנאי שלעיל. נגדיר:

$$B = \{x \in \mathbf{Z}_n^* : x^{2^j u} \equiv \pm 1 \pmod n\}$$

מכיוון ש־B סגורה תחת כפל מודולו n, היא מהווה תת־חבורה של $\mathbf{Z}_n^*$. לכן, $|B|$ מחלק את $|\mathbf{Z}_n^*|$. כל מספר שאינו עד חייב להיות איבר של B, שכן הסדרה (33.45), הנוצרת על־ידי מספר שאינו עד, חייבת להיות מורכבת כולה מ־1־ים או להכיל -1 אחד לא יאוחר מן המיקום ה־j (על פי המקסימליות של j).

עתה נשתמש בקיומו של v כדי להראות כי קיים $w \in \mathbf{Z}_n^* - B$. מאחר שמתקיים $v^{2^j u} \equiv -1 \pmod n$, אזי על פי מסקנה 33.29, מתקיים גם $v^{2^j u} \equiv -1 \pmod{n_1}$. על פי מסקנה 33.28, קיים w המספק בו בזמן את שתי המשוואות:

$$w \equiv v \pmod{n_1}$$
$$w \equiv 1 \pmod{n_2}$$

לכן:

$$w^{2^j u} \equiv -1 \pmod{n_1}$$
$$w^{2^j u} \equiv 1 \pmod{n_2}$$

יחד עם מסקנה 33.29, משוואות אלה גוררות:

$$w^{2^j u} \not\equiv \pm 1 \pmod n \tag{33.46}$$

ולכן $w \notin B$. מכיוון ש־$v \in \mathbf{Z}_{n_1}^*$, מקבלים כי $v \in \mathbf{Z}_n^*$. לפיכך, $w \in \mathbf{Z}_n^*$, ולכן, $w \in \mathbf{Z}_n^* - B$. אנו מסיקים כי B היא תת־חבורה ממש של $\mathbf{Z}_n^*$.

בכל אחד מהמקרים אנו רואים שמספר העדים לפריקותו של n הוא לפחות $(n-1)/2$. ■

משפט 33.39
לכל שלם אי־זוגי $n > 2$ ושלם חיובי s, ההסתברות כי MILLER-RABIN(n, s) שוגה היא לכל היותר 2^{-s}.

הוכחה על פי משפט 33.38, אנו רואים שאם n פריק, אזי בכל הרצה של הלולאה שבשורות 4-1 קיימת הסתברות של $1/2$ לפחות לגלות עד x לפריקותו של n. MILLER-RABIN שוגה רק אם היתה כה חסרת מזל עד שנכשלה בגילוי עד לפריקותו של n בכל אחת מ־s האיטרציות של הלולאה הראשית. ההסתברות לסדרה כזאת של כשלונות היא לכל היותר 2^{-s}. ■

אם כן, בחירה של $s = 50$ אמורה להספיק כמעט לכל יישום שניתן להעלות על הדעת. אם מנסים למצוא מספרים ראשוניים גדולים על־ידי הפעלת MILLER-RABIN על שלמים גדולים *שנבחרו באקראי*, אזי ניתן להראות (אם כי לא נעשה זאת כאן) שקיימת סבירות קטנה מאוד שבחירה של s

קטן (נאמר $s = 3$) תוביל לתוצאות שגויות. דהיינו, עבור שלם אי־זוגי פריק n שנבחר באקראי, סביר שתוחלת מספר המספרים שאינם עדים לפריקותו של n תהיה קטנה בהרבה מ־$(n-1)/2$. אבל אם השלם n אינו נבחר באקראי, כל שניתן להוכיח הוא, שמספר המספרים שאינם עדים הוא לכל היותר $(n-1)/4$, וזאת באמצעות גרסה משופרת של משפט 33.38. יתר על כן, אכן קיימים שלמים n שעבורם מספר המספרים שאינם עדים הוא $(n-1)/4$.

תרגילים

33.8-1

הוכח שאם שלם אי־זוגי $n > 1$ אינו ראשוני או חזקה של ראשוני, אזי קיים שורש ריבועי לא טריוויאלי של 1 מודולו n.

* 33.8-2

ניתן לטעון טענה חזקה במעט ממשפט אוילר:
עבור כל $a \in \mathbf{Z}_n^*$,

$$a^{\lambda(n)} \equiv 1 \pmod{n}$$

כאשר $\lambda(n)$ מוגדר על־ידי:

$$\lambda(n) = \operatorname{lcm}(\phi(p_1^{e_1}), \ldots, \phi(p_r^{e_r})) \tag{33.47}$$

הוכח כי $\lambda(n) \mid \phi(n)$. מספר פריק n הוא מספר קרמייקל אם $\lambda(n) \mid (n-1)$. מספר קרמייקל הקטן ביותר הוא $561 = 3 \cdot 11 \cdot 17$; כאן, $\lambda(n) = \operatorname{lcm}(2, 10, 16) = 80$, המחלק את 560. הוכח כי מספרי קרמייקל חייבים לקיים את שתי התכונות הבאות: להיות "חופשיים מריבועים" (מספרים שאינם מתחלקים בריבוע של מספר ראשוני כלשהו), ולהיות מכפלה של שלושה ראשוניים לפחות. זוהי הסיבה לכך שאינם שכיחים כל כך.

33.8-3

הוכח שאם x הוא שורש ריבועי לא טריוויאלי של 1, מודולו n, אזי $\gcd(x-1, n)$ וכן $\gcd(x+1, n)$ הם שניהם מחלקים לא טריוויאליים של n.

* 33.9 פירוק לגורמים של מספרים שלמים

נניח שנתון לנו שלם n שברצוננו **לפרק לגורמים** (to factor), דהיינו, לפרק אותו למכפלה של מספרים ראשוניים. מבחן הראשוניות מהסעיף הקודם יגלה לנו ש־n פריק אולם לא יאמר לנו מהם הגורמים הראשוניים של n. כפי הנראה, קשה הרבה יותר לפרק לגורמים שלם גדול n מאשר לקבוע אם n ראשוני או פריק. גם באמצעות מחשבי־העל החדישים ביותר והאלגוריתמים הטובים ביותר המוכרים כיום אין זה אפשרי מבחינה מעשית לפרק לגורמים מספר שרירותי בן 200 ספרות עשרוניות.

היוריסטיקת רו של פולארד

שיטה בדוקה לקבלת פירוק מלא לגורמים של כל מספר עד B^2 היא לנסות לחלק את המספר הנתון בכל השלמים עד B. בהשקעה של אותה כמות עבודה, השגרה שלהלן תפיק פירוק לגורמים של כל מספר עד B^4 (אלא אם כן נהיה חסרי מזל במיוחד). מאחר שהשגרה אינה אלא היוריסטיקה, אין ערובה לזמן הריצה שלה או להצלחתה, אף כי בשימוש מעשי זוהי שגרה יעילה מאוד.

```
POLLARD-RHO(n)
1   i ← 1
2   x_1 ← RANDOM(0, n − 1)
3   y ← x_1
4   k ← 2
5   while TRUE
6       do i ← i + 1
7          x_i ← (x_{i−1}^2 − 1) mod n
8          d ← gcd(y − x_i, n)
9          if d ≠ 1 and d ≠ n
10            then print d
11         if i = k
12            then y ← x_i
13                 k ← 2k
```

השגרה פועלת כך: שורות 1-2 מאתחלות את i לערך 1 ואת x_1 לערך שנבחר באקראי מתוך $\mathbf{Z}_n$. לולאת ה־**while** שמתחילה בשורה 5 מבצעת אינסוף איטרציות בחיפושיה אחר גורמים של n. במהלך כל איטרציה של לולאת ה־**while**, משתמשים בשורה 7 בנוסחת הנסיגה:

$$x_i \leftarrow (x_{i-1}^2 - 1) \bmod n \tag{33.48}$$

להפקת הערך הבא x_i בסדרה האינסופית:

$$x_1, x_2, x_3, x_4, \ldots \tag{33.49}$$

ערכו של i גדל ב־1 בהתאם, בשורה 6. לשם בהירות נכתב הקוד תוך שימוש במשתנים x_i בעלי אינדקסים, אבל התכנית פועלת באותו אופן גם אם משמיטים את האינדקסים, משום שיש לשמור רק את הערך האחרון של x_i.

מפעם לפעם שומרת התכנית במשתנה y את הערך x_i האחרון שנוצר. ובמפורט, הערכים הנשמרים הם אלה שהאינדקס שלהם הוא חזקה של 2:

$$x_1, x_2, x_4, x_8, x_{16}, \ldots$$

שורה 3 שומרת את הערך x_1 ושורה 12 שומרת את x_k בכל פעם ש־i שווה ל־k. המשתנה k מאותחל לערך 2 בשורה 4, וערכו מוכפל פי 2 בשורה 13 בכל פעם שמעדכנים את y. לכן, k מקבל את סדרת הערכים $1, 2, 4, 8, \ldots$, וערכו הוא תמיד האינדקס של הערך x_k הבא שיש לשמור ב־y.

שורות 8-10 מנסות למצוא גורם של n בעזרת הערך השמור של y והערך x_i הנוכחי. ובמפורט, שורה 8 מחשבת את המחלק המשותף הגדול ביותר $d = \gcd(y - x_i, n)$. אם d הוא מחלק לא טריוויאלי של n (שורה 9 בודקת זאת), אזי שורה 10 מדפיסה את d.

שגרה זו למציאת גורם עשויה להיראות במבט ראשון מיסתורית משהו. אולם נשים לב כי POLLARD-RHO לעולם אינה מדפיסה תשובה שגויה; כל מספר שהיא מדפיסה הוא מחלק לא טריוויאלי של n. אלא ש־POLLARD-RHO עשויה שלא להדפיס דבר; אין שום ערובה לכך שהיא תפיק תוצאות כלשהן. אבל בהמשך נראה כי יש סיבות טובות לצפות ש־POLLARD-RHO תדפיס גורם p של n לאחר $\sqrt{p}$ איטרציות בקירוב של לולאת ה־**while**. לכן, אם n פריק, נוכל לצפות ששגרה זו תגלה מספיק מחלקים עבור פירוק מלא לגורמים של n לאחר $n^{1/4}$ עדכונים בקירוב, שכן כל גורם ראשוני p של n, פרט אולי לגדול ביותר, קטן מ־$\sqrt{n}$.

נבחן את התנהגותה של שגרה זו על־ידי כך שנחקור כמה זמן נדרש לסדרה אקראית מודולו n לחזור על ערך. מאחר ש־$\mathbf{Z}_n$ סופית, ומאחר שכל ערך בסדרה (33.49) תלוי אך ורק בערך הקודם לו, הרי שבסופו של דבר הסדרה (33.49) חוזרת על עצמה. ברגע שמגיעים ל־x_i המקיים $x_i = x_j$ עבור איזשהו $j < i$, פירוש הדבר שנסגר מעגל ומתחיל מחזור נוסף, שכן $x_{i+1} = x_{j+1}$, $x_{i+2} = x_{j+2}$ וכן הלאה. הסיבה לשם "היוריסטיקת רו (ρ)" היא, כפי שמראה איור 33.7, שניתן לצייר את הסדרה $x_1, x_2, \ldots, x_{j-1}$ כ"זנב" הרו ואת המעגל $x_j, x_{j+1}, \ldots, x_i$ כ"גוף" הרו.

הבה נדון בשאלה כמה זמן נדרש לסדרת ה־x_i להתחיל לחזור על עצמה. זה אינו בדיוק מה שאנו צריכים, אולם לאחר מכן נראה כיצד לשנות את הטיעון.

לצורך הערכה זו, נניח שהפונקציה $(x^2 - 1) \bmod n$ מתנהגת כמו פונקציה "אקראית". כמובן, הפונקציה אינה אקראית ממש, אבל הנחה זו מניבה תוצאות המתיישבות עם ההתנהגות הנצפית של POLLARD-RHO. כך נוכל להתייחס לכל x_i כאילו נשלף באופן בלתי תלוי מ־$\mathbf{Z}_n$, על פי התפלגות אחידה של איברי $\mathbf{Z}_n$. לפי ניתוח פרדוקס ימי־ההולדת מסעיף 6.6.1, תוחלת מספר הצעדים המתבצעים לפני שהסדרה מתחילה לחזור על עצמה היא $\Theta(\sqrt{n})$.

ועתה נעבור לשינוי הנדרש. יהי p גורם לא טריוויאלי של n המקיים $\gcd(p, n/p) = 1$. לדוגמה, אם $n = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$ הוא פירוק לגורמים של n, אזי ניתן, למשל, לבחור כ־p את $p_1^{e_1}$. (אם $e_1 = 1$, אזי p הוא פשוט הגורם הראשוני הקטן ביותר של n, דוגמה שכדאי לזכור אותה.) הסדרה $\langle x_i \rangle$ משרה סדרה מתאימה $\langle x'_i \rangle$ מודולו p, כאשר לכל i:

$$x'_i = x_i \bmod p$$

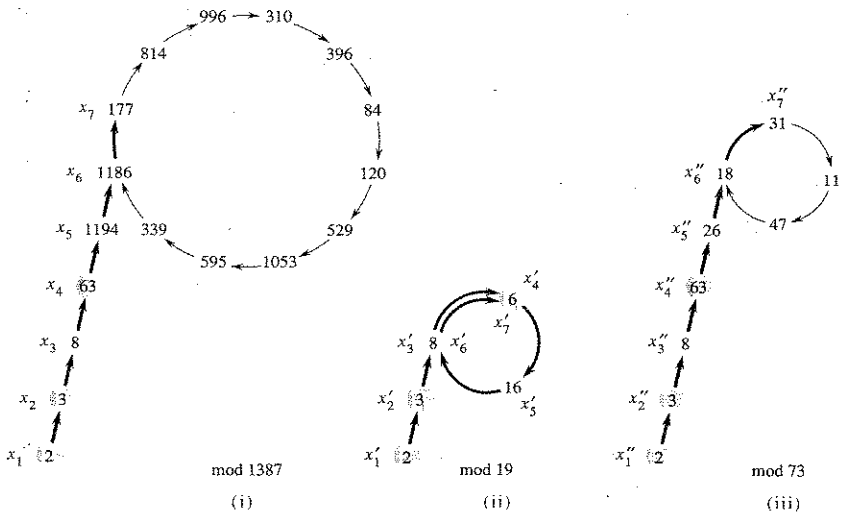
יתירה מזו, ממשפט השאריות הסיני נובע כי:

$$x'_{i+1} = (x'^2_i - 1) \bmod p \tag{33.50}$$

שכן על פי תרגיל 33.1-6:

$$(x \bmod n) \bmod p = x \bmod p$$

באמצעות טיעון כמו זה שלעיל, אנו מקבלים כי תוחלת מספר הצעדים לפני שהסדרה $\langle x'_i \rangle$ מתחילה לחזור על עצמה היא $\Theta(\sqrt{p})$. אם p קטן בהשוואה ל־n, הסדרה $\langle x'_i \rangle$ עשויה לחזור על עצמה מהר הרבה יותר מאשר הסדרה $\langle x_i \rangle$, ואכן, הסדרה $\langle x'_i \rangle$ חוזרת על עצמה ברגע ששני איברים של הסדרה $\langle x_i \rangle$ שקולים זה לזה מודולו p בלבד, ולא מודולו n. להדגמה, ראה חלקים (ii) ו־(iii) באיור 33.7.



איור 33.7 היוריסטיקת "רו" של פולארד. **(i)** הערכים שיוצרת נוסחת הנסיגה $x_{i+1} \leftarrow (x_i^2 - 1) \bmod 1387$ החל מ־$x_1 = 2$ הפירוק של 1387 לגורמים ראשוניים הוא 19·73. החיצים העבים מציינים את האיטרציות המתבצעות לפני שמתגלה הגורם 19. החיצים הדקים מצביעים על ערכים שהאיטרציה לא הגיעה אליהם והם מסומנים באיור כדי להמחיש את צורת ה"רו". הערכים המוצללים הם ערכי y שנשמרו על־ידי השגרה POLLARD-RHO. הגורם 19 מתגלה לאחר שמגיעים ל־$x_7 = 177$, כאשר מחשבים את $\gcd(63 - 177, 1387) = 19$. הערך x הראשון שהיה חוזר על עצמו הוא 1186, אבל הגורם 19 מתגלה לפני שמגיעים לערך זה. **(ii)** הערכים שיוצרת אותה נוסחת נסיגה, מודולו 19. כל ערך x_i המופיע בחלק (i) שקול, מודולו 19, לערך x'_i המופיע בחלק זה. לדוגמה, $x_4 = 63$ וגם $x_7 = 177$ שקולים ל־6, מודולו 19. **(iii)** הערכים שיוצרת אותה נוסחת נסיגה, מודולו 73. כל ערך x_i המופיע בחלק (i) שקול, מודולו 73, לערך x''_i המופיע בחלק זה. על פי משפט השאריות הסיני, כל צומת בחלק (i) מתאים לזוג צמתים, אחד מחלק (ii) ואחד מחלק (iii).

נסמן ב־t את האינדקס של הערך הראשון החוזר על עצמו בסדרה $\langle x'_i \rangle$, ויהי $u > 0$ אורכו של המעגל שנוצר. כלומר, t ו־$u > 0$ הם הערכים הקטנים ביותר המקיימים $x'_{t+i} = x'_{t+u+i}$ לכל $i \geq 0$. על פי הטיעונים שלעיל, התוחלת של t והתוחלת של u שתיהן $\Theta(\sqrt{p})$. נשים לב שאם $x'_{t+i} = x'_{t+u+i}$, אזי $p \mid (x_{t+u+i} - x_{t+i})$. לכן, $\gcd(x_{t+u+i} - x_{t+i}, n) > 1$.

לכן, ברגע ש־POLLARD-RHO שמרה ב־y ערך x_k כלשהו כך ש־$k \geq t$, אזי $y \bmod p$ נמצא תמיד על המעגל מודולו p. (אם נשמר ב־y ערך חדש, אזי גם ערך זה נמצא על המעגל מודולו p.) לבסוף, k מקבל ערך גדול מ־u, ואז השגרה עוברת על המעגל מודולו p כולו בלי לשנות את ערכו של y. גורם של n מתגלה אז כאשר x_i "נתקל" בערך הקודם שנשמר ב־y, מודולו p, כלומר, כאשר $x_i \equiv y \pmod{p}$.

אפשר להניח כי הגורם המתגלה באופן זה הוא הגורם p, אף כי לעתים מתגלה כפולה של p, מאחר שהתוחלות של t ושל u הן $\Theta(\sqrt{p})$, תוחלת מספר הצעדים הנדרשים למציאת הגורם p היא $\Theta(\sqrt{p})$.

קיימות שתי סיבות לכך שהאלגוריתם עלול שלא לפעול כצפוי. ראשית, הניתוח ההיוריסטי של זמן הריצה אינו מדויק, וייתכן שמעגל הערכים, מודולו p, יהיה גדול בהרבה מ־$\sqrt{p}$. במקרה זה, האלגוריתם יפעל נכונה אבל יהיה איטי בהרבה מהרצוי. הלכה למעשה, בעיה זו אינה נראית משמעותית. שנית, ייתכן שהמחלקים של n שמוצא אלגוריתם זה יהיו תמיד המחלקים הטריוויאליים 1 או n. לדוגמה, נניח כי $n = pq$, כאשר p ו־q ראשוניים. ייתכן שערכי t ו־u עבור p זהים לערכי t ו־u עבור q, ואז הגורם p מתגלה תמיד באותה פעולת gcd המגלה את הגורם q. מכיוון ששני הגורמים מתגלים באותו זמן, מתגלה הגורם הטריוויאלי $pq = n$ שאינו נותן לנו דבר. אך שוב, בעיה זו אינה בעיה ממשית ביישומים מעשיים. במקרה הצורך ניתן להתחיל מחדש את התהליך ההיוריסטי עם נוסחת נסיגה שונה מהצורה $x_{i+1} \leftarrow (x_i^2 - c) \bmod n$ (מסיבות שלא נפרט כאן, יש להימנע מן הערכים $c = 0$ ו־$c = 2$, אולם ניתן להשתמש בכל ערך אחר.)

כמובן, ניתוח זה הוא היוריסטי ואינו מדויק דיו, שכן נוסחת הנסיגה אינה "אקראית" באמת. אף על פי כן, הביצועים בפועל של השגרה טובים, ונראה שיעילותה אינה נופלת מהמצופה על סמך הניתוח ההיוריסטי. זוהי השיטה המומלצת למציאת גורמים ראשוניים קטנים של מספרים גדולים. כדי למצוא פירוק מלא לגורמים של מספר פריק n בן β סיביות, עלינו למצוא רק את כל הגורמים הראשוניים הקטנים מ־$\lfloor n^{1/2} \rfloor$ אנו מצפים אפוא שלשגרה POLLARD-RHO יידרשו לשם כך לכל היותר $n^{1/4} = 2^{\beta/4}$ פעולות אריתמטיות ולכל היותר $n^{1/4}\beta^3 = 2^{\beta/4}\beta^3$ פעולות על סיביות. העובדה שתוחלת מספר הפעולות האריתמטיות שמבצעת POLLARD-RHO כדי למצוא גורם ראשוני קטן p של n היא $\Theta(\sqrt{p})$, היא לעתים קרובות התכונה המושכת ביותר שלה.

תרגילים

33.9-1

במהלך ההרצה של POLLARD-RHO המוצגת באיור 33.7(i), מתי מדפיסה השגרה את הגורם 73 של 1387?

33.9-2

נניח שנתונים פונקציה $f : \mathbf{Z}_n \to \mathbf{Z}_n$ וערך התחלתי $x_0 \in \mathbf{Z}_n$. נגדיר $x_i = f(x_{i-1})$ עבור $i = 1, 2, \ldots$, ויהיו t ו־$u > 0$ הערכים הקטנים ביותר המקיימים $x_{t+i} = x_{t+u+i}$ עבור $i = 0, 1, \ldots$. במינוח של אלגוריתם "רו" של פולארד, t הוא אורכו של זנב הרו ו־u הוא אורכה של לולאת הרו. מצא אלגוריתם יעיל הקובע במדויק את t ו־u, ונתח את זמן הריצה שלו.

33.9-3

מהי לדעתך תוחלת מספר הצעדים שיידרשו ל־POLLARD-RHO כדי למצוא גורם מהצורה p^e כאשר p ראשוני ו־$e > 1$?

33.9-4 *

אחד החסרונות של POLLARD-RHO כפי שתוארה כאן הוא, שדרוש לה חישוב gcd אחד עבור כל צעד של נוסחת הנסיגה. ישנה הצעה "לקבץ" יחדיו חישובי gcd על־ידי חישוב מצטבר של מכפלתם של כמה x_i עוקבים ואז מציאת ה־gcd של מכפלה זו ושל הערך y שנשמר. תאר במדויק כיצד היית מממש רעיון זה ומדוע הוא פועל, ואיזה גודל של "מקבץ" היית בוחר כיעיל ביותר עבור מספר n בן β סיביות.

בעיות

33-1 אלגוריתם gcd בינרי

על רוב המחשבים, הפעולות של חיסור, בדיקת הזוגיות של מספר בינרי וחלוקה בשתיים, מתבצעות במהירות רבה יותר מאשר חישוב שאריות. בעיה זו חוקרת **אלגוריתם בינרי לחישוב מ"מ** (binary gcd algorithm) אשר נמנע מחישובי השאריות המשמשים באלגוריתם של אוקלידס.

א. הוכח שאם a ו־b שניהם זוגיים, אזי $\gcd(a, b) = 2\gcd(a/2, b/2)$.

ב. הוכח שאם a אי־זוגי ו־b זוגי, אזי $\gcd(a, b) = \gcd(a, b/2)$.

ג. הוכח שאם a ו־b שניהם אי־זוגיים, אזי $\gcd(a, b) = \gcd((a - b)/2, b)$.

ד. תכנן אלגוריתם בינרי יעיל שמקבל כקלט שני שלמים a ו־b, כאשר $a \geq b$, ומחשב את המ"מ שלהם בזמן $O(\lg(\max(a, b)))$. הנח שכל פעולה של חיסור, בדיקת זוגיות וחלוקה בשתיים, מתבצעת ביחידת זמן.

33-2 ניתוח הפעולות על סיביות באלגוריתם של אויקלידס

א. הראה שהאלגוריתם הרגיל לחילוק ארוך — חילוק a ב־b לקבלת מנה q ושארית r — דורש $O((1 + \lg q)\lg b)$ פעולות על סיביות.

ב. נגדיר $\mu(a, b) = (1 + \lg a)(1 + \lg b)$. הראה שמספר הפעולות על סיביות שמבצעת השגרה EUCLID כדי לצמצם את בעיית החישוב של $\gcd(a, b)$ לבעיית החישוב של $\gcd(b, a \bmod b)$, הוא לכל היותר $c(\mu(a, b) - \mu(b, a \bmod b))$, עבור איזשהו קבוע $c > 0$ גדול דיו.

ג. הראה שלשגרה $\text{EUCLID}(a, b)$ דרושות ככלל $O(\mu(a, b))$ פעולות על סיביות, ו־$O(\beta^2)$ פעולות על סיביות כאשר היא מופעלת על שני קלטים בני β סיביות.

33-3 שלושה אלגוריתמים עבור מספרי פיבונצ'י

בעיה זו משווה את יעילותן של שלוש שיטות לחישוב F_n, מספר פיבונצ'י ה־n, בהינתן n. נניח שהעלות של חיבור, חיסור או כפל של שני מספרים היא $O(1)$, ללא תלות בגודל המספרים.

א. הראה שזמן הריצה של השיטה הרקורסיבית הישירה לחישוב F_n על פי נוסחת הנסיגה (2.13) הוא אקספוננציאלי ב־n.

ב. הראה כיצד ניתן לחשב את F_n בזמן $O(n)$ באמצעות תזכור.

ג. הראה כיצד ניתן לחשב את F_n בזמן $O(\lg n)$ באמצעות פעולות חיבור וכפל של שלמים בלבד. (**רמז:** התבונן במטריצה הבאה:

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

ובחזקותיה.)

ד. עתה נניח שחיבור שני מספרים בני β סיביות מתבצע בזמן $\Theta(\beta)$ וכפל שני מספרים בני β סיביות מתבצע בזמן $\Theta(\beta^2)$. בהינתן עלויות סבירות יותר אלה של ביצועי הפעולות האריתמטיות הבסיסיות, מהו זמן הריצה של כל אחת משלוש השיטות?

33-4 שאריות ריבועיות

יהי p מספר ראשוני אי־זוגי. מספר $a \in \mathbf{Z}_p^*$ נקרא **שארית ריבועית** (quadratic residue) אם למשוואה $x^2 = a \pmod p$ יש פתרון עבור הנעלם x.

א. הראה שקיימות בדיוק $(p-1)/2$ שאריות ריבועיות מודולו p.

ב. אם p ראשוני, אנו מגדירים את **סימן לז'אנדר** (Legendre symbol) (a/p) עבור $a \in \mathbf{Z}_p^*$ כ־1 אם a הוא שארית ריבועית מודולו p וכ־1– אם a אינו שארית ריבועית מודולו p. הוכח שאם $a \in \mathbf{Z}_p^*$, אזי:

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod{p}$$

מצא אלגוריתם יעיל שיקבל כקלט מספר a ויקבע אם מספר זה הוא שארית ריבועית מודולו p או לא. נתח את יעילות האלגוריתם שלך.

ג. הוכח שאם p ראשוני מהצורה $4k+3$ ואם a הוא שארית ריבועית ב־$\mathbf{Z}_p^*$, אזי $a^{k+1} \bmod p$ הוא שורש ריבועי של a, מודולו p. כמה זמן נדרש למציאת השורש הריבועי של שארית ריבועית a מודולו p?

ד. תאר אלגוריתם אקראי יעיל למציאת שארית לא ריבועית, מודולו ראשוני שרירותי p. כמה פעולות אריתמטיות מבצע האלגוריתם שלך בממוצע?

הערות ביבליוגרפיות לפרק 33

אצל Niven & Zuckerman [151] מופיע מבוא מצוין ליסודות תורת המספרים. Knuth [122] מכיל דיון טוב באלגוריתמים למציאת מחלק משותף מקסימלי וכן באלגוריתמים בסיסיים אחרים בתורת המספרים. Riesel [168] ו־Bach [16] מספקים סקירות עדכניות יותר של תורת המספרים החישובית. Dixon [56] סוקר פירוק לגורמים ומבחני ראשוניות. ספר הכנס בעריכתו של Pomerance [159] מכיל כמה מאמרי סקירה נאים.

Knuth [122] דן במקורו של האלגוריתם של אויקלידס. האלגוריתם מופיע בטענות 1 ו־2 בספר 7 של *יסודות* (*Elements*) מאת המתמטיקאי היווני אויקלידס, שנכתב בשנת 300 לפנה"ס לערך. ייתכן שתיאורו של אויקלידס לקוח מאלגוריתם שפיתח אודוקסוס (Eudoxus) בסביבות שנת 375 לפנה"ס. האלגוריתם של אויקלידס מחזיק אולי בתואר האלגוריתם הלא טריוויאלי העתיק ביותר; מתחרהו היחיד הוא אלגוריתם האיכר הרוסי לכפל שני מספרים (ראה פרק 29), שהיה ידוע כבר במצרים העתיקה.

קנותי (Knuth) מייחס מקרה פרטי של משפט השאריות הסיני (משפט 33.27) למתמטיקאי הסיני סון־טסו (Sun-Tsu), שחי מתי שהוא בין שנת 200 לפנה"ס לשנת 200 לספירה — התאריכים המדויקים אינם ידועים. אותו מקרה פרטי הוצג על־ידי המתמטיקאי היווני ניקומאכוס (Nichomachus) בסביבות שנת 100 לספירה. מקרה פרטי זה הוכלל על ידי צ'ין צ'יו־שאו (Chhin Chiu-Shao) בשנת 1247. משפט השאריות הסיני נוסח והוכח בצורתו הכללית ביותר על־ידי אוילר (L. Euler) בשנת 1734.

האלגוריתם האקראי לבדיקת ראשוניות שהוצג בפרק זה הוא של Miller [147] ו־Rabin [166]; זהו האלגוריתם האקראי המהיר ביותר המוכר כיום לבדיקת ראשוניות, עד כדי גורמים קבועים. הוכחת משפט 33.39 הוא שינוי קל של הוכחה שהוצעה על־ידי Bach [15]. הוכחה של תוצאה חזקה יותר עבור Miller-Rabin ניתנה על ידי Monier [148, 149]. נראה שכדי להשיג זמן ריצה פולינומיאלי עבור אלגוריתם לבדיקת ראשוניות, אין מנוס מאקראיות. האלגוריתם הדטרמיניסטי המהיר ביותר המוכר כיום לבדיקת ראשוניות הוא גרסת Cohen-Lenstra [45] של מבחן

הראשוניות של Adelman, Pomerance & Rumely [3]. זמן הריצה של אלגוריתם זה על מספר n באורך $\lceil \lg(n+1) \rceil$ הוא $(\lg n)^{O(\lg\lg\lg n)}$, זמן שהוא על־פולינומיאלי אך במקצת.

בעיית מציאתם של מספרים ראשוניים "אקראיים" גדולים נדונה בצורה נאה במאמר מאת Beauchemin, Brassard, Crépeau, Goutier & Pomerance [20].

הרעיון של מערכת הצפנה המבוססת על מפתח ציבורי הוא של Diffie & Hellman [54]. מערכת ההצפנה RSA הוצעה ב־1977 על־ידי Rivest, Shamir & Adelman [169]. מאז פרח תחום ההצפנה. במיוחד, פותחו שיטות חדשות רבות להוכחת בטיחותן של מערכות הצפנה. לדוגמה, Goldwasser & Micali [86] מראים שאקראיות יכולה להיות כלי יעיל בתכנון של שיטות בטוחות להצפנה באמצעות מפתח ציבורי. בנושא החתימות הדיגיטליות, Goldwasser, Micali & Rivest [88] מציגים שיטה לחתימה דיגיטלית שעבורה מוכח כי כל סוג של זיוף שניתן להעלות על הדעת קשה כמו פירוק לגורמים. לאחרונה הציגו Goldwasser, Micali & Rackoff [87] מחלקה של שיטות הצפנה המבוססות על "אפס־ידע" (zero-knowledge) שעבורן ניתן להוכיח (בכפוף להנחות סבירות מסוימות) כי אף אחד מהצדדים המתקשרים אינו לומד מההתקשרות יותר ממה שהוא אמור ללמוד.

היוריסטיקת "רו" לפירוק שלמים לגורמים הומצאה על־ידי Pollard [156]. הגרסה המוצגת כאן הוצעה על־ידי Brent [35].

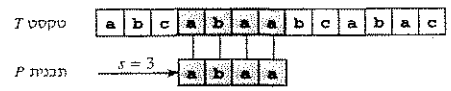
זמן הריצה של האלגוריתמים הטובים ביותר לפירוק של מספרים גדולים לגורמים הוא פונקציה הגדלה (בקירוב) באופן אקספוננציאלי בשורש הריבועי של אורך המספר n שיש לפרק לגורמים. אלגוריתם הנפה־הריבועית (quadratic-sieve) לפירוק לגורמים של Pomerance [158] הוא אולי האלגוריתם הכללי היעיל ביותר מסוג זה עבור קלטים גדולים. אף כי קשה לנתח במדויק את האלגוריתם הזה, הרי שבהנחות סבירות ניתן לקבל עבור זמן הריצה אומדן של $L(n)^{1+o(1)}$, כאשר $L(n) = e^{\sqrt{\ln n \ln\ln n}}$. שיטת העקום האליפטי (elliptic-curve) של Lenstra [137] אולי יעילה יותר משיטת הנפה הריבועית עבור קלטים מסוימים, שכן בדומה לשיטת "רו" של פולארד, היא יכולה למצוא גורם ראשוני קטן p במהירות גדולה למדי. בשיטה זו, הזמן למציאת p נאמד ב־$L(p)^{\sqrt{2}+o(1)}$.

34 התאמת מחרוזות תווים

מציאת כל המופעים של תבנית בטקסט היא בעיה החוזרת וצצה תדיר בתכניות עיבוד תמלילים. בדרך כלל, הטקסט הוא מסמך שיש לערוך, והתבנית היא מילה מסוימת שהמשתמש מחפש. אלגוריתמים יעילים לפתרון בעיה זו יכולים לשפר במידה ניכרת את מהירות התגובה של מעבד התמלילים. אלגוריתמים להתאמת מחרוזות משמשים גם, למשל, לחיפוש אחר תבניות מסוימות ברצפים של DNA.

להלן הניסוח הפורמלי של **בעיית התאמת־המחרוזות** (string-matching problem). מניחים כי הטקסט הוא מערך $T[1..n]$ באורך n והתבנית היא מערך $P[1..m]$ באורך $m \le n$. עוד מניחים שאיברי P ו־T הם תווים הלקוחים מאלפבית סופי Σ. לדוגמה, האלפבית עשוי להיות $\Sigma = \{0, 1\}$ או $\Sigma = \{a, b, \ldots, z\}$. מערכי התווים P ו־T נקראים לעתים קרובות **מחרוזות** (strings) של תווים.

נאמר שהתבנית P **מופיעה עם היסט** s (occurs with shift s) בטקסט T, או, באופן שקול, שההתבנית P **מופיעה החל מהמיקום** $s+1$ (P occurs beginning at position $s+1$) בטקסט T אם $0 \le s \le n-m$ ו־$T[s+1..s+m] = P[1..m]$ (כלומר, אם $T[s+j] = P[j]$ עבור $1 \le j \le m$). אם P מופיעה ב־T עם היסט s, נאמר ש־s הוא **היסט תקף** (valid shift); אחרת, נאמר ש־s הוא **היסט בלתי־תקף** (invalid shift). בעיית התאמת המחרוזות היא הבעיה של מציאת כל ההיסטים התקפים שבהם מופיעה תבנית נתונה P בטקסט נתון T. איור 34.1 מדגים הגדרות אלה.



איור 34.1 בעיית התאמת־המחרוזות. המטרה היא למצוא את המופעים של התבנית $P =$ abaa בטקסט $T =$ abcabaabcabac. התבנית מופיעה בטקסט פעם אחת בלבד, עם היסט של $s = 3$. אנו אומרים שההיסט $s = 3$ הוא היסט תקף. באיור, כל תו בתבנית מחובר בקו אנכי לתו המתאים לו בטקסט, וכל התווים התואמים מוצללים.

פרק זה מאורגן באופן הבא: בסעיף 34.1 נסקור את האלגוריתם הנאיבי להתאמת־מחרוזות, שרץ במקרה הגרוע בזמן $O((n-m+1)m)$. בסעיף 34.2 נציג אלגוריתם מעניין להתאמת־מחרוזות של רבין וקארפ (Rabin & Karp). גם אלגוריתם זה רץ במקרה הגרוע בזמן $O((n-m+1)m)$, אולם ביצועיו טובים בהרבה בממוצע וביישומים מעשיים. כמו כן, ניתן להכליל אותו בנקל לבעיות אחרות של התאמת־מחרוזות. לאחר מכן, בסעיף 34.3, נתאר אלגוריתם להתאמת־מחרוזות המתחיל בכך שהוא בונה אוטומט סופי המיועד במיוחד לחיפוש המופעים של תבנית נתונה P

בטקסט. אלגוריתם זה רץ בזמן $O(n + m|\Sigma|)$. אלגוריתם דומה, אך חכם בהרבה, הוא אלגוריתם קנות־מוריס־פראט (Knuth-Morris-Pratt או KMP), שיוצג בסעיף 34.4; אלגוריתם KMP רץ בזמן $O(n+m)$. לבסוף, בסעיף 34.5, נתאר אלגוריתם של בויאר ומור (Boyer & Moore) שהוא לעתים קרובות הבחירה המעשית הטובה ביותר, אף כי זמן הריצה שלו במקרה הגרוע (כמו זה של אלגוריתם רבין־קארפ) אינו טוב יותר מזה של האלגוריתם הנאיבי להתאמת מחרוזות.

סימון ומינוח

נסמן ב־Σ^* (קרי: "סיגמה־כוכבי") את קבוצת כל המחרוזות בעלות אורך סופי שאפשר ליצור מתווים הלקוחים מן האלפבית Σ. בפרק זה נעסוק רק במחרוזות בעלות אורך סופי. **המחרוזת הריקה** (empty string), שאורכה 0, המסומנת ב־ε, שייכת גם היא ל־Σ^*. אורכה של מחרוזת x מסומן על־ידי $|x|$. **השרשור** (concatenation) של שתי מחרוזות x ו־y, המסומן על־ידי xy, הוא מחרוזת באורך $|x| + |y|$ המורכבת מתווי x ואחריהם תווי y.

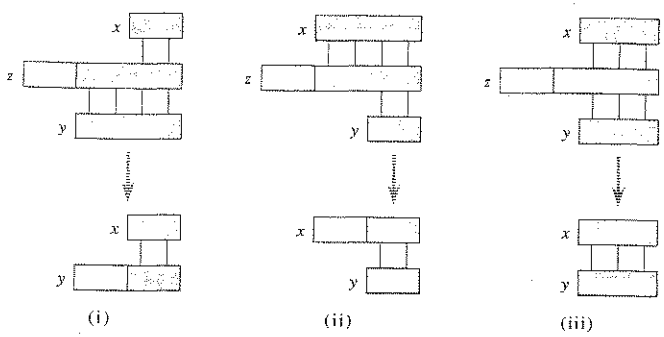
אנו אומרים שמחרוזת w היא **רישא** (prefix) של מחרוזת x, ומסמנים זאת על־ידי $w \sqsubset x$, אם קיימת מחרוזת $y \in \Sigma^*$ שעבורה $x = wy$. נשים לב כי אם $w \sqsubset x$, אזי $|w| \le |x|$. באופן דומה, אנו אומרים שמחרוזת w היא **סיפא** (suffix) של מחרוזת x, ומסמנים זאת על־ידי $w \sqsupset x$, אם קיימת מחרוזת $y \in \Sigma^*$ שעבורה $x = yw$. מ־$x = yw$ נובע $|w| \le |x|$. המחרוזת הריקה ε היא גם רישא וגם סיפא של כל מחרוזת. לדוגמה, $\mathtt{ab} \sqsubset \mathtt{abcca}$ ו־$\mathtt{cca} \sqsupset \mathtt{abcca}$. כדאי לשים לב שעבור כל שתי מחרוזות x ו־y ותו כלשהו a, מתקיים $x \sqsubset y$ אם ורק אם $xa \sqsubset ya$. עוד נשים לב כי $\sqsubset$ ו־$\sqsupset$ הם יחסים טרנזיטיביים. הלמה שלהלן תועיל לנו בהמשך.

למה 34.1 (למת הסיפות החופפות)

נניח ש־x, y ו־z הן מחרוזות המקיימות $x \sqsupset z$ ו־$y \sqsupset z$. אם $|x| \le |y|$, אזי $x \sqsupset y$. אם $|x| \ge |y|$, אזי $y \sqsupset x$. אם $|x| = |y|$, אזי $x = y$.

הוכחה ראה הוכחה גרפית באיור 34.2.

■



איור 34.2 הוכחה גרפית ללמה 34.2. אנו מניחים כי $x \sqsupset z$ ו־$y \sqsupset z$, שלושת חלקי האיור מדגימים את שלושת המקרים המופיעים בלמה. קווים אנכיים מחברים אזורים תואמים (המופיעים כשהם מוצללים) של המחרוזות. **(i)** אם $|y| \le |x|$, אזי $y \sqsupset x$. **(ii)** אם $|y| \ge |x|$, אזי $x \sqsupset y$. **(iii)** אם $|x| = |y|$, אזי $x = y$.

כדי לקצר את הסימונים, נסמן ב־P_k את הרישא בת k התווים $P[1..k]$ של התבנית $P[1..m]$. בסיכון זה, $P_0 = \varepsilon$ ו־$P_m = P = P[1..m]$. באופן דומה, נסמן ב־T_k את הרישא בת k התווים של הטקסט T. באמצעות סימון זה, נוכל לנסח את הבעיה של התאמת מחרוזות כבעיית מציאתם של כל ההיסטים s בתחום $0 \le s \le n-m$ שעבורם $P \sqsupset T_{s+m}$.

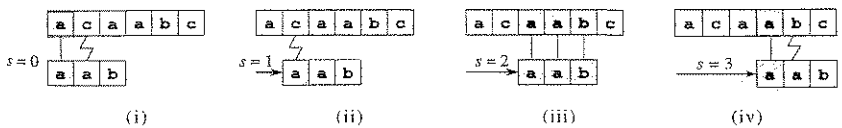
בפסאודו־קוד שלנו, נתייחס אל פעולת ההשוואה של שתי מחרוזות שוות־אורך כאל פעולת יסוד. אם השוואת המחרוזות נעשית משמאל לימין ונפסקת ברגע שמתגלה חוסר התאמה, נניח כי הזמן לביצוע בדיקה כזאת הוא פונקציה לינארית של מספר התווים התואמים שנמצאו. ביתר דיוק, אנו מניחים שהבדיקה "$x = y$" מתבצעת בזמן $\Theta(t+1)$, כאשר t הוא אורכה של המחרוזת הארוכה ביותר z המקיימת $x \sqsupset z$ ו־$y \sqsupset z$.

34.1 האלגוריתם הנאיבי להתאמת מחרוזות

האלגוריתם הנאיבי מוצא את כל ההיסטים התקפים באמצעות לולאה הבודקת אם התנאי $P[1..m] = T[s+1..s+m]$ מתקיים עבור כל אחד מ־$(n-m+1)$ הערכים האפשריים של s.

```
NAIVE-STRING-MATCHER(T, P)
1  n ← length[T]
2  m ← length[P]
3  for s ← 0 to n − m
4      do if P[1..m] = T[s + 1..s + m]
5          then print "Pattern occurs with shift" s
```

את פעולתה של השגרה הנאיבית להתאמת מחרוזות ניתן לדמות להחלקת "סרגל" המכיל את התבנית לאורך הטקסט, וחיפוש ההיסטים שעבורם כל התווים בתבנית שווים לתווים במקומות המקבילים בטקסט (ראה איור 34.3). לולאת ה־**for** המתחילה בשורה 3 גורמת בפועל כל היסט אפשרי. הבדיקה בשורה 4 קובעת אם ההיסט הנוכחי תקף או לא; בדיקה זו כרוכה למעשה בלולאה מובלעת המשווה בין תווים הנמצאים במיקומים מתאימים עד שמתגלה חוסר התאמה או עד שהיא מוצאת כי בכל המיקומים נמצאים תווים שווים. שורה 5 מדפיסה כל היסט תקף s.



איור 34.3 פעולתו של האלגוריתם הנאיבי להתאמת מחרוזות על התבנית $P = aab$ והטקסט $T = acaabc$. ניתן לדמות את פעולתו של האלגוריתם להחלקת "סרגל" המכיל את התבנית P לאורך הטקסט. בחלקים **(i)**-**(iv)** באיור מופיעים ארבעת ההיסטים העוקבים שבודק האלגוריתם הנאיבי. בכל חלק, קווים אנכיים מחברים אזורים מתאימים שנמצאו בהם תווים שווים (מוצללים באיור), וקו זיגזג מחבר את התווים הראשונים שנמצאו שונים זה מזה, אם יש כאלה. במקרה זה נמצא בטקסט מופע אחד של התבנית, בהיסט $s = 2$, והוא מוצג בחלק **(iii)**.

השגרה NAIVE-STRING-MATCHER רצה במקרה הגרוע בזמן $\Theta((n-m+1)m)$. לדוגמה, נתבונן במחרוזת הטקסט a^n (מחרוזת של n תווי a) ובתבנית a^m. עבור כל אחד מ־$(n-m+1)$ הערכים האפשריים של ההיסט s, הלולאה המובלעת בשורה 4 המשווה תווים במיקומים מתאימים חייבת להתבצע m פעמים עד שהיא מוודאת שההיסט אכן תקף. זמן הריצה במקרה הגרוע הוא אפוא $\Theta((n-m+1)m)$, השווה ל־$\Theta(n^2)$ אם $m = \lfloor n/2 \rfloor$.

כפי שנראה בהמשך, NAIVE-STRING-MATCHER אינה שגרה אופטימלית עבור בעיה זו. ואכן, בפרק זה נראה אלגוריתם שזמן הריצה שלו במקרה הגרוע הוא $O(n+m)$. האלגוריתם הנאיבי להתאמת מחרוזות אינו יעיל משום שהוא מתעלם לחלוטין מן המידע על הטקסט שנרכש עבור ערך אחד של s, בעת שהוא בוחן ערכי s אחרים. אבל מידע כזה יכול להיות בעל ערך רב. לדוגמה, אם $P = aaab$ ואנו מוצאים כי $s = 0$ הוא היסט תקף, אזי אף אחד מן ההיסטים 1, 2 או 3 אינו יכול להיות תקף, שכן $T[4] = b$. בסעיפים הבאים נבחן כמה דרכים לשימוש יעיל בסוג זה של מידע.

תרגילים

34.1-1

הצג את ההשוואות שמבצע האלגוריתם הנאיבי להתאמת מחרוזות בהינתן התבנית $P = 0001$ והטקסט $T = 000010001010001$.

34.1-2

הראה שבמקרה הגרוע, הזמן הדרוש לאלגוריתם הנאיבי להתאמת מחרוזות כדי למצוא את המופע *הראשון* של תבנית בטקסט הוא $\Theta((n-m+1)(m-1))$.

34.1-3

נניח שכל התווים בתבנית P שונים זה מזה. הראה כיצד ניתן להחיש את פעולתה של NAIVE-STRING-MATCHER כך שתרוץ בזמן $O(n)$ על טקסט T בן n תווים.

34.1-4

נניח שהתבנית P והטקסט T הם מחרוזות באורך m ו־n, בהתאמה, שנבחרו *באקראי* מן האלפבית בעל d התווים $\Sigma_d = \{0, 1, \ldots, d-1\}$, כאשר $d \ge 2$. הראה כי *תוחלת* מספר ההשוואות בין תווים בודדים שמבצעת הלולאה המובלעת בשורה 4 של האלגוריתם הנאיבי היא:

$$(n-m+1)\frac{1-d^{-m}}{1-d^{-1}} \le 2(n-m+1)$$

(הנח שבעת בחינת היסט נתון, האלגוריתם הנאיבי מפסיק לבצע השוואות ברגע שנמצא חוסר התאמה או כאשר נמצאה התאמה של התבנית כולה.) לכן, עבור מחרוזות אקראיות, האלגוריתם הנאיבי יעיל למדי.

34.1-5

נניח כי אנו מרשים שהתבנית P תכיל **תו מרווח** (gap character) ◊ שיכול להתאים למחרוזת *שרירותית* של תווים (גם למחרוזת באורך 0). לדוגמה, התבנית ab◊ba◊c מופיעה בטקסט cabccbacbacab באופן הבא:

```
c ab  cc  ba  cba  c  ab
  ab  ◊   ba  ◊    c
```

וגם באופן הבא:

```
c ab  ccbac  ba  _  c  ab
  ab    ◊    ba  ◊  c
```

שים לב כי תו המרווח יכול להופיע בתבנית מספר כלשהו של פעמים אבל ההנחה היא שאינו מופיע כלל בטקסט. מצא אלגוריתם שזמן ריצתו פולינומיאלי הקובע אם תבנית כזאת P מופיעה בטקסט נתון T. נתח את זמן הריצה של האלגוריתם שלך.

34.2 אלגוריתם רבין־קארפ

רבין וקארפ (Rabin & Karp) הציעו אלגוריתם התאמת־מחרוזות שביצועיו ביישומים מעשיים טובים, וכמו כן ניתן להכלילו לבעיות דומות אחרות, כגון התאמת מחרוזות דו־ממדית. זמן הריצה של אלגוריתם רבין־קארפ במקרה הגרוע הוא $O((n-m+1)m)$, אולם ביצועיו טובים במקרה הממוצע.

האלגוריתם עושה שימוש במושגים יסודיים מתורת המספרים, כמו שקילות של שני מספרים מודולו מספר שלישי. הקורא מוזמן לעיין בסעיף 33.1, שם מובאות ההגדרות הרלוונטיות.

לצורך ההסבר, הבה נניח כי $\Sigma = \{0, 1, 2, \ldots, 9\}$, כלומר, כל תו הוא ספרה עשרונית. (במקרה הכללי ניתן להניח שכל תו הוא ספרה בבסיס d, כאשר $d = |\Sigma|$.) נוכל אפוא להסתכל על מחרוזת בת k תווים עוקבים כאילו היא מייצגת מספר עשרוני באורך k. לדוגמה, מחרוזת התווים 31415 מייצגת את המספר העשרוני 31,415. בהינתן פירוש כפול זה של תווי הקלט, הן כסימנים גרפיים והן כספרות, מצאנו כי נוח לכתוב אותם בסעיף זה באותו גופן (font) שבו נכתבות בדרך כלל הספרות בטקסט.

בהינתן תבנית $P[1..m]$, נסמן ב־p את הערך העשרוני שמייצגת התבנית. באופן דומה, בהינתן טקסט $T[1..n]$, נסמן ב־t_s את הערך העשרוני של התת־מחרוזת באורך m, $T[s+1..s+m]$, עבור $s = 0, 1, \ldots, n-m$. ברור כי $t_s = p$ אם ורק אם $T[s+1..s+m] = P[1..m]$; לכן, s הוא היסט תקף אם ורק אם $t_s = p$. אם נוכל לחשב את p בזמן $O(m)$ ואת כל הערכים t_s בזמן כולל $O(n)$, אזי נוכל למצוא את כל ההיסטים התקפים s בזמן $O(n)$ על־ידי השוואת p עם כל אחד מערכי t_s. (לעת עתה, בל תדאיג אותנו האפשרות שהערכים של p ושל ה־t_s עלולים להיות גדולים מאוד.)

את p ניתן לחשב בזמן $O(m)$ באמצעות כלל הורנר (ראה סעיף 32.1):

$$p = P[m] + 10(P[m-1] + 10(P[m-2] + \cdots + 10(P[2] + 10(P[1]))\cdots))$$

בצורה דומה ניתן לחשב את הערך t_0 מתוך $T[1..m]$ בזמן $O(m)$.

כדי לחשב את הערכים הנותרים $t_1, t_2, \ldots, t_{n-m}$ בזמן $O(n-m)$, די לשים לב כי את t_{s+1} ניתן לחשב מ־t_s בזמן קבוע, שכן:

$$t_{s+1} = 10(t_s - 10^{m-1}T[s+1]) + T[s+m+1] \tag{34.1}$$

לדוגמה, אם $m = 5$ ו־$t_s = 31415$, אזי ברצוננו לסלק את הספרה המשמעותית ביותר $T[s+1] = 3$, לצרף את הספרה הפחות משמעותית (נניח שזוהי $T[s+5+1] = 2$), ולקבל:

$$\begin{aligned} t_{s+1} &= 10(31415 - 10000 \cdot 3) + 2 \\ &= 14152 \end{aligned}$$

חיסור $10^{m-1}T[s+1]$ מסלק מ־t_s את הספרה המשמעותית ביותר, כפל התוצאה ב־10 מזיז את המספר מקום אחד שמאלה, וחיבור $T[s+m+1]$ מצרף את הספרה הפחות משמעותית המתאימה. אם מחשבים מראש את הקבוע 10^{m-1} (חישוב שניתן לבצעו בזמן $O(\lg m)$ באמצעות השיטות מסעיף 33.6, אף כי עבור יישום זה ניתן להסתפק בחישוב הישיר המתבצע בזמן $O(m)$), אזי כל חישוב של משוואה (34.1) מתבצע באמצעות מספר קבוע של פעולות אריתמטיות. לכן, ניתן לחשב את p ואת $t_0, t_1, \ldots, t_{n-m}$ כולם בזמן כולל של $O(n+m)$, וניתן למצוא את כל המופעים של התבנית $P[1..m]$ בטקסט $T[1..n]$ בזמן $O(n+m)$.

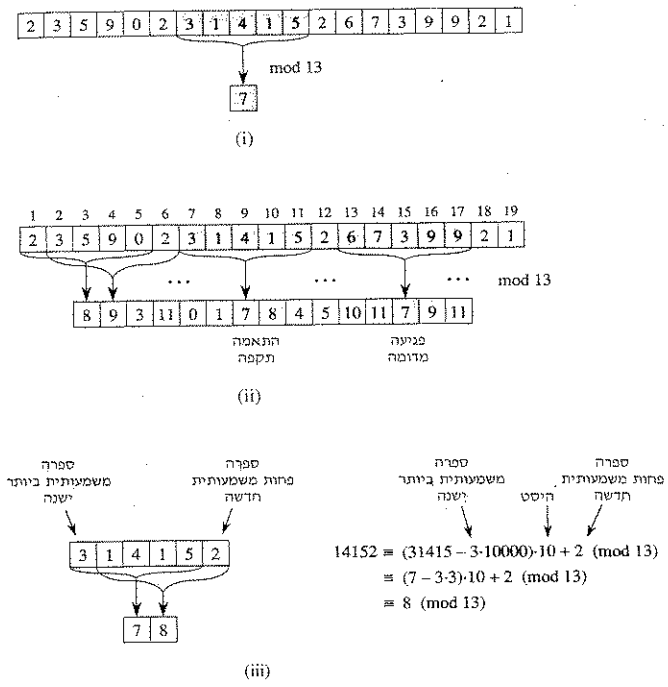
הקושי היחיד בתהליך זה הוא האפשרות שערכי p ו־t_s יהיו גדולים מדי לעבודה נוחה. אם P מכיל m תווים, אזי ההנחה שכל פעולה אריתמטית על p (שהוא מספר בן m ספרות) מתבצעת ב"זמן קבוע" אינה סבירה. למרבה המזל, קיימת תרופה פשוטה לבעיה זו, כמתואר באיור 34.4: חישוב ערכי p וה־t_s מודולו מספר מתאים q. מכיוון שהחישובים של p, t_0 ונוסחת הנסיגה (34.1) יכולים כולם להתבצע מודולו q, הרי שניתן לחשב את ערכי p וה־t_s מודולו q בזמן $O(n+m)$. בדרך כלל בוחרים כ־q מספר ראשוני כזה ש־$10q$ נכנס בדיוק לתוך מילת מחשב אחת, דבר המאפשר לבצע את כל החישובים הנחוצים באמצעות אריתמטיקה של "דיוק יחיד" (single-precision).

באופן כללי, כאשר עובדים עם אלפבית בעל d תווים $\{0, 1, \ldots, d-1\}$, בוחרים q כך ש־dq נכנס לתוך מילת מחשב אחת, ומתאימים את נוסחת הנסיגה (34.1) לעריכת החישובים מודולו q באופן הבא:

$$t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) \bmod q \tag{34.2}$$

כאשר $h \equiv d^{m-1} \pmod q$ הוא ערכה של הספרה "1" במיקום המשמעותי ביותר בחלון טקסט בן m ספרות.

אבל, אליה וקוץ בה; בעבודה מודולו q, $t_s \equiv p \pmod q$ אינו גורר $t_s = p$. מצד שני, אם $t_s \not\equiv p \pmod q$, אזי אנו יודעים בוודאות כי $t_s \neq p$, ולכן ההיסט s אינו תקף. לכן, נוכל להשתמש בבדיקה $t_s \equiv p \pmod q$ כבדיקה היוריסטית מהירה שמטרתה להוציא מכלל חשבון את כל ההיסטים s הבלתי־תקפים. כל היסט s שעבורו $t_s \equiv p \pmod q$ יש להמשיך ולבדוק כדי לגלות אם s הוא אכן היסט תקף או שזוהי **פגיעה מדומה** (spurious hit). בדיקה זו יכולה להיעשות על־ידי בדיקה מפורשת של התנאי $P[1..m] = T[s+1..s+m]$. אם q גדול דיו, אזי ניתן לקוות שפגיעות מדומות יופיעו בתדירות נמוכה דיה, כך שעלות הבדיקות הנוספות תהיה נמוכה.



איור 34.4 אלגוריתם רבין־קארפ. כל תו הוא ספרה עשרונית והחישובים מבוצעים מודולו 13. **(i)** מחרוזת טקסט. באיור מוצלל חלון באורך 5. הערך המספרי של המספר המוצלל מחושב מודולו 13, והתוצאה היא 7. **(ii)** אותה מחרוזת טקסט עם הערכים המחושבים מודולו 13 עבור כל מיקום אפשרי של חלון באורך 5. בהנחה שהתבנית היא $P = 31415$, אנו מחפשים חלונות שערכם מודולו 13 הוא 7, שכן $31415 \equiv 7 \pmod{13}$. מתגלים שני חלונות כאלה (והם מוצללים באיור). הראשון, המתחיל במיקום 7 בטקסט, הוא אכן מופע של התבנית; אבל השני, המתחיל במיקום 13 בטקסט, הוא פגיעה מדומה. **(iii)** חישוב ערכו של חלון בזמן קבוע, בהינתן ערכו של החלון הקודם. ערכו של החלון הראשון הוא 31415. משלקים את הספרה המשמעותית ביותר 3, מזיזים מקום אחד שמאלה (ובכך כופלים ב־10), מצרפים את הספרה הכי פחות משמעותית 2, ומקבלים את הערך החדש 14152. אבל כל החישובים מבוצעים מודולו 13, ולכן ערכו של החלון הראשון הוא 7, והערך המחושב עבור החלון החדש הוא 8.

השגרה שלהלן מממשת רעיונות אלה. הקלטים לשגרה הם הטקסט T, התבנית P, הבסיס d (בדרך כלל זהו $|\Sigma|$) והמספר הראשוני q.


```
RABIN-KARP-MATCHER(T, P, d, q)
 1  n ← length[T]
 2  m ← length[P]
 3  h ← d^(m−1) mod q
 4  p ← 0
 5  t_0 ← 0
 6  for i ← 1 to m
 7      do p ← (dp + P[i]) mod q
 8         t_0 ← (dt_0 + T[i]) mod q
 9  for s ← 0 to n − m
10      do if p = t_s
11            then if P[1..m] = T[s + 1..s + m]
12                    then print "Pattern occurs with shift" s
13         if s < n − m
14            then t_{s+1} ← (d(t_s − T[s + 1]h) + T[s + m + 1]) mod q
```

השגרה RABIN-KARP-MATCHER פועלת כדלקמן: השגרה מפרשת את כל התווים כספרות בבסיס d. האינדקסים של t נרשמו כאן לשם בהירות בלבד; התכנית פועלת כהלכה גם כאשר משמיטים את כל האינדקסים. שורה 3 מאתחלת את h לערך של מיקום הספרה המשמעותית ביותר בחלון בן m ספרות. שורות 4-8 מחשבות את p כערך של $P[1..m]$ מודולו q ואת t_0 כערך של $T[1..m]$ מודולו q. לולאת ה־**for** המתחילה בשורה 9 מתבצעת עבור כל היסט אפשרי s. הלולאה משמרת את האינווריאנטה הבאה: בכל עת שמבוצעת שורה 10, $t_s = T[s + 1..s + m] \bmod q$. אם שורה 10 מוצאת כי $p = t_s$ ("פגיעה"), אזי שורה 11 בודקת אם $P[1..m] = T[s + 1..s + m]$ כדי לפסול אפשרות של פגיעה מדומה. כל היסט תקף שהתגלה מודפס בשורה 12. אם $s < n - m$ (בדיקה המתבצעת בשורה 13), אזי יש לבצע את לולאת ה־**for** לפחות עוד פעם אחת, ולכן מתבצעת תחילה שורה 14 כדי לוודא שהאינווריאנטה תתקיים גם כאשר שורה 10 תתבצע שוב. שורה 14 מחשבת בזמן קבוע את הערך של $t_{s+1} \bmod q$ ישירות מהערך $t_s \bmod q$ באמצעות משוואה (34.2).

זמן הריצה של RABIN-KARP-MATCHER במקרה הגרוע הוא $\Theta((n - m + 1)m)$, שכן (כמו האלגוריתם הנאיבי להתאמת מחרוזות) אלגוריתם רבין־קארפ מאמת במפורש כל היסט תקף. אם $P = a^m$ ו־$T = a^n$, אזי אימותים אלה מתבצעים בזמן $\Theta((n - m + 1)m)$, שכן כל אחד מ־$(n - m + 1)$ ההיסטים האפשריים הוא תקף. (נשים לב גם לכך שלחישוב $d^{m-1} \bmod q$ בשורה 3 ולביצוע הלולאה בשורות 6-8 דרוש זמן של $O(m)$.)

ביישומים רבים אנו מצפים להיסטים תקפים מעטים (אולי $O(1)$ כאלה), ולכן תוחלת זמן הריצה של האלגוריתם היא $O(n + m)$ ועוד הזמן הדרוש לעיבוד הפגיעות המדומות. ניתן לבסס ניתוח היוריסטי על ההנחה שצמצום הערכים מודולו q פועל כמו מיפוי אקראי מ־Σ^* ל־$\mathbf{Z}_q$. (ראה הדיון בסעיף 12.3.1 על השימוש בחילוק לשם גיבוב. קשה לנסח באופן פורמלי הנחה כזאת ולהוכיח אותה, אף כי גישה אפשרית אחת היא להניח ש־q נבחר אקראית מבין השלמים בגודל המתאים. אנו לא נמשיך כאן בטיעון פורמלי זה.) ניתן לצפות אז שמספר הפגיעות המדומות יהיה $O(n/q)$, שכן את הסיכוי ש־t_s שרירותי ישווה ל־p מודולו q ניתן לאמוד ב־$1/q$. תוחלת הזמן לביצוע אלגוריתם רבין־קארפ היא אפוא:

$$O(n) + O(m(v + n/q))$$

כאשר v הוא מספר ההיסטים התקפים. זמן ריצה זה הוא $O(n)$ אם בוחרים $q \geq m$. כלומר, אם תוחלת מספר ההיסטים התקפים היא קטנה ($O(1)$) והמספר הראשוני q נבחר כך שהוא גדול יותר מאורך התבנית, אזי ניתן לצפות שאלגוריתם רבין־קארפ ירוץ בזמן $O(n + m)$.

תרגילים

34.2-1
בעבודה מודולו $q = 11$, כמה פגיעות מדומות מוצא אלגוריתם רבין־קארפ כאשר הוא מחפש את התבנית $P = 26$ בטקסט $T = 3141592653589793$?

34.2-2
כיצד היית מרחיב את שיטת רבין־קארפ עבור הבעיה של חיפוש בטקסט נתון אחר מופע של תבנית כלשהי מבין קבוצה נתונה של k תבניות?

34.2-3
הראה כיצד להרחיב את שיטת רבין־קארפ לטיפול בבעיה של חיפוש תבנית נתונה של $m \times m$ במערך תווים בגודל $n \times n$. (מותר להסיט את התבנית אנכית או אופקית אולם אסור לסובב אותה.)

34.2-4
לאיה יש עותק של קובץ ארוך בן n סיביות $A = \langle a_{n-1}, a_{n-2}, \ldots, a_0 \rangle$ וגם לבועז יש קובץ בן n סיביות $B = \langle b_{n-1}, b_{n-2}, \ldots, b_0 \rangle$. איה ובועז רוצים לדעת אם הקבצים שלהם זהים. כדי להימנע מהעברה של הקובץ A כולו או של B כולו, הם משתמשים בבדיקה ההסתברותית המהירה הבאה: הם בוחרים ביחד מספר ראשוני $q > 1000n$ ושלח אקראי x מתוך $\{0, 1, \ldots, q-1\}$. איה מחשבת את:

$$A(x) = \left(\sum_{i=0}^{n} a_i x^i\right) \bmod q$$

ובועז מחשב באופן דומה את $B(x)$. הוכח שאם $A \neq B$, אזי קיים סיכוי של 1 ל־1000 לכל היותר ש־$A(x) = B(x)$, ואילו אם שני הקבצים זהים, אזי בהכרח $A(x) = B(x)$. (**רמז:** ראה תרגיל 33.4-4.)

34.3 התאמת מחרוזות באמצעות אוטומטים סופיים

אלגוריתמים רבים להתאמת מחרוזות בונים אוטומט סופי הסורק את מחרוזת הטקסט T כדי לגלות בה את כל המופעים של התבנית P. בסעיף זה נציג שיטה לבניית אוטומט כזה. אוטומטים כאלה להתאמת מחרוזות הם יעילים מאוד: הם בוחנים כל תו בטקסט *בדיוק פעם אחת*, בזמן קבוע לכל תו. הזמן הדרוש — לאחר בניית האוטומט — הוא אפוא $\Theta(n)$. אולם, הזמן לצורך בניית האוטומט עשוי להיות ארוך אם Σ גדול. סעיף 34.4 מתאר דרך חכמה לעקיפת בעייה זו.

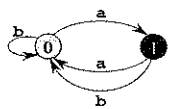
נפתח סעיף זה בהגדרה של אוטומט סופי. לאחר מכן נבחן אוטומט מיוחד להתאמת מחרוזות ונראה כיצד ניתן להשתמש בו למציאת מופעים של תבנית בטקסט. הדיון יכלול גם פרטים על הדרך שבה ניתן לדמות את התנהגותו של אוטומט להתאמת מחרוזות על טקסט נתון. לבסוף, נראה כיצד לבנות אוטומט להתאמת מחרוזות עבור תבנית קלט נתונה.

אוטומטים סופיים

אוטומט סופי (finite automaton) M הוא חמישייה $(Q, q_0, A, \Sigma, \delta)$, כאשר:

- Q היא קבוצה סופית של **מצבים** (states),
- $q_0 \in Q$ הוא **המצב ההתחלתי** (start state),
- $A \subseteq Q$ היא קבוצה של **מצבים מקבלים** (accepting states),
- Σ הוא **אלפבית קלט** (input alphabet) סופי,
- δ היא פונקציה מ־$Q \times \Sigma$ ל־Q הנקראת **פונקציית המעברים** (transition function) של M.

האוטומט הסופי מתחיל במצב q_0 וקורא את תווי מחרוזת הקלט בזה אחר זה. אם האוטומט נמצא במצב q וקורא את תו הקלט a, הוא עובר (״מבצע מעבר״) מהמצב q למצב $\delta(q, a)$. בכל פעם שהמצב הנוכחי q שבו נמצא האוטומט הוא איבר של A, אנו אומרים שהמחרוזת שקראה המכונה M עד כה **נתקבלה** (accepted). על מחרוזת קלט שאינה מתקבלת אנו אומרים שהיא **נדחית** (rejected). באיור 34.5 מודגמות הגדרות אלה באמצעות אוטומט פשוט בעל שני מצבים.



(i)

(ii)

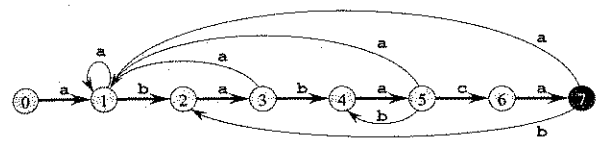
איור 34.5 אוטומט סופי פשוט בעל שני מצבים, שבו קבוצת המצבים היא $Q = \{0, 1\}$, המצב ההתחלתי הוא $q_0 = 0$, ואלפבית הקלט הוא $\Sigma = \{a, b\}$. **(i)** טבלה המייצגת את פונקציית המעברים δ. **(ii)** תרשים שקול של מעברים בין מצבים. מצב 1 הוא המצב המקבל היחיד (מופיע באיור בשחור). קשתות מכוונות מייצגות מעברים. לדוגמה, הקשת ממצב 1 למצב 0 המסומנת ב־b מציינת כי $\delta(1, b) = 0$. אוטומט זה מקבל את המחרוזות המסתיימות במספר אי־זוגי של תווי a. ביתר דיוק, האוטומט מקבל מחרוזת x אם ורק אם $x = yz$, כאשר $y = \varepsilon$ או y מסתיימת ב־b, ו־$z = a^k$ עבור k אי־זוגי. לדוגמה, עבור הקלט abaaa עובר אוטומט זה דרך סדרת המצבים (כולל המצב ההתחלתי) $\langle 0, 1, 0, 1, 0, 1 \rangle$, ומקבל בסופו של דבר את הקלט. עבור הקלט abbaa, סדרת המצבים היא $\langle 0, 1, 0, 0, 1, 0 \rangle$, כך שהקלט נדחה.

אוטומט סופי M משרה פונקציה ϕ הנקראת **פונקציית המצבים הסופיים** (final-state function). זוהי פונקציה מ־Σ^* ל־Q כך ש־$\phi(w)$ הוא המצב שבו נמצא M לאחר סריקת המחרוזת w כולה. לכן, M מקבל את המחרוזת w אם ורק אם $\phi(w) \in A$. הפונקציה ϕ מוגדרת על־ידי היחס הרקורסיבי הבא:

$$\begin{aligned} \phi(\varepsilon) &= q_0 \\ \phi(wa) &= \delta(\phi(w), a) \qquad \text{עבור } w \in \Sigma^*, a \in \Sigma \end{aligned}$$

אוטומטים להתאמת מחרוזות

ניתן לבנות אוטומט להתאמת מחרוזות עבור כל תבנית P; את האוטומט יש לבנות על פי התבנית בעיבוד מקדים לפני שניתן להשתמש בו לחיפוש התבנית בטקסט. באיור 34.6 מודגמת בנייתו של אוטומט סופי עבור התבנית $P = \text{ababaca}$. מכאן ואילך נניח ש־P היא מחרוזת נתונה בתבנית קבועה; לשם קיצור, לא נציין את התלות ב־P בסימונים שלנו.



(i)

קלט

מצב	a	b	c	P
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	4	0	b
4	5	0	0	a
5	1	4	6	c
6	7	0	0	a
7	1	2	0	

(ii)

i	—	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	—	a	b	a	b	a	b	a	c	a	b	a
$\phi(T_i)$ מצב	0	1	2	3	4	5	4	5	6	7	2	3

(iii)

איור 34.6 **(i)** תרשים מצבים עבור אוטומט להתאמת מחרוזות שמקבל את כל המחרוזות המסתיימות במחרוזת ababaca. מצב 0 הוא המצב ההתחלתי, ומצב 7 (שחור באיור) הוא המצב המקבל היחיד. קשת מכוונת ממצב i למצב j המסומנת ב־a מייצגת $\delta(i, a) = j$. הקשתות המכוונות ימינה, היוצרות את "חוט השדרה" של האוטומט ומודגשות באיור, מציינות התאמות מוצלחות בין תווי התבנית לתווי הקלט. הקשתות המכוונות שמאלה מייצגות התאמות שנכשלו. חלק מן הקשתות המייצגות התאמות שנכשלו אינן מופיעות באיור; על פי מוסכמה, אם עבור מצב i ואיזשהו תו $a \in \Sigma$ לא קיימת קשת היוצאת מ־i ומסומנת ב־a, פירוש הדבר ש־$\delta(i, a) = 0$. **(ii)** פונקציית המעברים המתאימה δ והתבנית $P = \text{ababaca}$. תאי הטבלה המייצגים התאמות מוצלחות בין תווי התבנית לתווי הקלט מופיעים מוצללים. **(iii)** פעולתו של האוטומט על הטקסט $T = \text{ababababacaba}$. מתחת לכל תו $T[i]$ של הטקסט מופיע המצב $\phi(T_i)$ שבו נמצא האוטומט לאחר עיבוד תרישא T_i. האוטומט מוצא מופע אחד של התבנית, המסתיים במיקום 9.

כדי לציין את האוטומט להתאמת מחרוזות המתאים לתבנית נתונה $P[1..m]$, נגדיר תחילה פונקציית עזר σ, הנקראת **פונקציית הסיפא** (suffix function) המתאימה ל־P. הפונקציה σ היא מיפוי מ־Σ^* ל־$\{0, 1, \ldots, m\}$ כך ש־$\sigma(x)$ הוא אורכה של הרישא הארוכה ביותר של P שהיא סיפא של x:

$$\sigma(x) = \max\{k : P_k \sqsupset x\}$$

פונקציית הסיפא σ מוגדרת היטב, שכן המחרוזת הריקה $P_0 = \varepsilon$ היא סיפא של כל מחרוזת. לדוגמה, עבור התבנית $P = \text{ab}$, נקבל $\sigma(\varepsilon) = 0$, $\sigma(\text{ccaca}) = 1$, ו־$\sigma(\text{ccab}) = 2$. עבור תבנית P באורך m מתקיים $\sigma(x) = m$ אם ורק אם $P \sqsupset x$. מהגדרתה של פונקציית הסיפא נובע שאם $x \sqsupset y$, אזי $\sigma(x) \le \sigma(y)$.

את האוטומט להתאמת מחרוזות המתאים לתבנית נתונה $P[1..m]$ אנו מגדירים באופן הבא:

- קבוצת המצבים Q היא הקבוצה $\{0, 1, \ldots, m\}$. המצב ההתחלתי q_0 הוא המצב 0, והמצב m הוא המצב המקבל היחיד.
- פונקציית המעברים δ מוגדרת על־ידי המשוואה שלהלן. עבור כל מצב q וכל תו a:

$$\delta(q, a) = \sigma(P_q a) \tag{34.3}$$

להגדרה $\delta(q, a) = \sigma(P_q a)$ ניתן להציע את ההסבר האינטואיטיבי שלהלן. המכונה משמרת בעת פעולתה את האינווריאנטה:

$$\phi(T_i) = \sigma(T_i) \tag{34.4}$$

תוצאה זו מוכחת בהמשך, במשפט 34.4. בניסוח מילולי, פירוש הדבר שאחרי סריקת i התווים הראשונים של מחרוזת הטקסט T, המכונה נמצאת במצב $\phi(T_i) = q$, כאשר $q = \sigma(T_i)$ הוא אורכה של הסיפא הארוכה ביותר של T_i שהיא גם רישא של התבנית P. אם התו הבא שנסרק הוא $T[i+1] = a$, אזי המכונה צריכה לעבור למצב $\sigma(T_i a) = \sigma(T_{i+1})$. הוכחת המשפט מראה כי $\sigma(T_i a) = \sigma(P_q a)$; דהיינו, כדי לחשב את אורכה של הסיפא הארוכה ביותר של $T_i a$ שהיא גם רישא של P, נוכל לחשב את הסיפא הארוכה ביותר של $P_q a$ שהיא גם רישא של P. בכל מצב, המכונה צריכה לדעת רק את אורכה של הרישא הארוכה ביותר של P שהיא סיפא של המחרוזת שנקראה עד כה. לכן, ההצבה $\delta(q, a) = \sigma(P_q a)$ משמרת את האינווריאנטה הרצויה (34.4). טיעון בלתי פורמלי זה יונסח במדויק בהמשך.

לדוגמה, באוטומט להתאמת מחרוזות שבאיור 34.6 מקבלים $\delta(5, \text{b}) = 4$. דבר זה נובע מן העובדה שאם האוטומט קורא תו b במצב $q = 5$, אזי $P_q\text{b} = \text{ababab}$, והרישא הארוכה ביותר של P שהיא גם סיפא של ababab היא $P_4 = \text{abab}$.

כדי להבהיר את אופן פעולתו של אוטומט להתאמת מחרוזות, נציג עתה תכנית פשוטה ויעילה המדמה את התנהגותו של אוטומט כזה (המיוצג על־ידי פונקציית המעברים שלו δ) במציאת המופעים של תבנית P באורך m בטקסט קלט $T[1..n]$. כמו בכל אוטומט להתאמת מחרוזות עבור תבנית באורך m, קבוצת המצבים Q היא $\{0, 1, \ldots, m\}$, המצב ההתחלתי הוא 0, והמצב המקבל היחיד הוא המצב m.

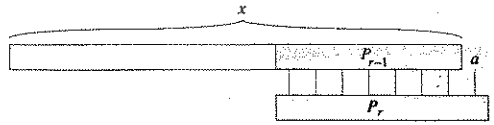
```
FINITE-AUTOMATON-MATCHER(T, δ, m)
1  n ← length[T]
2  q ← 0
3  for i ← 1 to n
4      do q ← δ(q, T[i])
5         if q = m
6            then s ← i − m
7                 print "Pattern occurs with shift" s
```

מבנה הלולאה הפשוט של FINITE-AUTOMATON-MATCHER נובע שזמן הריצה שלה על מחרוזת טקסט באורך n הוא $O(n)$. אולם, זמן ריצה זה אינו כולל את הזמן הנדרש לחישוב פונקציית המעברים δ. לבעיה זו נפנה מאוחר יותר, לאחר שנוכיח כי FINITE-AUTOMATON-MATCHER פועלת כהלכה.

נתבונן בפעולתו של האוטומט על טקסט קלט $T[1..n]$. נוכיח שלאחר סריקת התו $T[i]$ האוטומט נמצא במצב $\sigma(T_i)$. מאחר ש־$m = \sigma(T_i)$ אם ורק אם $P \sqsupset T_i$, הרי שהמכונה תימצא במצב המקבל m אם ורק אם התבנית P נסרקה זה עתה. כדי להוכיח תוצאה זו, ניעזר בשתי הלמות הבאות, העוסקות בפונקציית הסיפא σ.

למה 34.2 (אי־שוויון ערכי פונקציית הסיפא)
לכל מחרוזת x ותו a, מתקיים $\sigma(xa) \le \sigma(x) + 1$.

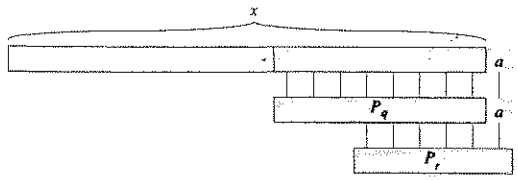
הוכחה בהתייחס לאיור 34.7, יהי $r = \sigma(xa)$. אם $r = 0$, אזי המסקנה $r \le \sigma(x) + 1$ מתקיימת באופן טריוויאלי, שכן $\sigma(x)$ הוא אי־שלילי. נניח אפוא כי $r > 0$. כעת, $P_r \sqsupset xa$ על פי הגדרת σ. לכן, $P_{r-1} \sqsupset x$, על־ידי השמטת a מסופה של P_r ומסופה של xa. לכן, $r - 1 \le \sigma(x)$, שכן $\sigma(x)$ שווה ל־k הגדול ביותר שעבורו $P_k \sqsupset x$. ■



איור 34.7 הדגמת ההוכחה של למה 34.2. האיור מראה כי $r \le \sigma(x) + 1$, כאשר $r = \sigma(xa)$.

למה 34.3 (למת הרקורסיה של פונקציית הסיפא)
לכל מחרוזת x ותו a, אם $q = \sigma(x)$, אזי $\sigma(xa) = \sigma(P_q a)$.

הוכחה מהגדרת σ מקבלים כי $P_q \succ x$. כפי שמראה איור 34.8, $P_q a \succ xa$. אם נציב $r = \sigma(xa)$, אזי על פי למה 34.2, $r \le q + 1$. מאחר ש־$xa \succ P_q a$, ו־$P_r \succ xa$, נובע כי $|P_r| \le |P_q a|$. אזי למה 34.1 גוררת כי $P_r \succ P_q a$. לכן, $r \le \sigma(P_q a)$, כלומר, $\sigma(xa) \le \sigma(P_q a)$. אבל מתקיים גם $\sigma(P_q a) \le \sigma(xa)$, שכן $P_q a \succ xa$. מכאן נקבל כי $\sigma(xa) = \sigma(P_q a)$. ■



איור 34.8 הדגמת ההוכחה של למה 34.3. האיור מראה כי $r = \sigma(P_q a)$, כאשר $q = \sigma(x)$ ו־$r = \sigma(xa)$.

עתה אנו ערוכים להוכחת המשפט העיקרי שלנו, המאפיין את התנהגותו של אוטומט להתאמת מחרוזות על טקסט קלט נתון. כפי שציינו לעיל, משפט זה מראה שהאוטומט פשוט עוקב בכל צעד אחר הרישא הארוכה ביותר של התבנית שהיא גם סיפא של המחרוזת שנקראה עד לצעד זה.

משפט 34.4
אם ϕ היא פונקציית המצבים הסופיים של אוטומט להתאמת מחרוזות עבור תבנית נתונה P ו־$T[1..n]$ הוא טקסט קלט לאוטומט, אזי, עבור $i = 0, 1, \ldots, n$:

$$\phi(T_i) = \sigma(T_i)$$

הוכחה ההוכחה נעשית באינדוקציה על i. עבור $i = 0$, המשפט נכון באופן טריוויאלי, שכן $T_0 = \varepsilon$. לכן, $\phi(T_0) = \sigma(T_0) = 0$.

עתה, נניח כי $\phi(T_i) = \sigma(T_i)$ ונוכיח כי $\phi(T_{i+1}) = \sigma(T_{i+1})$. נסמן ב־$q$ את $\phi(T_i)$ וב־a את $T[i+1]$. אזי:

$$\begin{aligned}
\phi(T_{i+1}) &= \phi(T_i a) && \text{(על־פי הגדרת } a\text{ ו־}T_{i+1}\text{)}\\
&= \delta(\phi(T_i), a) && \text{(על־פי הגדרת } \phi\text{)}\\
&= \delta(q, a) && \text{(על־פי הגדרת } q\text{)}\\
&= \sigma(P_q a) && \text{(על־פי ההגדרה (34.3) של } \delta\text{)}\\
&= \sigma(T_i a) && \text{(על־פי למה 34.3 והנחת האינדוקציה)}\\
&= \sigma(T_{i+1}) && \text{(על־פי הגדרת } T_{i+1}\text{)}
\end{aligned}$$

ובאינדוקציה, המשפט מוכח. ■

על־פי משפט 34.4, אם המכונה נכנסת למצב q בשורה 4, אזי q הוא הערך הגדול ביותר כך ש־$P_q \succ T_i$. לכן, אנו מקבלים $q = m$ בשורה 5 אם ורק אם נסרק זה עתה מופע של התבנית P. אנו מסיקים אפוא שהשגרה FINITE-AUTOMATON-MATCHER פועלת כהלכה.

חישוב פונקציית המעברים

השגרה הבאה מחשבת את פונקציית המעברים δ עבור תבנית נתונה $P[1..m]$.

```
COMPUTE-TRANSITION-FUNCTION(P, Σ)
1  m ← length[P]
2  for q ← 0 to m
3      do for each character a ∈ Σ
4          do k ← min(m + 1, q + 2)
5             repeat k ← k − 1
6             until P_k ⊐ P_q a
7             δ(q, a) ← k
8  return δ
```

שגרה זו מחשבת את $\delta(q, a)$ באופן ישיר, על־פי הגדרתה של δ. הלולאות המקוננות המתחילות בשורות 2 ו־3 בוחנות את כל המצבים q והתווים a, ושורות 4-7 מציבות ב־$\delta(q, a)$ את ה־k הגדול ביותר שעבורו $P_k \succ P_q a$. הקוד מתחיל מן הערך הגדול ביותר האפשרי של k, שהוא $\min(m, q + 1)$, ובכל פעם מקטין את k ב־1 עד אשר $P_k \succ P_q a$.

זמן הריצה של COMPUTE-TRANSITION-FUNCTION הוא $O(m^3|\Sigma|)$, משום שהלולאות החיצוניות תורמות לזמן הריצה גורם של $m|\Sigma|$, לולאת ה־**repeat** הפנימית יכולה להתבצע $m + 1$ פעמים לכל היותר, והבדיקה $P_k \succ P_q a$ בשורה 6 כרוכה בהשוואה של m תווים לכל היותר. קיימות שגרות מהירות בהרבה; את הזמן הדרוש לחישוב δ בהינתן P ניתן לשפר ל־$O(m|\Sigma|)$ על־ידי חישוב מתוחכם של מידע אודות התבנית P וניצולו (ראה תרגיל 34.4-6). כאשר משתמשים בשגרה משופרת זו לחישוב δ, זמן הריצה הכולל למציאת כל המופעים של תבנית באורך m בטקסט באורך n מעל אלפבית Σ, הוא $O(n + m|\Sigma|)$.

תרגילים

34.3-1
בנה את האוטומט להתאמת מחרוזות עבור התבנית $P = \text{aabab}$ והדגם את פעולתו על מחרוזת הטקסט $T = \text{aaababaabaababaab}$.

34.3-2
שרטט תרשים מצבים של אוטומט להתאמת מחרוזות עבור התבנית $\text{ababbabbababbababbabb}$ מעל האלפבית $\Sigma = \{\text{a}, \text{b}\}$.

34.3-3
אנו מכנים תבנית P תבנית **נטולת־חפיפות** (nonoverlappable) אם $P_k \succ P_q$ גורר $k = 0$ או $k = q$. תאר את תרשים המצבים של אוטומט להתאמת מחרוזות עבור תבנית נטולת־חפיפות.

***34.3-4**

בהינתן שתי תבניות P ו־P', תאר כיצד בונים אוטומט סופי המוצא בטקסט את כל המופעים של שתי התבניות. נסה למזער את מספר המצבים באוטומט שלך.

***34.3-5**

בהינתן תבנית P המכילה תווי "מרווח" (ראה תרגיל 34.1-5), תראה כיצד בונים אוטומט סופי המוצא את המופעים של P בטקסט T בזמן $O(n)$, כאשר $n = |T|$.

34.4 אלגוריתם קנות'־מוריס־פראט

עתה נציג אלגוריתם התאמת־מחרוזות של קנות', מוריס ופראט (Knuth-Morris-Pratt) שרץ בזמן לינארי. האלגוריתם משיג זמן ריצה של $\Theta(n+m)$ על־ידי כך שהוא נמנע לחלוטין מחישוב פונקציית המעברים δ, ומשתמש לשם התאמת התבנית רק בפונקציית עזר $\pi[1..m]$ המחושבת מראש מן התבנית בזמן $O(m)$. המערך π מאפשר חישוב יעיל (במובן של עלות לשיעורין) של פונקציית המעברים δ "תוך כדי ריצה" בעת הצורך. באופן כללי, עבור כל מצב $q = 0,1,\ldots,m$ וכל תו $a \in \Sigma$, הערך $\pi[q]$ מכיל את המידע הדרוש לחישוב $\delta(q,a)$ שאינו תלוי ב־a. (הערה זו תובהר מיד.) מאחר שהמערך π מכיל רק m תאים, ואילו ל־δ יש $O(m|\Sigma|)$ ערכים, הרי שאנו חוסכים גורם של $|\Sigma|$ בעיבוד המקדים על־ידי כך שאנו מחשבים את π ולא את δ.

פונקציית הרישא של תבנית

פונקציית הרישא של תבנית אוצרת ידע אודות התאמת התבנית להיסטים של עצמה. במידע זה ניתן להשתמש כדי להימנע מבדיקה מיותרת של היסטים חסרי תועלת כפי שעושה האלגוריתם הנאיבי להתאמת מחרוזות, או כדי להימנע מחישוב מראש של δ עבור אוטומט להתאמת מחרוזות.

נתבונן בפעולתו של האלגוריתם הנאיבי להתאמת מחרוזות. באיור 34.9(i) ניתן לראות היסט מסוים s של התבנית $P = \text{ababaca}$ ביחס לטקסט T. בדוגמה זו, $q = 5$ מהתווים הותאמו בהצלחה, אבל התו השישי של התבנית אינו מתאים לתו המקביל בטקסט. המידע ש־q תווים הותאמו בהצלחה מלמד מהם q תווי הטקסט האלה, וידיעתם מאפשרת להסיק מיד שהיסטים מסוימים אינם תקפים. בדוגמה שבאיור, ההיסט $s+1$ בהכרח אינו תקף, שכן במקרה זה, התו הראשון בתבנית, a, יושווה לתו טקסט שעליו ידוע כי הוא מתאים לתו השני בתבנית, שהוא b. לעומת זאת, עבור ההיסט $s+2$ המוצג בחלק (ii) של האיור, מסתדרים שלושת תווי התבנית הראשונים מול תווי טקסט שבהכרח זהים להם. ככלל, מועיל לדעת את התשובה לשאלה הבאה:

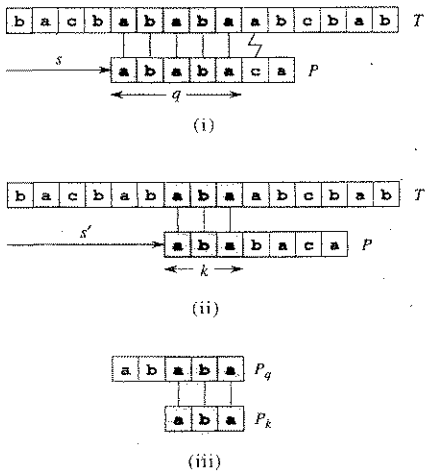
כאשר ידוע שקיימת התאמה בין תווי התבנית $P[1..q]$ לתווי הטקסט $T[s+1..s+q]$, מהו ההיסט הקטן ביותר $s' > s$ שעבורו:

$$P[1..k] = T[s'+1..s'+k] \tag{34.5}$$

כאשר $s'+k = s+q$?

היסט כזה s' הוא ההיסט הראשון הגדול מ־s שאינו בהכרח לא תקף על פי ידיעתנו את $T[s+1..s+q]$. במקרה הטוב ביותר, אנו מקבלים כי $s' = s+q$, וניתן לפסול מיד את ההיסטים $s+1, s+2, \ldots, s+q-1$. בכל מקרה, עבור ההיסט החדש s' אין צורך להשוות את k התווים הראשונים של P עם התווים המקבילים להם ב־T, שכן על פי משוואה (34.5) מובטח לנו שהם שווים.

את המידע הנחוץ ניתן לחשב מראש על־ידי השוואת התבנית לעצמה, כמודגם באיור 34.9(iii). מאחר ש־$T[s'+1..s'+k]$ הוא חלק מהקטע הידוע של הטקסט, הוא מהווה סיפא של המחרוזת P_q. ניתן אפוא לפרש את משוואה (34.5) כשאלה מהו ה־$k > q$ הגדול ביותר שעבורו $P_k \sqsupset P_q$. אזי, $s' = s + (q-k)$ הוא ההיסט הבא שעשוי להיות תקף. מתברר שנוח לאחסן את המספר k של תווים תואמים עבור ההיסט החדש s' במקום לאחסן, למשל, את $s' - s$. ניתן להשתמש במידע זה כדי להחיש את פעולתו של האלגוריתם הנאיבי להתאמת מחרוזות וגם את פעולתו של האוטומט הסופי להתאמת מחרוזות.



איור 34.9 פונקציית הרישא π. **(i)** התבנית $P = \text{ababaca}$ מסתדרת מול הטקסט T כך ש־$q = 5$ התווים הראשונים תואמים. באיור, התווים התואמים מוצללים ומחוברים בקווים אנכיים. **(ii)** ידיעתנו את חמשת התווים התואמים די בה כדי להסיק שההיסט $s+1$ אינו תקף. אבל היסט של $s' = s+2$ קונסיסטנטי עם כל הידוע לנו על הטקסט ולכן עשוי להיות היסט תקף. **(iii)** את המידע המשמש להסקת מסקנות כאלה ניתן לחשב מראש על־ידי השוואת התבנית לעצמה. באיור, אנו רואים שהרישא הארוכה ביותר של P שהיא גם סיפא של P_5 היא P_3. מידע זה מחושב מראש ומאוחסן במערך π, כך ש־$\pi[5] = 3$. אם ידוע שבהיסט s נמצאה התאמה בין q תווים, אזי ההיסט הבא שעשוי להיות תקף הוא $s' = s + (q - \pi[q])$.

את החישובים המקדימים הדרושים ננסח באופן פורמלי כדלקמן. בהינתן תבנית $P[1..m]$, **פונקציית הרישא** (prefix function) עבור התבנית P היא הפונקציה $\pi : \{1,2,\ldots,m\} \to \{0,1,\ldots,m-1\}$ המקיימת:

$$\pi[q] = \max\{k : k < q \text{ וגם } P_k \sqsupset P_q\}$$

כלומר, $\pi[q]$ הוא אורכה של הרישא הארוכה ביותר של P שהיא סיפא ממש של P_q. כדוגמה נוספת, באיור 34.10(i) מופיעים כל הערכים של פונקציית הרישא π עבור התבנית ababababca.

אלגוריתם התאמת־המחרוזות של קנות'־מוריס־פראט נתון בפסאודו־קוד שלהלן כשגרה KMP-Matcher. כפי שנראה בהמשך, המבנה של שגרה זו מבוסס במידה רבה על המבנה של השגרה Finite-Automaton-Matcher. השגרה KMP-Matcher קוראת לשגרת העזר Compute-Prefix-Function לשם חישוב π.

```
KMP-MATCHER(T, P)
 1  n ← length[T]
 2  m ← length[P]
 3  π ← COMPUTE-PREFIX-FUNCTION(P)
 4  q ← 0
 5  for i ← 1 to n
 6      do while q > 0 and P[q + 1] ≠ T[i]
 7             do q ← π[q]
 8         if P[q + 1] = T[i]
 9           then q ← q + 1
10         if q = m
11           then print "Pattern occurs with shift" i − m
12                q ← π[q]
```

```
COMPUTE-PREFIX-FUNCTION(P)
 1  m ← length[P]
 2  π[1] ← 0
 3  k ← 0
 4  for q ← 2 to m
 5      do while k > 0 and P[k + 1] ≠ P[q]
 6             do k ← π[k]
 7         if P[k + 1] = P[q]
 8           then k ← k + 1
 9         π[q] ← k
10  return π
```

נתחיל בניתוח זמני הריצה של שתי שגרות אלה. הוכחת נכונותן מורכבת יותר.

ניתוח זמן הריצה

באמצעות ניתוח לשיעורין (ראה פרק 18) נקבל כי זמן הריצה של COMPUTE-PREFIX-FUNCTION הוא $O(m)$. נייחס פוטנציאל k למצב הנוכחי k של האלגוריתם. על־פי שורה 3, ערכו ההתחלתי של הפוטנציאל הוא 0. בכל פעם ששורה 6 מתבצעת, ערכו של k קטן, שכן $\pi[k] < k$. אולם, מכיוון ש־$\pi[k] \geq 0$ עבור כל k, אזי k לעולם אינו יכול להפוך למספר שלילי. השורה הנוספת היחידה המשפיעה על k היא שורה 8, שבה ערכו של k גדל ב־1 לכל היותר במהלך כל ביצוע של גוף לולאת ה־**for**. מכיוון שעם הכניסה ללולאת ה־**for** מתקיים $k < q$, ומכיוון שערכו של q גדל ב־1 בכל איטרציה של גוף לולאת ה־**for**, אזי $k < q$ מתקיים תמיד. (טיעון זה מצדיק גם את הטענה ש־$\pi[q] < q$, על פי שורה 9.) נוכל לשלם עבור כל ביצוע של גוף לולאת ה־**while** בשורה 6 באמצעות הירידה המתאימה בערכה של פונקציית הפוטנציאל, שכן $\pi[k] < k$. שורה 8 מגדילה את ערכה של פונקציית הפוטנציאל ב־1 לכל היותר, כך שהעלות לשיעורין של גוף הלולאה שבשורות 5-9 היא $O(1)$. מכיוון שמספר האיטרציות של הלולאה החיצונית הוא $O(m)$, ומכיוון שפונקציית הפוטנציאל הסופית גדולה לפחות כמו פונקציית הפוטנציאל ההתחלתית, הרי שזמן הריצה הכולל בפועל של COMPUTE-PREFIX-FUNCTION במקרה הגרוע הוא $O(m)$.

אלגוריתם קנות־מוריס־פראט רץ בזמן $O(m+n)$. הקריאה ל־COMPUTE-PREFIX-FUNCTION צורכת זמן של $O(m)$ כפי שראינו זה עתה, וניתוח לשיעורין דומה, תוך שימוש בערכו של q כפונקציית הפוטנציאל, מראה שיתרת השגרה KMP-MATCHER מתבצעת בזמן $O(n)$.

בהשוואה ל־FINITE-AUTOMATON-MATCHER, על־ידי שימוש ב־π במקום ב־δ, קיצרנו את זמן העיבוד המקדים של התבנית מ־$O(m|\Sigma|)$ ל־$O(m)$, תוך שמירה על חסם של $O(m+n)$ על הזמן שאורכת ההתאמה עצמה.

נכונות החישוב של פונקציית הרישא

נתחיל בלמה חיונית המראה שעל־ידי הפעלה חוזרת של פונקציית הרישא π ניתן למנות את כל הרישות P_k שהן סיפות של רישא נתונה P_q. נסמן:

$$\pi^*[q] = \{q, \pi[q], \pi^2[q], \pi^3[q], \ldots, \pi^t[q]\}$$

כאשר $\pi^i[q]$ מוגדר במונחים של הרכבת פונקציות, כך שמתקיים $\pi^0[q] = q$ ו־$\pi^{i+1}[q] = \pi[\pi^i[q]]$ עבור $i \geq 1$, וכאשר מוסכם שהסדרה $\pi^*[q]$ נעצרת כאשר מגיעים ל־$\pi^t[q] = 0$.

למה 34.5 (למת ההפעלה החוזרת של פונקציית הרישא)
תהי P תבנית באורך m עם פונקציית רישא π. אזי, עבור $q = 1, 2, \ldots, m$ מתקיים $\pi^*[q] = \{k : P_k \succ P_q\}$.

הוכחה תחילה נוכיח כי:

$$i \in \pi^*[q] \Rightarrow P_i \succ P_q \qquad (34.6)$$

אם $i \in \pi^*[q]$, אזי קיים u שעבורו $i = \pi^u[q]$. נוכיח את משוואה (34.6) באינדוקציה על u. עבור $u = 0$, מתקיים $i = q$, והטענה מתקיימת שכן $P_q \succ P_q$. נכונותה של הטענה עבור כל i ב־$\pi^*[q]$ נובעת מן היחס $P_{\pi[i]} \succ P_i$ ומהטרנזיטיביות של היחס $\succ$. לכן, $\pi^*[q] \subseteq \{k : P_k \succ P_q\}$.

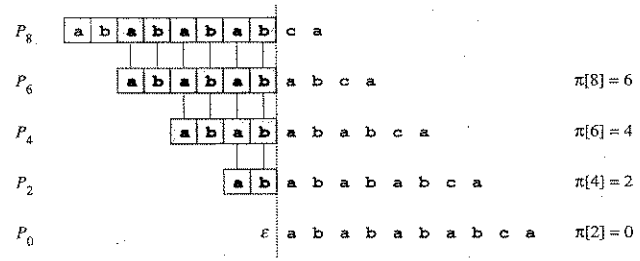
את הכיוון ההפוך, $\{k : P_k \succ P_q\} \subseteq \pi^*[q]$, נוכיח בדרך השלילה. נניח כי הקבוצה $\{k : P_k \succ P_q\} - \pi^*[q]$ מכילה מספר שלם, ונניח כי j הוא הערך השלם הגדול ביותר שהיא מכילה. מכיוון ש־q שייך ל־$\{k : P_k \succ P_q\} \cap \pi^*[q]$, אזי $j < q$, ולכן נסמן ב־j' את השלם הקטן ביותר ב־$\pi^*[q]$ שגדול מ־j. (ניתן לבחור $j' = q$ אם אין ב־$\pi^*[q]$ מספר אחר שגדול מ־j.) מכיוון ש־$j \in \{k : P_k \succ P_q\}$, נובע מכך כי $P_j \succ P_q$; מכיוון ש־$j' \in \pi^*[q]$, נובע מכך כי $P_{j'} \succ P_q$. לכן, על פי למה 34.1, יתר על כן, j הוא הערך הגדול ביותר בעל תכונה זו. לכן, בהכרח מתקיים $j = \pi[j']$, ולפיכך $j \in \pi^*[q]$. סתירה זו מוכיחה את הלמה. ■

איור 34.10 מדגים את הלמה שלעיל.

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	**a**	**b**	**a**	**b**	**a**	**b**	**a**	**b**	**c**	**a**
$\pi[i]$	0	0	1	2	3	4	5	6	0	1

(i)

איור 34.10 הדגמה של למה 34.5 עבור התבנית $P = \text{ababababca}$ ו־$q = 8$. **(i)** הפונקציה π עבור התבנית הנתונה. מכיוון ש־$\pi[8] = 6$, $\pi[6] = 4$, $\pi[4] = 2$, $\pi[2] = 0$ ו־$\pi[1] = 0$, אזי מהפעלה חוזרת של π מקבלים $\pi^*[8] = \{8, 6, 4, 2, 0\}$. **(ii)** אנו מזיזים ימינה את הסרגל המכיל את התבנית P ובודקים מתי איזושהי רישא P_k של P תואמת איזושהי סיפא ממש של P_8; דבר זה קורה עבור $k = 6, 4, 2, 0$. באיור, P מוצגת בשורה הראשונה, והקו האנכי המקווקו משוויטט מיד אחרי P_8. בשורות הבאות מוצגים כל ההיסטים של P שעבורם קיימת התאמה בין איזושהי רישא P_k של P לבין איזושהי סיפא של P_8. תווים תואמים מופיעים באיור כשהם מוצללים ומחוברים זה לזה בקווים אנכיים. מקבלים אפוא כי $\pi^*[q] = \{k : P_k \succ P_q\} = \{8, 6, 4, 2, 0\}$. הלמה טוענת כי $\{k : P_k \succ P_q\} = \{8, 6, 4, 2, 0\}$ עבור כל q.



(ii)

איור 34.10 (המשך)

COMPUTE-PREFIX-FUNCTION מחשבת את $\pi[q]$ לפי הסדר עבור $q = 1,2, \ldots, m$. החישוב של $\pi[1] = 0$ בשורה 2 של COMPUTE-PREFIX-FUNCTION בוודאי נכון, שכן $\pi[q] < q$ עבור כל q. בעזרת הלמה הבאה ומסקנתה נוכיח כי COMPUTE-PREFIX-FUNCTION מחשבת נכונה את $\pi[q]$ עבור כל $q > 1$.

למה 34.6

תהי P תבנית באורך m, ותהי π פונקציית הרישא עבור P. עבור $q = 1,2, \ldots, m$, אם $\pi[q] > 0$, אזי $\pi[q] - 1 \in \pi^*[q-1]$.

הוכחה אם $k = \pi[q] > 0$, אזי $P_k \succ P_q$, ולכן $P_{k-1} \succ P_{q-1}$ (על־ידי השמטת התו האחרון מ־P_k ומ־P_q). לכן, על פי למה 34.5, $k - 1 \in \pi^*[q-1]$. ■

עבור $q = 2, 3, \ldots, m$, נגדיר את תת־קבוצה $E_{q-1} \subseteq \pi^*[q-1]$ על־ידי:

$$E_{q-1} = \{k : k \in \pi^*[q-1] \text{ וגם } P[k+1] = P[q]\}$$

הקבוצה E_{q-1} מורכבת מערכי k שעבורם $P_k \succ P_{q-1}$ (על פי למה 34.5); מכיוון ש־$P[k+1] = P[q]$, אזי עבור ערכים אלה של k מתקיים גם $P_{k+1} \succ P_q$. אינטואיטיבית, E_{q-1} מכילה אותם ערכים $k \in \pi^*[q-1]$ שעבורם ניתן להרחיב את P_k ל־P_{k+1} ולקבל סיפא של P_q.

משקנה 34.7

תהי P תבנית באורך m, ותהי π פונקציית הרישא עבור P. עבור $q = 2, 3, \ldots, m$:

$$\pi[q] = \begin{cases} 0 & \text{אם } E_{q-1} = \phi \\ 1 + \max\{k \in E_{q-1}\} & \text{אם } E_{q-1} \neq \phi \end{cases}$$

הוכחה אם $r = \pi[q]$, אזי $P_r \succ P_q$, ולכן $r \geq 1$ גורר $P[r] = P[q]$. מכאן, על פי למה 34.6, אם $r \geq 1$, נובע מכך כי:

$$r = 1 + \max\{k \in \pi^*[q-1] : P[k+1] = P[q]\}$$

אבל הקבוצה שעליה לוקחים מקסימום אינה אלא E_{q-1}, כך ש־$r = 1 + \max\{k \in E_{q-1}\}$ ו־E_{q-1} אינה ריקה. אם $r = 0$, אזי לא קיים $k \in \pi^*[q-1]$ שעבורו ניתן להרחיב את P_k ל־P_{k+1} ולקבל סיפא של P_q, שכן אז היינו מקבלים $\pi[q] > 0$. לכן, $E_{q-1} = \phi$. ■

עתה נסיים את ההוכחה ש־COMPUTE-PREFIX-FUNCTION מחשבת נכונה את π. בשגרה COMPUTE-PREFIX-FUNCTION, בתחילת כל איטרציה של לולאת ה־**for** שבשורות 4-9 מתקיים $k = \pi[q-1]$. תנאי זה נכפה על־ידי שורות 2 ו־3 כאשר נכנסים ללולאה בפעם הראשונה, והוא נותר נכון בכל האיטרציות הבאות עקב שורה 9. שורות 5-8 מתאימות את k כך שהוא הופך עתה לערך הנכון של $\pi[q]$. הלולאה בשורות 5-6 מבצעת חיפוש בין כל הערכים $k \in \pi^*[q-1]$ עד שנמצא ערך שעבורו $P[k+1] = P[q]$; בנקודה זו, מקבלים כי k הוא הערך הגדול ביותר בקבוצה E_{q-1}, ולכן, על פי מסקנה 34.7, ניתן להציב ב־$\pi[q]$ את $k+1$. אם לא נמצא k כזה, אזי $k = 0$ בשורות 7-9, וב־$\pi[q]$ מוצב 0. בכך נשלמה הוכחת נכונותה של השגרה COMPUTE-PREFIX-FUNCTION.

נכונותו של אלגוריתם קנות׳־מוריס־פראט (KMP)

את השגרה KMP-MATCHER ניתן לראות כמימוש מחדש של FINITE-AUTOMATON-MATCHER. ובמפורט, נוכיח שהקוד בשורות 6-9 של KMP-MATCHER שקול לשורה 4 של השגרה FINITE-AUTOMATON-MATCHER, המציבה ב־q את $\delta(q, T[i])$. אבל במקום להשתמש בערך מאוחסן של $\delta(q, T[i])$, KMP-MATCHER מחשבת ערך זה מ־π בעת הצורך. ברגע שנראה כי KMP-MATCHER מדמה את התנהגותה של FINITE-AUTOMATON-MATCHER, נכונותה של KMP-MATCHER תנבע מנכונותה של FINITE-AUTOMATON-MATCHER (אם כי מיד נראה מדוע דרושה שורה 12 של KMP-MATCHER).

נכונותה של KMP-MATCHER נובעת מן הטענה שמתקיים אחד מהשניים: או $\delta(q, T[i]) = 0$ או $\delta(q, T[i]) - 1 \in \pi^*[q]$. כדי לבדוק טענה זו, יהי $k = \delta(q, T[i])$. אזי, על פי הגדרת δ ו־σ, מתקיים $P_k \succ P_q T[i]$. לכן, או שמתקיים $k = 0$, או שמתקיים גם $k \geq 1$ וגם $P_{k-1} \succ P_q$ על־ידי השמטת התו האחרון מ־P_k וגם מ־$P_q T[i]$ (ובמקרה זה, $k - 1 \in \pi^*[q]$). לכן, או שמתקיים $k = 0$ או שמתקיים $k - 1 \in \pi^*[q]$, ובכך הוכחה הטענה.

נשתמש בטענה זו כדלקמן: נסמן ב־q' את ערכו של q כאשר ההרצה מגיעה לשורה 6. את ההצבה החוזרת $q \leftarrow \pi[q]$ המונה את איברי $\{k : P_k \succ P_{q'}\}$ אנו מצדיקים באמצעות השקילות $\pi^*[q] = \{k : P_k \succ P_q\}$. שורות 6-9 קובעות את הערך של $\delta(q', T[i])$ על־ידי בחינת איברי $q' = \phi(T_{i-1}) = \sigma(T_{i-1})$ בסדר יורד. הקוד משתמש בטענה שלעיל כדי להתחיל עם $\pi^*[q']$ ולבצע שוב ושוב את ההצבה $q \leftarrow \pi[q]$ עד אשר נמצא q כך ש־$q = 0$ או $P[q+1] = T[i]$. במקרה הראשון, $\delta(q', T[i]) = 0$; במקרה השני, q הוא האיבר המקסימלי ב־E_q, כך שעל פי מסקנה 34.7, $\delta(q', T[i]) = q + 1$.

שורה 12 של KMP-MATCHER נחוצה כדי להימנע מאפשרות של פנייה ל־$P[m+1]$ בשורה 6 לאחר מציאת מופע של P. (הטיעון ש־$q = \sigma(T_{i-1})$ באיטרציה הבאה של שורה 6 נותר תקף, על פי הרמז המובא בתרגיל 34.4-6: $\delta(m, a) = \delta(\pi[m], a)$ או, באופן שקול, $\sigma(Pa) = \sigma(P_{\pi[m]}a)$ עבור כל $a \in \Sigma$.) יתרת הטיעון בדבר נכונותו של אלגוריתם קנות׳־מוריס־פראט נובעת מהנכונות של FINITE-AUTOMATON-MATCHER, שכן אנו רואים כי KMP-MATCHER מדמה את פעולתה של FINITE-AUTOMATON-MATCHER.

תרגילים

34.4-1

חשב את פונקציית הרישא π עבור התבנית $P = \texttt{ababbabbababbababbabb}$ מעל האלפבית $\Sigma = \{\texttt{a}, \texttt{b}\}$.

34.4-2

מצא חסם עליון על גודלה של $\pi^*[q]$ כפונקציה של q. הבא דוגמה המראה שהחסם שלך הדוק.

34.4-3

הסבר כיצד מוצאים את מופעיה של תבנית P בטקסט T על־ידי בחינת ערכיה של הפונקציה π עבור המחרוזת PT (המחרוזת באורך $m + n$ המתקבלת משרשור P ו־T).

34.4-4

הראה כיצד ניתן לשפר את KMP-MATCHER על־ידי החלפת המופע של π בשורה 7 (אך לא בשורה 12) ב־π', כאשר π' מוגדרת בצורה רקורסיבית עבור $q = 1, 2, \ldots, m$ על־ידי המשוואה:

$$\pi'[q] = \begin{cases} 0 & \pi[q] = 0 & \text{אם} \\ \pi'[\pi[q]] & \pi[q] \neq 0 \text{ וגם } P[\pi[q]+1] = P[q+1] & \text{אם} \\ \pi[q] & \pi[q] \neq 0 \text{ וגם } P[\pi[q]+1] \neq P[q+1] & \text{אם} \end{cases}$$

הסבר מדוע גרסה זו של האלגוריתם נכונה, והסבר באיזה מובן שינוי זה אכן מהווה שיפור.

34.4-5

כתוב אלגוריתם שזמן ריצתו לינארי, אשר קובע אם טקסט T מהווה רוטציה מעגלית של מחרוזת אחרת T'. לדוגמה, המחרוזות "arc" ו־"car" הן רוטציות מעגליות זו של זו.

*** 34.4-6**

כתוב אלגוריתם יעיל לחישוב פונקציית המעברים δ עבור האוטומט להתאמת מחרוזות המתאים לתבנית נתונה P. האלגוריתם שלך צריך לרוץ בזמן $O(m|\Sigma|)$. (**רמז:** הוכח שאם $q = m$ או $P[q+1] \neq a$, אזי $\delta(q, a) = \delta(\pi[q], a)$.)

* 34.5 אלגוריתם בויאר־מור

כאשר התבנית P ארוכה יחסית והאלפבית Σ גדול למדי, אלגוריתם התאמת־המחרוזות שעשוי להיות היעיל ביותר הוא אלגוריתם של בויאר (Robert S. Boyer) ומור (J Strother Moore).

```
BOYER-MOORE-MATCHER(T, P, Σ)
 1  n ← length[T]
 2  m ← length[P]
 3  λ ← COMPUTE-LAST-OCCURRENCE-FUNCTION(P, m, Σ)
 4  γ ← COMPUTE-GOOD-SUFFIX-FUNCTION(P, m)
 5  s ← 0
 6  while s ≤ n − m
 7      do j ← m
 8         while j > 0 and P[j] = T[s + j]
 9             do i ← j − 1
10         if j = 0
11             then print "Pattern occurs with shift" s
12                  s ← s + γ[0]
13             else s ← s + max(γ[j], j − λ[T[s + j]])
```

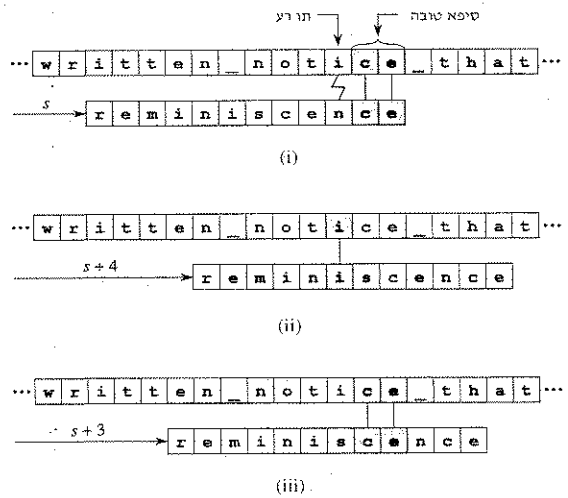
חוץ מ־λ ו־γ המיסתוריים, תכנית זו דומה להפליא לאלגוריתם הנאיבי להתאמת מחרוזות. ואמנם, נניח שנבטל את שורות 3-4 ונחליף את עדכון s בשורות 12-13 בפעולה הפשוטה של הגדלת ערכו של s ב־1:

```
12                  s ← s + 1
13             else s ← s + 1
```

לאחר שינויים אלה, התכנית פועלת בדיוק כמו האלגוריתם הנאיבי להתאמת מחרוזות: לולאת ה־**while** המתחילה בשורה 6 בוחנת בתורו כל אחד מ־$n - m + 1$ ההיסטים האפשריים s, ולולאת ה־**while** המתחילה בשורה 8 בודקת את התנאי $P[1..m] = T[s + 1..s + m]$ על־ידי השוואת $P[j]$ עם $T[s + j]$ עבור $j = m, m - 1, \ldots, 1$. אם הלולאה עוצרת עם $j = 0$, פירוש הדבר שנמצא היסט תקף s, ושורה 11 מדפיסה את ערכו של s. ברמה זו, ישנם רק שני מאפיינים ראויים לציון באלגוריתם בויאר־מור: הוא משווה את התבנית לטקסט *מימין לשמאל* והוא מגדיל את ערכו של s בשורות 12-13 בערך שאינו בהכרח 1.

אלגוריתם בויאר־מור מפעיל שתי היוריסטיקות המאפשרות לו להימנע מביצוע חלק ניכר מן העבודה שביצעו האלגוריתמים הקודמים שלנו להתאמת מחרוזות. היוריסטיקות אלה כה יעילות, עד שלעתים קרובות הן מאפשרות לאלגוריתם לדלג לחלוטין על בחינתם של תווי טקסט רבים. היוריסטיקות אלה, הידועות בשמות "היוריסטיקת התו־הרע" ו"היוריסטיקת הסיפא הטובה", מודגמות באיור 34.11. ניתן לראותן כהיוריסטיקות שפועלות באופן בלתי־תלוי ובמקביל. כאשר מתגלה חוסר התאמה, כל אחת מהן מציעה ערך שבו ניתן להגדיל את s ללא חשש להחמצת היסט תקף. אלגוריתם בויאר־מור בוחר בערך הגדול יותר ומגדיל את s בערך זה.

כאשר ההרצה מגיעה לשורה 13 לאחר שהתגלה חוסר התאמה, היוריסטיקת התו־הרע מציעה להגדיל את s ב־$j - \lambda[T[s + j]]$, והיוריסטיקת הסיפא הטובה מציעה להגדיל את s ב־$\gamma[j]$.



איור 34.11 הדגמת ההיוריסטיקות באלגוריתם בויאר־מור. **(i)** בדיקת ההתאמה בין התבנית `reminiscence` לטקסט על־ידי השוואת תווים מימין לשמאל. הזיהוי s אינו תקף, אף כי "סיפא טובה" `ce` בתבנית מתאימה לתווים המקבילים בטקסט (באיור מוצללים תווים תואמים), התגלה בטקסט "תו רע" `i` שלא התאים לתו המקביל `n` בתבנית. **(ii)** היוריסטיקת התו־הרע מציעה להזיז את התבנית ימינה, אם אפשר, במספר מקומות כזה שיבטיח התאמה בין התו הרע בטקסט לבין המופע הימני ביותר של התו הרע בתבנית. בדוגמה זו, הזזת התבנית 4 מקומות ימינה מציבה את התו הרע `i` שבטקסט מול ה־`i` הימני ביותר בתבנית, במיקום 6. אם התו הרע אינו מופיע בתבנית, אזי ניתן להזיז את התבנית כולה אל מעבר לתו־הרע בטקסט. אם המופע הימני ביותר של התו הרע בתבנית נמצא מימין למיקומו הנוכחי של התו הרע, אזי היוריסטיקה זו אינה מציעה שום הצעה. **(iii)** בהיוריסטיקת הסיפא הטובה מזיזים את התבנית ימינה במספר המקומות הקטן ביותר המבטיח התאמה בין תווי הסיפא הטובה שנמצאה בטקסט, `ce`, לבין תווי התבנית שיוצבו מולם. בדוגמה זו, הזזת התבנית 3 מקומות ימינה תקיים תנאי זה. מכיוון שהיוריסטיקת הסיפא הטובה מציעה הזזה ב־3 מקומות, והזזה זו קטנה מההזזה של 4 מקומות שהציעה היוריסטיקת התו־הרע, אלגוריתם בויאר־מור מגדיל את ההיסט ב־4.

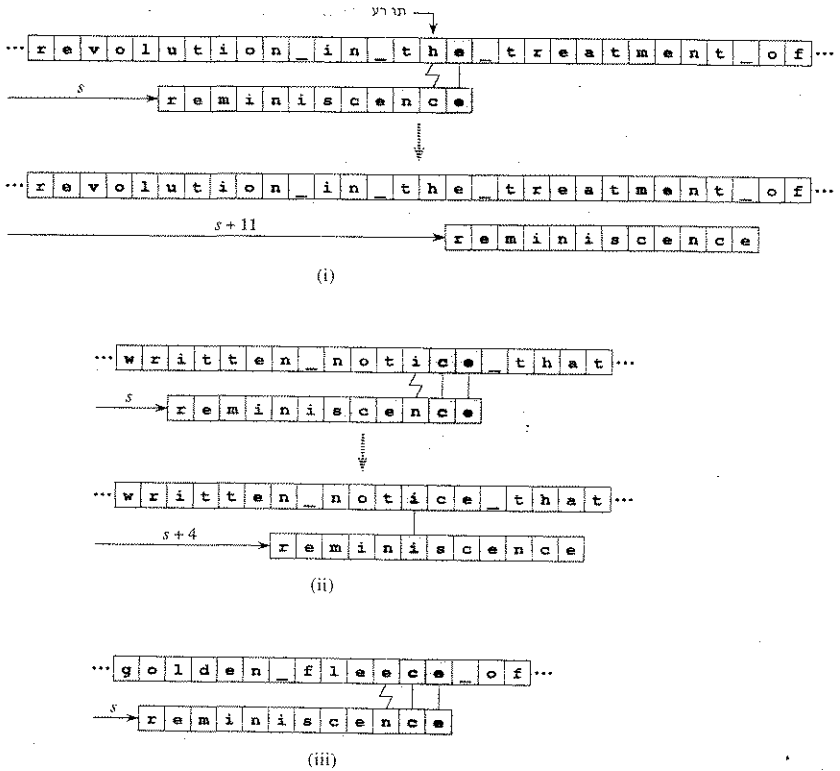
היוריסטיקת התו־הרע

כאשר מתגלה חוסר התאמה, היוריסטיקת התו־הרע מנצלת את המידע על מיקומו בתבנית של תו הטקסט הרע $T[s+j]$ (אם הוא מופיע בה בכלל) כדי להציע היסט חדש. במקרה הטוב ביותר, חוסר ההתאמה מתגלה בהשוואה הראשונה (כלומר, $P[m] \neq T[s+m]$) והתו הרע $T[s+m]$ אינו מופיע כלל בתבנית. (חשבו על חיפוש התבנית $a^m b$ במחרוזת הטקסט b^n.) במקרה זה, ניתן להגדיל ב־m את ההיסט s, שכן כל היסט קטן מ־$s+m$ יציב אחד מתווי התבנית מול התו הרע ויגרום בכך לחוסר התאמה. אם המצב הטוב ביותר מתרחש פעם אחר פעם, אזי אלגוריתם בויאר־מור בוחן רק $1/m$ מתווי הטקסט, שכן כל בחינה של תו טקסט מגלה חוסר התאמה הגורם להגדלת s ב־m. התנהגות זו של האלגוריתם במקרה הטוב ביותר מדגימה את כוחה של ההתאמה מימין לשמאל, במקום משמאל לימין.

ככלל, **היוריסטיקת התו־הרע** פועלת כדלקמן. נניח שגילינו זה עתה חוסר התאמה: $P[j] \neq T[s+j]$ עבור איזשהו j, כאשר $1 \leq j \leq m$. אזי אנו מציבים ב־k את האינדקס הגדול ביותר בתחום $1 \leq k \leq m$ שעבורו $T[s+j] = P[k]$, אם קיים k כזה. אחרת, אנו מציבים

$k = 0$. אנו טוענים שניתן להגדיל בבטחה את s ב־$j-k$. כדי להוכיח טענה זו, עלינו לבחון שלושה מקרים, כמודגם באיור 34.12.

- $k = 0$: כפי שניתן לראות באיור 34.12(i), התו הרע $T[s+j]$ אינו מופיע כלל בתבנית, ולכן ניתן להגדיל את s ב־j ללא חשש להחמצת היסטים תקפים.
- $k < j$: כפי שניתן לראות באיור 34.12(ii), המופע הימני ביותר של התו הרע בתבנית הוא משמאל למיקום j, כך ש־$j-k > 0$ ויש להזיז את התבנית $j-k$ מקומות ימינה עד שהתו הרע מן הטקסט יתאים לתו כלשהו בתבנית. לכן, נוכל להגדיל בבטחה את s ב־$j-k$, ללא חשש להחמצת היסטים תקפים.
- $k > j$: כפי שניתן לראות באיור 34.12(iii), $j-k < 0$, ולכן היוריסטיקת התו־הרע מציעה בעצם להקטין את s. אלגוריתם בויאר־מור יתעלם מהמלצה זו, שכן היוריסטיקת הסיפא־הטובה תציע היסט ימינה בכל מקרה.



איור 34.12 שלושת המקרים של היוריסטיקת התו־הרע. **(i)** התו הרע `h` אינו מופיע כלל בתבנית, ולכן ניתן לקדם את התבנית עד מעבר לתו זה, דהיינו, $j = 11$ תווים ימינה. **(ii)** המופע הימני ביותר של התו הרע בתבנית הוא במיקום $k < j$, ולכן ניתן לקדם את התבנית ב־$j-k$ תווים. מכיוון ש־$j = 10$ ו־$k = 6$ עבור התו הרע `i`, ניתן לקדם את התבנית ב־4 מקומות עד להתאמה בין ה־`i` שבטקסט ל־`i` הימני ביותר שבתבנית. **(iii)** המופע הימני ביותר של התו הרע בתבנית הוא במיקום $k > j$. בדוגמה זו, $j = 10$ ו־$k = 12$ עבור התו הרע `e`. היוריסטיקת התו הרע מציעה היסט שלילי, הצעה שהאלגוריתם מתעלם ממנה.

התכנית הפשוטה הבאה מגדירה את $\lambda[a]$, עבור כל $a \in \Sigma$, כאינדקס של המיקום הימני ביותר בתבנית שבו מופיע התו a. אם a אינו מופיע בתבנית, אזי $\lambda[a] = 0$. אנו מכנים את λ **פונקציית המופע האחרון** (last-occurrence function) של התבנית. בהינתן הגדרה זו, הביטוי $j - \lambda[T[s+j]]$, המופיע בשורה 13 בשגרה BOYER-MOORE-MATCHER, מממש את היוריסטיקת התו־הרע. (מאחר ש־$j - \lambda[T[s+j]]$ שלילי אם המופע הימני ביותר של התו הרע $T[s+j]$ בתבנית נמצא מימין למיקום j, אנו סומכים על כך שערכו החיובי של $\gamma[j]$, המוצע על־ידי היוריסטיקת הסיפא־הטובה, יבטיח שהאלגוריתם אכן יתקדם בכל צעד.)

```
COMPUTE-LAST-OCCURRENCE-FUNCTION(P, m, Σ)
1  for each character a ∈ Σ
2      do λ[a] ← 0
3  for j ← 1 to m
4      do λ[P[j]] ← j
5  return λ
```

זמן הריצה של השגרה COMPUTE-LAST-OCCURRENCE-FUNCTION הוא $O(|\Sigma| + m)$.

היוריסטיקת הסיפא־הטובה

הבה נגדיר את היחס $Q \sim R$ (קרי: "Q דומה ל־R") עבור המחרוזות Q ו־R כיחס שפירושו $Q \succ R$ או $Q = R$. אם שתי מחרוזות דומות, אזי ניתן להציבן זו מול זו כך שהתווים הימניים ביותר שלהן יתאימו, ולא יהיה אף זוג תווים מקבילים שאינם תואמים. היחס "$\sim$" הוא סימטרי: $Q \sim R$ אם ורק אם $R \sim Q$. כמו כן, כתוצאה מלמה 34.1, מתקיים:

$$\text{אם } S \succ R \text{ וגם } Q \succ R \text{ אזי } Q \sim S \tag{34.7}$$

אם אנו מוצאים כי $P[j] \neq T[s+j]$, עבור $j < m$, אזי **היוריסטיקת הסיפא־הטובה** טוענת כי נוכל לקדם בבטחה את s במספר המקומות:

$$\gamma[j] = m - \max\{k : 0 \le k < m \text{ וגם } P[j+1..m] \sim P_k\}$$

כלומר, $\gamma[j]$ הוא מספר המקומות הקטן ביותר שבו ניתן לקדם את s בלי שייווצר חוסר התאמה בין תווי "הסיפא הטובה" $T[s+j+1..s+m]$ לבין התווים המקבילים לה במקומה החדש של התבנית. הפונקציה γ מוגדרת היטב עבור כל j, שכן $P[j+1..m] \sim P_0$ עבור כל j: המחרוזת הריקה דומה לכל המחרוזות. אנו מכנים את γ **פונקציית הסיפא־הטובה** של התבנית P.

עתה נראה כיצד מחשבים את ערכיה של פונקציית הסיפא־הטובה γ. תחילה נשים לב כי $\gamma[j] \le m - \pi[m]$ עבור כל j: אם $w = \pi[m]$, אזי $P_w \succ P$, על פי הגדרת π. יתר על כן, מאחר ש־$P[j+1..m] \succ P$ עבור כל j, הרי שעל פי משוואה (34.7) נקבל כי $P_w \sim P[j+1..m]$. לכן, $\gamma[j] \le m - \pi[m]$ עבור כל j.

עתה נוכל לנסח מחדש את הגדרת γ באופן הבא:

$$\gamma[j] = m - \max\{k : \pi[m] \le k < m \text{ וגם } P[j+1..m] \sim P_k\}$$

התנאי $P[j+1..m] \sim P_k$ מתקיים אם $P[j+1..m] \succ P_k$ או $P[j+1..m] = P_k$. אבל האפשרות האחרונה גוררת כי $P_k \succ P$, ולכן, על פי הגדרת π, כי $k \le \pi[m]$. אפשרות אחרונה זו אינה יכולה להקטין את ערכו של $\gamma[j]$ מתחת ל־$m - \pi[m]$. נוכל אפוא לשוב ולנסח מחדש את הגדרת γ באופן הבא:

$$\gamma[j] = m - \max(\{\pi[m]\} \cup \{k : \pi[m] < k < m \text{ וגם } P[j+1..m] \succ P_k\})$$

(הקבוצה השנייה עשויה להיות ריקה.) כדאי לשים לב שהגדרה זו גוררת כי $\gamma[j] > 0$ עבור כל $j = 1, 2, \ldots, m$, דבר המבטיח כי אלגוריתם בויאר־מור אכן יתקדם.

כדי לפשט עוד יותר את הגדרת γ, נגדיר את P' כתבנית ההפוכה לתבנית P ואת π' כפונקציית הרישא המתאימה. כלומר, $P'[i] = P[m-i+1]$ עבור $i = 1, 2, \ldots, m$, ו־$\pi'[t]$ הוא ה־u הגדול ביותר המקיים $u < t$ וגם $P'_u \succ P'_t$.

אם k הוא הערך הגדול ביותר שעבורו $P[j+1..m] \succ P_k$, אזי אנו טוענים כי:

$$\pi'[l] = m - j \tag{34.8}$$

כאשר $l = (m-k) + (m-j)$. כדי להיווכח שטענה זו מוגדרת היטב, נשים לב כי $P[j+1..m] \succ P_k$ גורר $j < m$ ו־$k \le m$, ולכן $m - j \le k$. כמו כן, $l \ge m$. אנו נוכיח את הטענה באופן הבא: מאחר ש־$P[j+1..m] \succ P_k$, אזי $P'_{m-j} \succ P'_l$. לכן, $\pi'[l] \ge m - j$. עתה, נניח כי $p > m - j$, כאשר $p = \pi'[l]$. אזי, על פי הגדרת π', מקבלים כי $P'[1..p] = P'[l-p+1..l]$ או באופן שקול, $P'_p \succ P'_l$. נכתוב מחדש משוואה זו במונחי P במקום במונחי P', ונקבל $P[m-p+1..m] = P[m-l+1..m-l+p]$. נציב $l = 2m - k - j$ ונקבל $P[m-p+1..m] = P[k-m+j+1..k-m+j+p]$, ומכך נובע כי $P[m-p+1..m] \succ P_{k-m+j+p}$. מאחר ש־$p > m-j$, נובע מכך כי $j+1 > m-p+1$, ולכן $P[j+1..m] \succ P_{k-m+j+p}$, ומכך נובע כי $P[j+1..m] \succ P[m-p+1..m]$ על פי הטרנזיטיביות של $\succ$. לבסוף, מכיוון ש־$p > m-j$, אזי $k' > k$, כאשר $k' = k - m + j + p$, בסתירה לבחירת k כערך הגדול ביותר האפשרי שעבורו $P[j+1..m] \succ P_k$. סתירה זו פירושה שלא ייתכן כי $p > m-j$, לכן, $p = m-j$, ובכך הוכחה הטענה (34.8).

אם נשתמש במשוואה (34.8), ונשים לב כי $\pi'[l] = m - j$ גורר $j = m - \pi'[l]$ וגם $k = m - l + \pi'[l]$, נוכל שוב לכתוב מחדש את הגדרת γ בצורה:

$$\begin{aligned}\gamma[j] &= m - \max(\{\pi[m]\} \\ &\qquad \cup \{m - l + \pi'[l] : 1 \le l \le m \text{ וגם } j = m - \pi'[l]\}) \\ &= \min(\{m - \pi[m]\} \\ &\qquad \cup \{l - \pi'[l] : 1 \le l \le m \text{ וגם } j = m - \pi'[l]\})\end{aligned} \tag{34.9}$$

שוב, הקבוצה השנייה עשויה להיות ריקה.

עתה אנו ערוכים לבחינת השגרה המחשבת את γ.

```
COMPUTE-GOOD-SUFFIX-FUNCTION(P, m)
1  π ← COMPUTE-PREFIX-FUNCTION(P)
2  P' ← reverse(P)
3  π' ← COMPUTE-PREFIX-FUNCTION(P')
4  for j ← 0 to m
5      do γ[j] ← m − π[m]
6  for l ← 1 to m
7      do j ← m − π'[l]
8         if γ[j] > l − π'[l]
9            then γ[j] ← l − π'[l]
10 return γ
```

השגרה COMPUTE-GOOD-SUFFIX-FUNCTION היא מימוש ישיר של משוואה (34.9). זמן הריצה שלה הוא $O(m)$.

זמן הריצה של אלגוריתם בויאר־מור במקרה הגרוע הוא בבירור $O((n-m+1)m+|\Sigma|)$, שכן השגרה COMPUTE-LAST-OCCURRENCE-FUNCTION מתבצעת בזמן $O(m+|\Sigma|)$, השגרה COMPUTE-GOOD-SUFFIX-FUNCTION מתבצעת בזמן $O(m)$, ואלגוריתם בויאר־מור (כמו אלגוריתם רבין־קארפ) צורך זמן של $O(m)$ לבדיקת תקפותו של כל היסט תקף s. אבל באופן מעשי, אלגוריתם בויאר־מור הוא לעתים קרובות האלגוריתם המועדף.

תרגילים

34.5-1

חשב את הפונקציות λ ו־γ עבור התבנית $P = 0101101201$ והאלפבית $\Sigma = \{0, 1, 2\}$.

34.5-2

הבא דוגמאות המראות שעל־ידי צירוף של היוריסטיקת התו־הרע והיוריסטיקת הסיפא־הטובה, ביצועיו של אלגוריתם בויאר־מור טובים בהרבה מאלה שהיו מתקבלים אילו השתמש רק בהיוריסטיקת הסיפא־הטובה.

34.5-3 *

ביישומים מעשיים משפרים לעתים קרובות את אלגוריתם בויאר־מור חבטיסי באמצעות החלפת הפונקציה γ בפונקציה γ' המוגדרת על־ידי:

$$\gamma'[j] = m - \max\{k : 0 \le k < m \text{ וגם } P[j+1..m] \sim P_k \text{ וגם } (k-m+j > 0 \Rightarrow P[j] \ne P[k-m+j])\}$$

בנוסף לכך שהפונקציה γ' מבטיחה חוסר התאמה בין התווים בסיפא הטובה לבין התווים המקבילים בהיסט החדש, היא גם מבטיחה שאותו תו בתבנית לא ימוקם להשוואה מול התו הרע בטקסט. הראה כיצד לחשב ביעילות את הפונקציה γ'.

בעיות

34-1 התאמת מחרוזות המבוססת על גורמי חזרה

נסמן ב־y^i את שרשור המחרוזת y לעצמה i פעמים. למשל, $(\mathtt{ab})^3 = \mathtt{ababab}$. אנו אומרים שמחרוזת $x \in \Sigma^*$ היא בעלת **גורם חזרה** r (repetition factor) אם קיימים $y \in \Sigma^*$ ו־$r > 0$ שעבורם $x = y^r$. נסמן ב־$\rho(x)$ את ה־r הגדול ביותר שהוא גורם חזרה של x.

א. כתוב אלגוריתם יעיל המקבל כקלט תבנית $P[1..m]$ ומחשב את $\rho(P_i)$ עבור $i = 1, 2, \ldots, m$. מהו זמן הריצה של האלגוריתם שלך?

ב. בהינתן תבנית כלשהי $P[1..m]$, נגדיר את $\rho^*(P)$ על־ידי $\max_{1 \le i \le m} \rho(P_i)$. הוכח שאם התבנית P נבחרת באקראי מבין קבוצת כל המחרוזות הבינריות באורך m, אזי תוחלת ערכו של $\rho^*(P)$ היא $O(1)$.

ג. הראה שאלגוריתם התאמת־המחרוזות שלהלן מוצא באופן נכון את כל מופעיה של תבנית P בטקסט $T[1..n]$ בזמן $O(\rho^*(P)n + m)$.


```
REPETITION-MATCHER(P, T)
 1  m ← length[P]
 2  n ← length[T]
 3  k ← 1 + ρ*(P)
 4  q ← 0
 5  s ← 0
 6  while s ≤ n − m
 7      do if T[s + q + 1] = P[q + 1]
 8          then q ← q + 1
 9               if q = m
10                  then print "Pattern occurs with shift" s
11          if q = m or T[s + q + 1] ≠ P[q + 1]
12            then s ← s + max(1, ⌈q/k⌉)
13                 q ← 0
```

זהו אלגוריתם של גליל וסייפראס (Galil & Seiferas). על־ידי הרחבה ניכרת של רעיונות אלה הם משיגים אלגוריתם התאמת־מחרוזות שרץ בזמן לינארי ומשתמש רק במקום אחסון של $O(1)$ בנוסף למקום הדרוש לאחסון P ו־T.

34-2 התאמת מחרוזות מקבילית

נבחן את בעיית התאמת המחרוזות על מחשב מקבילי. נניח כי עבור תבנית נתונה בנינו אוטומט התאמת־מחרוזות M בעל קבוצת מצבים Q. תהי ϕ פונקציית המצבים הסופיים של M. נניח שטקסט הקלט שלנו הוא $T[1..n]$. ברצוננו לחשב את $\phi(T_i)$ עבור $i = 1, 2, \ldots, n$; כלומר, ברצוננו לחשב את המצב הסופי עבור כל רישא. האסטרטגיה שלנו היא שימוש בחישוב הרישות המקבילי המתואר בסעיף 30.1.2.

עבור כל מחרוזת קלט x, נגדיר את הפונקציה $\delta_x : Q \to Q$ כך שאם M מתחיל במצב q וקורא את הקלט x, אזי M מסיים במצב $\delta_x(q)$.

א. הוכח כי $\delta_y \circ \delta_x = \delta_{xy}$, כאשר הסימן $\circ$ מציין הרכבת פונקציות:

$$(\delta_y \circ \delta_x)(q) = \delta_y(\delta_x(q))$$

ב. הראה כי $\circ$ היא פעולה אסוציאטיבית.

ג. הראה כי בהינתן ייצוג על־ידי טבלאות של δ_x ו־δ_y, ניתן לחשב את δ_{xy} על CREW PRAM בזמן $O(1)$. נתח כמה מעבדים דרושים במונחי $|Q|$.

ד. הוכח כי $\phi(T_i) = \delta_{T_i}(q_0)$, כאשר q_0 הוא המצב ההתחלתי של M.

ה. הראה כיצד ניתן למצוא את כל המופעים של תבנית נתונה בטקסט באורך n בזמן $O(\lg n)$ על CREW PRAM. הנח שהתבנית נתונה על־ידי אוטומט התאמת־המחרוזות המתאים לה.

הערות ביבליוגרפיות לפרק 34

הקשר בין התאמת מחרוזות לבין אוטומטים סופיים נדון אצל Aho, Hopcroft & Ullman [4]. אלגוריתם קנותי־מוריס־פראט [125] פותח באופן בלתי תלוי על־ידי קנותי (Knuth) ופראט (Pratt) ועל־ידי מוריס (Morris); הם פרסמו במשותף את עבודתם. אלגוריתם רבין־קארפ הוצע על ידי Rabin & Karp [117] ואלגוריתם בויאר־מור הוא של Boyer & Moore [32]. Galil & Seiferas [78] מציגים אלגוריתם התאמת־מחרוזות דטרמיניסטי מעניין שרץ בזמן לינארי ומשתמש רק במקום אחסון של $O(1)$ בנוסף על מקום האחסון הדרוש עבור התבנית והטקסט.

35 גיאומטריה חישובית

גיאומטריה חישובית היא הענף במדעי המחשב החוקר אלגוריתמים לפתרון בעיות גיאומטריות. בתחומי ההנדסה והמתמטיקה המודרניות, הגיאומטריה החישובית מיושמת, בין השאר, בגרפיקה ממוחשבת, רובוטיקה, תכנון שבבי VLSI, תיב"ם וסטטיסטיקה. הקלט לבעיה בגיאומטריה חישובית הוא בדרך כלל תיאור של קבוצת עצמים גיאומטריים, כגון קבוצת נקודות, קבוצת קטעים או הקדקודים של מצולע בסדר מסוים. הפלט הוא תכופות תשובה לשאילתה על העצמים, כגון, האם ישנם ישרים שחותכים זה את זה, או לפעמים עצם גיאומטרי חדש, כגון הקמור של קבוצת הנקודות (המצולע הקמור הקטן ביותר המקיף את הנקודות).

בפרק זה נתבונן בכמה אלגוריתמים לפתרון בעיות גיאומטריות דו־ממדיות, דהיינו, בעיות גיאומטריות במישור. כל עצם קלט מיוצג כקבוצת נקודות $\{p_i\}$, כאשר $p_i = (x_i, y_i)$ ו־$x_i, y_i \in \mathbf{R}$. לדוגמה, מצולע P בן n קדקודים מיוצג על־ידי סדרה $\langle p_0, p_1, \ldots, p_{n-1} \rangle$ של קדקודיו בסדר הופעתם על היקף המצולע. גיאומטריה חישובית ניתן לבצע גם במרחב התלת־ממדי, ואף במרחבים מממד גבוה יותר, אולם בעיות כאלה ופתרונותיהן עלולים להיות קשים מאוד להמחשה. אבל גם בשני ממדים נוכל לראות מדגם טוב של טכניקות בגיאומטריה חישובית.

בסעיף 35.1 נראה כיצד ניתן לענות ביעילות ובדייקנות על שאלות פשוטות אודות קטעי ישרים: בהינתן שני קטעים היוצאים מאותה נקודה, האם קטע אחד נמצא בכיוון השעון או בכיוון המנוגד ביחס לקטע האחר; לאיזה כיוון אנו פונים כאשר אנו עוברים מקטע ישר אחד לקטע ישר סמוך לו; או, האם שני קטעים ישרים נחתכים. בסעיף 35.2 נציג טכניקה המכונה "סריקה" שאנו משתמשים בה לפיתוח אלגוריתם שזמן ריצתו $O(n \lg n)$ אשר, בהינתן קבוצה בת n קטעים ישרים, קובע אם היא מכילה קטעים נחתכים. בסעיף 35.3 נציג שני אלגוריתמי "סריקה סיבובית" המחשבים את הקמור של קבוצה בת n נקודות (כלומר, את המצולע הקמור הקטן ביותר המקיף את הנקודות): הסריקה של גרהאם, שרצה בזמן $O(n \lg n)$, והצעדה של ג'ארביס, שרצה בזמן $O(nh)$, כאשר h הוא מספר הקדקודים בקמור. לבסוף, בסעיף 35.4 נציג אלגוריתם הפרד־ומשול שזמן ריצתו $O(n \lg n)$, אשר מוצא את שתי הנקודות הקרובות ביותר מבין n נקודות נתונות במישור.

35.1 תכונות של קטעי ישרים

כמה מן האלגוריתמים בגיאומטריה חישובית שיוצגו בפרק זה ידרשו תשובות לשאלות אודות תכונות של קטעי ישרים. **צירוף קמור** (convex combination) של שתי נקודות שונות $p_1 = (x_1, y_1)$ ו־$p_2 = (x_2, y_2)$ הוא כל נקודה $p_3 = (x_3, y_3)$ כך שעבור איזשהו α בתחום $0 \le \alpha \le 1$ מתקיים $x_3 = \alpha x_1 + (1 - \alpha) x_2$ ו־$y_3 = \alpha y_1 + (1 - \alpha) y_2$. נכתוב זאת גם בצורה

$p_3 = \alpha p_1 + (1 - \alpha) p_2$. אינטואיטיבית, p_3 היא כל נקודה על הישר העובר דרך p_1 ו־p_2 הנמצאת בין שתי הנקודות או על אחת מהן. בהינתן שתי נקודות שונות p_1 ו־p_2, **הקטע הישר** (line segment) $\overline{p_1 p_2}$ הוא קבוצת הצירופים הקמורים של p_1 ו־p_2. אנו מכנים את p_1 ו־p_2 **נקודות הקצה** (endpoints) של הקטע $\overline{p_1 p_2}$. לפעמים יש חשיבות לסדר של p_1 ו־p_2, ואז מדברים על **הקטע המכוון** (directed segment) $\overrightarrow{p_1 p_2}$. אם p_1 היא **נקודת הראשית** (origin) $(0, 0)$, אזי ניתן להתייחס אל הקטע המכוון $\overrightarrow{p_1 p_2}$ כאל **הווקטור** p_2.

בסעיף זה נחקור את השאלות הבאות:

1. בהינתן שני קטעים מכוונים $\overrightarrow{p_0 p_1}$ ו־$\overrightarrow{p_0 p_2}$, האם $\overrightarrow{p_0 p_1}$ נמצא בכיוון השעון מ־$\overrightarrow{p_0 p_2}$, ביחס לנקודת הקצה המשותפת שלהם p_0?
2. בהינתן שני קטעים $\overline{p_1 p_2}$ ו־$\overline{p_2 p_3}$, אם נעים לאורכו של $\overline{p_1 p_2}$ ואחר־כך לאורכו של $\overline{p_2 p_3}$, האם בנקודה p_2 פונים שמאלה?
3. האם הקטעים $\overline{p_1 p_2}$ ו־$\overline{p_3 p_4}$ נחתכים?

אין כל מגבלות על הנקודות הנתונות.

על כל אחת מהשאלות הללו ניתן לענות בזמן $O(1)$, ואין בכך משום הפתעה, שכן גודל הקלט לכל אחת מהשאלות הוא $O(1)$. יתירה מזו, השיטות שלנו ישתמשו רק בפעולות חיבור, חיסור כפל והשוואה, איננו זקוקים לחילוק או לפונקציות טריגונומטריות, פעולות יקרות מבחינה חישובית ומועדות לבעיות בשל טעויות עיגול. לדוגמה, השיטה "הישירה" כדי לקבוע אם שני קטעים נחתכים — לחשב את משוואת הישר מהצורה $y = mx + b$ עבור כל קטע (m הוא השיפוע ו־b היא נקודת החיתוך עם ציר ה־y), למצוא את נקודת החיתוך של הישרים, ולבדוק אם נקודה זו נמצאת על שני הקטעים — משתמשת בחילוק למציאת נקודת החיתוך. כאשר הקטעים כמעט מקבילים, שיטה זו רגישה ביותר למידת הדיוק בביצוע פעולת החילוק על מחשבים ממשיים. השיטה שנביא בסעיף זה, הנמנעת מחילוק, מדויקת בהרבה.

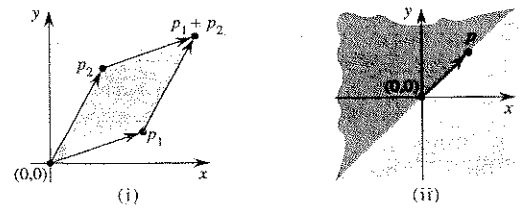
מכפלות וקטוריות

תישוב מכפלות וקטוריות הוא לב לבן של השיטות שלנו לטיפול בקטעים. נתבונן בווקטורים p_1 ו־p_2 המוצגים באיור 35.1(i). את **המכפלה הווקטורית** (cross product) $p_1 \times p_2$ ניתן לפרש כשטח המקבילית (עם הסימן המתאים) הנוצרת על־ידי הנקודות $(0, 0)$, p_1, p_2 ו־$p_1 + p_2 = (x_1 + x_2, y_1 + y_2)$. הגדרה שקולה אך שימושית יותר מגדירה את המכפלה הווקטורית כדטרמיננטה של מטריצה[1]:

$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 .$$

אם המכפלה הווקטורית $p_1 \times p_2$ חיובית, אזי p_1 נמצא בכיוון השעון מ־p_2 ביחס לראשית $(0, 0)$; אם היא שלילית, אזי p_1 נמצא במנוגד לכיוון השעון מ־p_2. באיור 35.1(ii) מופיעים האזורים הנמצאים בכיוון השעון ובמנוגד לכיוון השעון ביחס לווקטור p. מצב גבולי מתעורר כאשר המכפלה הווקטורית שווה לאפס; במקרה זה, הווקטורים נמצאים על ישר אחד (collinear) והם מכוונים באותו כיוון או בכיוונים מנוגדים.

1. למעשה, מכפלה וקטורית היא מושג תלת־ממדי. זהו וקטור הניצב הן ל־p_1 והן ל־p_2 על פי "כלל היד הימנית" וגודלו $x_1 y_2 - x_2 y_1$. אבל בפרק זה נוח יהיה להתייחס אל המכפלה הווקטורית פשוט כאל הערך $x_1 y_2 - x_2 y_1$.



איור 35.1 **(i)** המכפלה הווקטורית של הווקטורים p_1 ו־p_2 שווה לשטח המקבילית (עם הסימן המתאים). **(ii)** האזור הבהיר מכיל וקטורים הנמצאים בכיוון השעון מ־p. האזור הכהה יותר מכיל וקטורים הנמצאים במנגד לכיוון השעון מ־p.

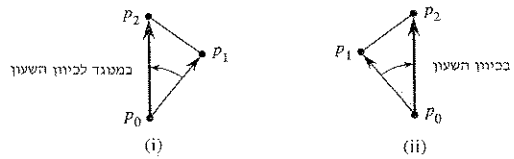
כדי לקבוע אם קטע מכוון $\overrightarrow{p_0p_1}$ נמצא בכיוון השעון מקטע מכוון $\overrightarrow{p_0p_2}$ ביחס לנקודת הקצה המשותפת שלהם p_0, עלינו פשוט לבצע העתקה כך ש־p_0 תהיה נקודת הראשית. כלומר, יהי $p_1 - p_0$ הווקטור $p_1' = (x_1', y_1')$, כאשר $x_1' = x_1 - x_0$ ו־$y_1' = y_1 - y_0$, ונגדיר בצורה דומה את $p_2 - p_0$. לאחר מכן נחשב את המכפלה הווקטורית:

$$(p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$$

אם מכפלה וקטורית זו היא חיובית, אזי $\overrightarrow{p_0p_1}$ נמצא בכיוון השעון מ־$\overrightarrow{p_0p_2}$, אם היא שלילית, אזי הוא נמצא במנגד לכיוון השעון.

כיצד קובעים אם קטעים עוקבים פונים ימינה או שמאלה

השאלה הבאה שלנו היא זו: בהינתן שני קטעים עוקבים $\overline{p_0p_1}$ ו־$\overline{p_1p_2}$, האם הם פונים ימינה או שמאלה בנקודה p_1. באופן שקול, אנו מחפשים שיטה שבה ניתן לקבוע לאיזה כיוון פונה זווית נתונה $\angle p_0p_1p_2$. מכפלות וקטוריות מאפשרות לנו לענות על שאלה זו בלי לחשב את הזווית. כפי שמראה איור 35.2, אנו פשוט בודקים אם הקטע המכוון $\overrightarrow{p_0p_2}$ נמצא בכיוון השעון או במנגד לכיוון השעון ביחס לקטע המכוון $\overrightarrow{p_0p_1}$. לשם כך אנו מחשבים את המכפלה הווקטורית $(p_1 - p_0) \times (p_2 - p_0)$. אם המכפלה הווקטורית הזאת שלילית, אזי $\overrightarrow{p_0p_2}$ נמצא במנגד לכיוון השעון ביחס ל־$\overrightarrow{p_0p_1}$, ולכן בנקודה p_1 אנו פונים שמאלה. מכפלה וקטורית חיובית מצביעה על כיוון השעון ועל פנייה ימינה. אם המכפלה הווקטורית שווה לאפס, פירוש הדבר שהנקודות p_0, p_1 ו־p_2 נמצאות כולן על ישר אחד.



איור 35.2 שימוש במכפלה וקטורית כדי לקבוע כיצד פונים שני קטעים עוקבים $\overline{p_0p_1}$ ו־$\overline{p_1p_2}$ בנקודה p_1. אנו בודקים אם הקטע המכוון $\overrightarrow{p_0p_2}$ נמצא בכיוון השעון או במנגד לכיוון השעון ביחס לקטע המכוון $\overrightarrow{p_0p_1}$. **(i)** אם הוא נמצא במנגד לכיוון השעון, אזי הפנייה היא שמאלה. **(ii)** אם הוא נמצא בכיוון השעון, אזי הפנייה היא ימינה.

כיצד קובעים אם שני קטעים נחתכים

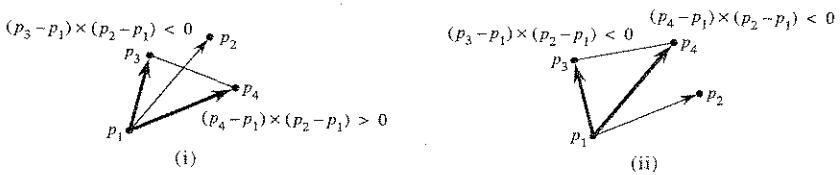
נשתמש בתהליך דו־שלבי כדי לקבוע אם שני קטעים נחתכים. השלב הראשון הוא **פסילה מהירה** (quick rejection): שני הקטעים אינם יכולים להיחתך אם הקופסאות החוסמות אותם אינן נחתכות. **הקופסה החוסמת** (bounding box) של צורה גיאומטרית היא המלבן הקטן ביותר המכיל את הצורה וצלעותיו מקבילות לציר ה־x ולציר ה־y. הקופסה החוסמת של קטע $\overline{p_1p_2}$ מיוצגת על־ידי המלבן $(\hat{p}_1, \hat{p}_2)$ שהנקודה השמאלית התחתונה שלו היא $\hat{p}_1 = (\hat{x}_1, \hat{y}_1)$ והנקודה הימנית העליונה שלו היא $\hat{p}_2 = (\hat{x}_2, \hat{y}_2)$, כאשר $\hat{x}_1 = \min(x_1, x_2)$, $\hat{y}_1 = \min(y_1, y_2)$, $\hat{x}_2 = \max(x_1, x_2)$ ו־$\hat{y}_2 = \max(y_1, y_2)$. שני מלבנים $(\hat{p}_1, \hat{p}_2)$ ו־$(\hat{p}_3, \hat{p}_4)$, המיוצגים על־ידי הנקודות השמאליות התחתונות והימניות העליונות שלהם, נחתכים אם ורק אם מתקיימת הקוניונקציה:

$$(\hat{x}_2 \geq \hat{x}_3) \wedge (\hat{x}_4 \geq \hat{x}_1) \wedge (\hat{y}_2 \geq \hat{y}_3) \wedge (\hat{y}_4 \geq \hat{y}_1)$$

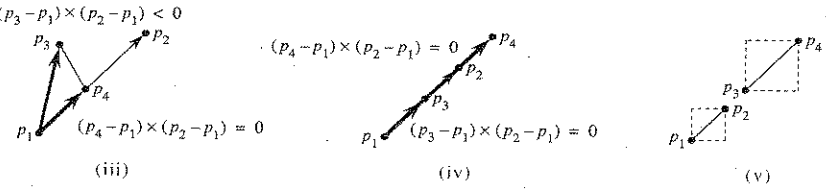
המלבנים חייבים להיחתך בשני ממדיהם. שתי ההשוואות הראשונות שלעיל בודקות אם המלבנים נחתכים ב־x, שתי ההשוואות האחרונות בודקות אם המלבנים נחתכים ב־y.

השלב השני בבדיקה אם שני קטעי ישרים נחתכים בודק עבור כל קטע אם הוא "רוכב" על הישר המכיל את הקטע האחר. קטע $\overline{p_1p_2}$ **רוכב** (straddles) על ישר אם הנקודה p_1 נמצאת מעברו האחד של הישר והנקודה p_2 נמצאת מעברו האחר. אם p_1 או p_2 נמצאות על הישר, גם אז אנו אומרים שהקטע רוכב על הישר. שני קטעים נחתכים אם ורק אם הם עוברים את מבחן הפסילה המהירה וגם כל קטע רוכב על הישר המכיל את הקטע האחר.

ניתן להשתמש בשיטת המכפלה הווקטורית כדי לקבוע אם קטע ישר $\overline{p_3p_4}$ רוכב על הישר המכיל את הנקודות p_1 ו־p_2. הרעיון, כמתואר באיורים 35.3(i)-(ii), הוא לבדוק אם הקטעים המכוונים $\overrightarrow{p_1p_3}$ ו־$\overrightarrow{p_1p_4}$ פונים בכיוונים מנוגדים ביחס ל־$\overrightarrow{p_1p_2}$. אם כך הדבר, אזי הקטע רוכב על הישר. נזכור כי ניתן לקבוע כיוונים יחסיים באמצעות מכפלות וקטוריות, ופשוט נבדוק אם הסימנים של המכפלות הווקטוריות $(p_2 - p_1) \times (p_3 - p_1)$ ו־$(p_2 - p_1) \times (p_4 - p_1)$ שונים זה מזה. מקרה גבול מתרחש אם אחת מהמכפלות הווקטוריות או שתיהן שוות לאפס. במקרה זה, p_3 או p_4 נמצאות על הישר המכיל את הקטע $\overline{p_1p_2}$. מכיוון ששני הקטעים כבר עברו את מבחן הפסילה המהירה, פירוש הדבר שאחת מן הנקודות p_3 או p_4 חייבת להימצא על הקטע $\overline{p_1p_2}$. שני מצבים כאלה מוצגים באיורים 35.3(iii)-(iv). המקרה שבו שני הקטעים נמצאים על ישר אחד אבל אינם נחתכים, מקרה המוצג באיור 35.3(v), נפסל על־ידי מבחן הפסילה המהירה. מקרה הגבול האחרון מתרחש אם אחד הקטעים או שניהם הם באורך אפס, דהיינו, אם שתי נקודות הקצה של הקטע מתלכדות. אם שני הקטעים הם באורך אפס, אזי די במבחן הפסילה המהירה. אם רק אחד הקטעים, נאמר $\overline{p_3p_4}$, הוא באורך אפס, אזי הקטעים נחתכים אם ורק אם המכפלה הווקטורית $(p_2 - p_1) \times (p_3 - p_1)$ שווה לאפס.



איור 35.3 בדיקה אם הקטע $\overline{p_3p_4}$ רוכב על הישר המכיל את הקטע $\overline{p_1p_2}$. **(i)** אם הקטע רוכב על הישר, אזי המכפלות הווקטוריות $(p_3 - p_1) \times (p_2 - p_1)$ ו־$(p_4 - p_1) \times (p_2 - p_1)$ הן בעלות סימנים מנוגדים. **(ii)** אם הקטע אינו רוכב על הישר, אזי המכפלות הווקטוריות הן בעלות אותו סימן. **(iii)-(iv)** מקרי גבול שבהם לפחות אחת מן המכפלות הווקטוריות שווה לאפס והקטע רוכב על הישר. **(v)** מקרה גבול שבו הקטעים נמצאים על אותו ישר אך אינם נחתכים. שתי המכפלות הווקטוריות שוות לאפס, אולם הן לא יחושבו על־ידי האלגוריתם שלנו, משום שהקטעים אינם עוברים את מבחן הפסילה המהירה — הקופסאות החוסמות אותם אינן נחתכות.



איור 35.3 (המשך)

יישומים נוספים של מכפלות וקטוריות

בסעיפים הבאים בפרק זה נציג שימושים נוספים של מכפלות וקטוריות. בסעיף 35.3 נצטרך למיין קבוצת נקודות על פי הזוויות הקוטביות שלהן ביחס לנקודת ראשית נתונה. כפי שיתבקש הקורא להראות בתרגיל 35.1-2, ניתן להשתמש במכפלות וקטוריות כדי לבצע את ההשוואות שעורכת שגרת המיון. בסעיף 35.2 נשתמש בעצים אדומים־שחורים לשמירת הסדר האנכי של קבוצת קטעים שאינם נחתכים. במקום לשמור ערכים מפורשים של מפתחות, נחליף כל השוואת מפתחות המופיעה בקוד המטפל בעצים אדומים־שחורים, בחישוב של מכפלה וקטורית כדי לקבוע איזה משני קטעים החותכים ישר אנכי נתון נמצא מעל הקטע האחר.

תרגילים

35.1-1
הוכח שאם המכפלה $p_1 \times p_2$ חיובית, אזי הווקטור p_1 נמצא בכיוון השעון מן הווקטור p_2 ביחס לנקודת הראשית $(0, 0)$, ואם $p_1 \times p_2$ שלילית, אזי p_1 נמצא במנוגד לכיוון השעון מ־p_2.

35.1-2
כתוב פסאודו־קוד למיון סדרה $\langle p_1, p_2, \ldots, p_n \rangle$ בת n נקודות על פי הזוויות הקוטביות של הנקודות ביחס לנקודת ראשית נתונה p_0. השגרה שלך צריכה לרוץ בזמן $O(n \lg n)$ ולהשתמש במכפלות וקטוריות לצורך השוואת זוויות.

35.1-3
הראה כיצד ניתן לקבוע בזמן $O(n^2 \lg n)$ אם שלוש נקודות כלשהן מתוך קבוצה בת n נקודות נמצאות על ישר אחד.

35.1-4
פרופסור אמונדסן מציע את השיטה שלהלן כדי לקבוע אם סדרה $\langle p_0, p_1, \ldots, p_{n-1} \rangle$ בת n נקודות היא סדרת הקדקודים העוקבים של מצולע קמור. (בסעיף 16.4 מופיעות הגדרות הנוגעות למצולעים.) הפלט הוא "כן" אם הקבוצה $\{\angle p_i p_{i+1} p_{i+2} : i = 0, 1, \ldots, n-1\}$, כאשר חיבור האינדקסים מתבצע מודולו n, אינה מכילה פניות שמאלה וגם פניות ימינה; אחרת, הפלט הוא "לא". הראה שאף כי שיטה זו מתבצעת בזמן לינארי, לא תמיד היא מניבה את התשובה הנכונה. שנה את שיטת הפרופסור כך שתפיק תמיד את התשובה הנכונה בזמן לינארי.

35.1-5
בהינתן נקודה $p_0 = (x_0, y_0)$, **הקרן האופקית הימנית** (right horizontal ray) היוצאת מ־p_0 היא קבוצת הנקודות $\{p_i = (x_i, y_i) : x_i \geq x_0 \text{ וגם } y_i = y_0\}$; כלומר, זוהי קבוצת הנקודות היוצרות קו אופקי מימין ל־p_0, כולל p_0 עצמה. הראה כיצד ניתן לקבוע בזמן $O(1)$ אם קרן אופקית ימנית נתונה היוצאת מ־p_0 חותכת את הקטע $\overline{p_1 p_2}$, על־ידי רדוקציה של בעיה זו לבעיית הבדיקה אם שני קטעים נחתכים.

35.1-6
אחת הדרכים שבהן ניתן לקבוע אם נקודה p_0 נמצאת בתוך מצולע P פשוט, אך לא בהכרח קמור, היא להתבונן בקרן כלשהי היוצאת מ־p_0 ולבדוק אם הקרן חותכת את השפה של P מספר אי־זוגי של פעמים אולם p_0 עצמה אינה נמצאת על שפת P. הראה כיצד ניתן לחשב בזמן $\Theta(n)$ אם נקודה p_0 נמצאת בתוך מצולע P בן n קדקודים. (**רמז:** היעזר בתרגיל 35.1-5. ודא שהאלגוריתם שלך פועל כהלכה גם כאשר הקרן חותכת את שפת המצולע בקדקוד וגם כאשר הקרן מתלכדת עם צלע של המצולע.)

35.1-7
הראה כיצד ניתן לחשב בזמן $\Theta(n)$ את שטחו של מצולע פשוט, אך לא בהכרח קמור, בן n קדקודים.

35.2 כיצד קובעים אם שני קטעים כלשהם נחתכים

בסעיף זה נציג אלגוריתם הקובע אם קיימים שני קטעים נחתכים כלשהם בתוך קבוצת קטעים. האלגוריתם משתמש בשיטה הידועה בשם "סריקה" שאלגוריתמים רבים בגיאומטריה חישובית משתמשים בה. יתירה מזו, כפי שמדגימים התרגילים שבסוף הסעיף הזה, ניתן להשתמש באלגוריתם זה, או בגרסאות פשוטות יותר שלו, כדי לפתור בעיות אחרות בגיאומטריה חישובית.

האלגוריתם רץ בזמן $O(n \lg n)$, כאשר n הוא מספר הקטעים הנתונים. האלגוריתם רק קובע אם קיים חיתוך כלשהו; הוא אינו מדפיס את כל החיתוכים. (על פי תרגיל 35.2-1, מציאת כל החיתוכים בקבוצה בת n קטעים מתבצעת במקרה הגרוע בזמן $\Omega(n^2)$.)

בעת **סריקה** (sweeping), מעבירים **קו סריקה** (sweep line) אנכי דמיוני דרך קבוצה נתונה של עצמים גיאומטריים, בדרך כלל משמאל לימין. אל הממד שלאורכו נע קו הסריקה, במקרה זה הממד x, מתייחסים כאל ממד הזמן. באמצעות סריקה ניתן לסדר עצמים גיאומטריים, בדרך כלל על־ידי הצבתם במבנה נתונים דינמי, ולנצל את היחסים ביניהם. האלגוריתם למציאת קטעים נחתכים שבסעיף זה בוחן את כל נקודות הקצה של הקטעים משמאל לימין, ובכל פעם שהוא מגיע לנקודת קצה, בודק אם קיים חיתוך.

האלגוריתם שלנו, הקובע אם בקבוצה בת n קטעים ישנם שני קטעים נחתכים כלשהם, מניח שתי הנחות מפשטות. ראשית, אנו מניחים שלא קיים בין קטעי הקלט קטע אנכי. שנית, אנו מניחים שבין קטעי הקלט לא קיימים שלושה קטעים הנחתכים בנקודה אחת. (בתרגיל 32.5-8 יתבקש

איור 35.4 (i) קטעים p_1q_1, p_2q_2 ו-p_3q_3 ונקודת החיתוך x שלהם. (ii) נתון מסלול T שמאונך לציר x ב-x, ומסלול T' שמאונך לציר x ב-x' כאשר $x' < x$. המסלול T מכיל את הקטעים p_1q_1 ו-p_2q_2, ולכן $p_1 <_x p_2$. המסלול T' מכיל את הקטעים p_2q_2 ו-p_3q_3, ולכן $p_2 <_{x'} p_3$. מכיוון ש-p_2 נמצא בין p_1 ל-p_3 במסלול T, נובע שאם $p_1 <_x p_3$ אז $p_1 <_{x'} p_3$ או $p_1 = p_3$.

לבסוף נדון בבעיה שבה נתונה קבוצה S של n קטעים ישרים, ואנו רוצים לבדוק אם קיים זוג קטעים ב-S שנחתכים. נניח לשם הפשטות שאין שלושה קטעים ב-S שנחתכים באותה נקודה, ושאף קטע ב-S אינו מקביל לציר y. נשתמש באלגוריתם הסריקה הכללי שתואר בסעיף 35.2, אבל נחליף את הנחת הכלליות שהוצגה בתחילת הסעיף בהנחה שהקטעים ב-S מופיעים בסדר הנכון לאורך הציר x. כלומר, נניח שאם p_1q_1 ו-p_2q_2 הם קטעים ב-S ו-$p_1 <_x p_2$, אז p_1 מופיע לפני p_2 בסדר הנתון. נניח גם שאין שני קטעים ב-S שמתחילים או נגמרים באותה נקודה.

המבנה הנתונים שבו נשתמש הוא מסלול T שמעדכן את רשימת הקטעים המתאימים לכל נקודה בסריקה. נשתמש בפעולות הבאות:

- BELOW(T, s): מחזירה את הקטע מעל הקטע s ב-T.
- ABOVE(T, s): מחזירה את הקטע הנמצא מעל s ב-T.
- DELETE(T, s): מחיקת הקטע s מ-T.
- INSERT(T, s): הכנסת הקטע s ל-T.

מצב יקר הוא מסלול T מלא שמעדכן את רשימת הקטעים המתאימים.

מאמוד-קוד לבדיקת קיומם של קטעים נחתכים

המסלול T של שני קטעים נראה תמיד כמו בציור 35.2-2.

במהלך האלגוריתם נתחזק את הקטעים המתאימים בסריקה בסדר הנכון לאורכם. נניח בינתיים שאין שני קטעים ב-S שנחתכים בנקודת קצה משותפת. נדון בהמשך בדרכים לטפל במקרים שבהם קיימת נקודת קצה משותפת לשני קטעים או יותר. נניח גם שאין שלושה קטעים ב-S שנחתכים באותה נקודה. נדון בהמשך בדרכים לטפל במקרים שבהם קיימת נקודה שבה נחתכים שלושה קטעים או יותר. נניח שאין קטעים ב-S שמקבילים לציר y. נניח שהקטעים ב-S מופיעים בסדר הנכון לאורך הציר x, ושאין שני קטעים ב-S שמתחילים או נגמרים באותה נקודה.

האלגוריתם שמתואר כאן מחזיר TRUE אם שני קטעים כלשהם ב-S נחתכים, ו-FALSE אחרת. הסדר המלא T ממומש על-ידי עץ אדום-שחור.[2]

---

[1] בהנחה שאין שני קטעים ב-S שנחתכים בנקודת קצה משותפת.

[2] ניתן לממש את T גם בעזרת עץ חיפוש מאוזן אחר, כגון עץ AVL או עץ 2-3. אנו בחרנו בעץ אדום-שחור מכיוון שהוא מתאים יותר לשימוש בהמשך הפרק.

מצב קו הסריקה (sweep-line status)

מצא את היחסים בין הקטעים המתחכים עם קו הסריקה.

1. **מצב קו הסריקה** (sweep-line status) מצא את היחסים בין הקטעים המתחכים עם קו הסריקה.
2. **לוח המנים של נקודות אירוע** (event-point schedule) הוא סדרה של נקודות בתוכנית שנקראות **נקודות אירוע** (event points). כל נקודת אירוע היא נקודת קצה של קטע, שבה מתחיל או נגמר קטע. לוח המנים מכיל את נקודות האירוע לפי סדר הופעתן בסריקה.

מבנים נתונים

המבנים הנתונים שבהם נשתמש הם:

- T: מסלול שמייצג את מצב קו הסריקה. כל עדכון של T יתבצע בעזרת הפעולות המתאימות.
- Q: תור עדיפויות שמייצג את לוח המנים. כל עדכון של Q יתבצע בעזרת הפעולות המתאימות.

מיון נקודות הקצה

נניח שהקטעים ב-S ממוינים לפי הקואורדינטה x של נקודות הקצה שלהם. נניח גם שאין שני קטעים ב-S שמתחילים או נגמרים באותה נקודה, ושאין קטע ב-S שמקביל לציר y. נניח שהקטעים ב-S מופיעים בסדר הנכון לאורך הציר x, כלומר אם p_1q_1 ו-p_2q_2 הם קטעים ב-S ו-$p_1 <_x p_2$, אז p_1 מופיע לפני p_2 בסדר הנתון. נניח גם שאין שני קטעים ב-S שנחתכים בנקודת קצה משותפת או שלושה קטעים ב-S שנחתכים באותה נקודה. נדון בהמשך בדרכים לטפל במקרים שבהם קיימת נקודת קצה משותפת לשני קטעים או יותר, או נקודה שבה נחתכים שלושה קטעים או יותר.

מבני נתונים

מבני נתונים מעניין אותנו בעיקר מבחינת הפעולות שהם תומכים בהן. נתבונן בשני סוגים של מבני נתונים: מילונים ותורי עדיפויות.

מילונים

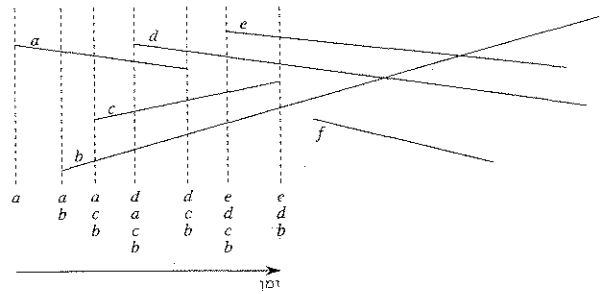
מילון (dictionary) הוא מבנה נתונים שתומך בפעולות הכנסה (INSERT), מחיקה (DELETE) ובדיקת חברות (MEMBER). כלומר, מילון מתחזק קבוצה S של איברים, כאשר כל איבר x מזוהה עם מפתח $key[x]$. אם S ו-T הן קבוצות, אז $S \cup T$ מסמן את האיחוד של S ו-T, $S \cap T$ מסמן את החיתוך של S ו-T, ו-$S - T$ מסמן את ההפרש של S ו-T, כלומר הקבוצה המכילה את כל האיברים ב-S שאינם ב-T.

תור עדיפויות (priority queue) הוא מבנה נתונים שתומך בפעולות הכנסה (INSERT), מציאת מקסימום (MAXIMUM) והוצאת מקסימום (EXTRACT-MAX). כלומר, תור עדיפויות מתחזק קבוצה S של איברים, כאשר כל איבר x מזוהה עם מפתח $key[x]$. אם S ו-T הן קבוצות, אז $S \cup T$ מסמן את האיחוד של S ו-T, $S \cap T$ מסמן את החיתוך של S ו-T, ו-$S - T$ מסמן את ההפרש של S ו-T, כלומר הקבוצה המכילה את כל האיברים ב-S שאינם ב-T.

```
ANY-SEGMENTS-INTERSECT(S)
 1  T ← φ
 2  sort the endpoints of the segments in S from left to right,
        breaking ties by putting left endpoints before right endpoints
 3  for each point p in the sorted list of endpoints
 4      do if p is the left endpoint of a segment s
 5          then INSERT(T, s)
 6               if (ABOVE(T, s) exists and intersects s)
                        or (BELOW(T, s) exists and intersects s)
 7                  then return TRUE
 8       if p is the right endpoint of a segment s
 9          then if both ABOVE(T, s) and BELOW(T, s) exist
                        and ABOVE(T, s) intersects BELOW(T, s)
10                  then return TRUE
11               DELETE(T, s)
12  return FALSE
```

איור 35.5 מדגים את פעולתו של האלגוריתם. שורה 1 מאתחלת את הסדר המלא כך שיהיה ריק. בשורה 2 קובעים את לוח הזמנים של נקודות האירוע על־ידי כך שממיינים את נקודות הקצה של $2n$ הקטעים משמאל לימין, ובמקרה של שוויון, נקודת קצה שמאלית מוצבת לפני נקודת קצה ימנית. נשים לב כי ניתן לבצע את שורה 2 על־ידי מיון לקסיקוגרפי של נקודות הקצה על פי (x, y).

כל איטרציה של לולאת ה־**for** שבשורות 3-11 מעבדת נקודת אירוע אחת p. אם p היא נקודת הקצה השמאלית של קטע s, אזי שורה 5 מוסיפה את s לסדר המלא, ושורות 6-7 מחזירות TRUE אם s חותך את הקטע הנמצא מעליו או את הקטע הנמצא מתחתיו בסדר המלא המוגדר על־ידי קו הסריקה העובר דרך p. (מקרה גבול הוא המקרה שבו p שוכנת על קטע אחר s'. במקרה זה, אנו דורשים רק שהקטעים s ו־s' יצורפו ל־T בזה אחר זה.) אם p היא נקודת הקצה הימנית של קטע s, אזי יש למחוק את s מן הסדר המלא. שורות 9-10 מחזירות TRUE אם קיים חיתוך בין הקטעים הנמצאים משני עברי s בסדר המלא המוגדר על־ידי קו הסריקה העובר דרך p, כאשר s יימחק, קטעים אלה יהפכו להיות עוקבים בסדר המלא. אם קטעים אלה אינם נחתכים, שורה 11 מוחקת את הקטע s מן הסדר המלא. לבסוף, אם בתום עיבוד כל $2n$ נקודות הקצה לא נמצאו שום חיתוכים, שורה 12 מחזירה FALSE.



איור 35.5 הרצה של ANY-SEGMENTS-INTERSECT. כל קו מקווקו הוא קו סריקה העובר דרך נקודת אירוע, והסדר של שמות הקטעים מתחת לכל קו סריקה הוא הסדר המלא T בסיומה של האיטרציה בלולאת ה־**for** שבה עובדה נקודת האירוע המתאימה. החיתוך בין הקטעים d ו־b מתגלה כאשר מורחקים את הקטע c.

נכונות

המשפט הבא מוכיח את נכונותה של ANY-SEGMENTS-INTERSECT.

משפט 35.1
הקריאה ANY-SEGMENTS-INTERSECT(S) מחזירה TRUE אם ורק אם קיים אישהו חיתוך בין הקטעים השייכים ל־S.

הוכחה השגרה יכולה לטעות רק אם היא מחזירה TRUE כאשר לא קיימים שום חיתוכים או אם היא מחזירה FALSE כאשר קיים לפחות חיתוך אחד. המקרה הראשון אינו יכול להתרחש, מכיוון ש־ANY-SEGMENTS-INTERSECT מחזירה TRUE רק אם היא מוצאת חיתוך בין שני קטעי קלט.

כדי להראות שגם המקרה השני אינו יכול להתרחש, הבה נניח בדרך השלילה שקיים לפחות חיתוך אחד, ובכל זאת ANY-SEGMENTS-INTERSECT מחזירה את הערך FALSE. תהי p נקודת החיתוך השמאלית ביותר, כאשר במקרה של שוויון, בוחרים בנקודה בעלת קואורדינטת ה־y הקטנה ביותר, ויהיו a ו־b הקטעים הנחתכים ב־p. מכיוון שמשמאל ל־p לא קיימות נקודות חיתוך, הסדר הנתון על־ידי T נכון בכל הנקודות שמשמאל ל־p. מאחר שלא קיימים שלושה קטעים הנחתכים בנקודה אחת, קיים קו סריקה z שבסדר המלא שלו a ו־b הם עוקבים[3]. יתר על כן, z נמצא משמאל ל־p או עובר דרכה. על קו הסריקה z קיימת נקודת קצה q של איזשהו קטע, שהיא נקודת האירוע אשר בה a ו־b הופכים להיות עוקבים בסדר המלא. אם p נמצאת על קו הסריקה z, אזי $p = q$. אם p אינה נמצאת על קו הסריקה z, אזי q נמצאת משמאל ל־p. בשני המקרים, הסדר הנתון על־ידי T הוא נכון ממש לפני שמתחיל העיבוד של q. (כאן אנו מסתמכים על כך ש־p היא הנמוכה מבין נקודות החיתוך השמאליות ביותר. בשל העובדה שנקודות האירוע מעובדות בסדר לקסיקוגרפי, גם אם p נמצאת על קו הסריקה z וקיימת נקודת חיתוך נוספת p' על z, נקודת האירוע $p = q$ מעובדת לפני שנקודת החיתוך האחרת p' יכולה להתערב בסדר המלא T.) בנקודת האירוע q קיימות שתי אפשרויות פעולה בלבד:

1. אחד מבין הקטעים a או b מוכנס ל־T, והקטע האחר נמצא מעליו או מתחתיו בסדר המלא. שורות 4-7 מגלות מקרה זה.
2. הקטעים a ו־b כבר נמצאים ב־T, והקטע הנמצא ביניהם בסדר המלא נמחק, כך שהם הופכים להיות עוקבים. שורות 8-11 מגלות מקרה זה.

בכל אחד מהמקרים, השגרה מוצאת את נקודת החיתוך p, בסתירה להנחה שהשגרה מחזירה FALSE. ■

זמן הריצה

אם הקבוצה S מכילה n קטעים, אזי ANY-SEGMENTS-INTERSECT רצה בזמן $O(n \lg n)$. שורה 1 מתבצעת בזמן $O(1)$. שורה 2 מתבצעת בזמן $O(n \lg n)$ תוך שימוש במיון־מיזוג או במיון־ערימה. מכיוון שקיימות $2n$ נקודות אירוע, לולאת ה־**for** בשורות 3-11 מבצעת $2n$ איטרציות לכל היותר. כל איטרציה מתבצעת בזמן $O(\lg n)$, שכן כל פעולה על עץ אדום־שחור מתבצעת בזמן $O(\lg n)$, ואם משתמשים בשיטה מסעיף 35.1, כל בדיקת חיתוך מתבצעת בזמן $O(1)$. זמן הריצה הכולל הוא אפוא $O(n \lg n)$.

3. אם נרשה ששלושה קטעים ייחתכו באותה נקודה, ייתכן שיהיה קיים קטע מתערב c, החותך את a וגם את b בנקודה p. כלומר, ייתכן שיתקיים $a <_w c$ ו־$c <_w b$ עבור כל קווי הסריקה w הנמצאים משמאל ל־p ואשר עבורם $a <_w b$.

תרגילים

35.2-1
הראה שקבוצה בת n קטעים עשויה להכיל $\Theta(n^2)$ חיתוכים.

35.2-2
בהינתן שני קטעים a ו־b שאינם נחתכים והם בני־השוואה ב־x, הראה כיצד ניתן להשתמש במכפלות וקטוריות כדי לקבוע בזמן $O(1)$ איזה מ־b $a >_x b$ או $b >_x a$ מתקיים.

35.2-3
פרופסור מאזיינו מציע לשנות את ANY-SEGMENTS-INTERSECT כך שבמקום לעצור ולחזור עם גילויה של נקודת חיתוך, השגרה תדפיס את הקטעים הנחתכים ותמשיך לאיטרציה הבאה של לולאת ה־**for**. הפרופסור מכנה את השגרה המתקבלת בשם PRINT-INTERSECTING-SEGMENTS וטוען שהיא מדפיסה את כל החיתוכים, משמאל לימין, כפי שהם מופיעים בקבוצת הקטעים. הראה שהפרופסור טועה על־ידי הצגתן של שתי דוגמאות נגדיות: מצא קבוצת קטעים שעבורה החיתוך הראשון שמוצאת PRINT-INTERSECTING-SEGMENTS אינו השמאלי ביותר וקבוצת קטעים שעבורה PRINT-INTERSECTING-SEGMENTS אינה מצליחה למצוא את כל החיתוכים.

35.2-4
כתוב אלגוריתם שזמן ריצתו $O(n \lg n)$ אשר קובע אם מצולע בן n קדקודים הוא פשוט.

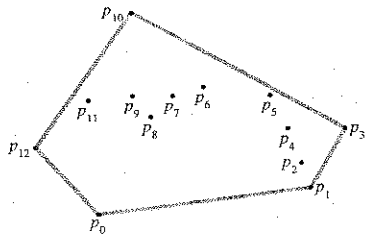
35.2-5
כתוב אלגוריתם שזמן ריצתו $O(n \lg n)$ אשר קובע אם שני מצולעים פשוטים שמספר קדקודיהם הכולל n, נחתכים.

35.2-6
עיגול (disk) מורכב ממעגל ומפנים המעגל, ומיוצג על־ידי המרכז והרדיוס שלו. שני עיגולים נחתכים אם יש להם נקודה משותפת כלשהי. כתוב אלגוריתם שזמן ריצתו $O(n \lg n)$ המקבל קבוצה בת n עיגולים וקובע אם קיימים ביניהם שני עיגולים כלשהם שנחתכים.

35.2-7
בהינתן קבוצה בת n קטעים המכילה בסך הכל k חיתוכים, הראה כיצד ניתן להדפיס את כל k החיתוכים בזמן $O((n+k)\lg n)$.

35.3 מציאת הקמור

ה**קמור** (convex hull) של קבוצת נקודות Q הוא המצולע הקמור הקטן ביותר P אשר כל נקודה של Q נמצאת בתוכו או על שפתו. אנו מסמנים את הקמור של Q ב־$\mathrm{CH}(Q)$. אינטואיטיבית, ניתן לחשוב על כל נקודה ב־Q כעל נעץ התקוע בלוח. הקמור הוא הצורה הנוצרת על־ידי גומייה מתוחה המקיפה את הנעצים. באיור 35.6 ניתן לראות קבוצת נקודות ואת הקמור שלה.



איור 35.6 קבוצת נקודות Q והקמור שלה $\mathrm{CH}(Q)$ (באפור).

בסעיף זה נציג שני אלגוריתמים המחשבים את הקמור של קבוצה בת n נקודות. שני האלגוריתמים מדפיסים את קדקודי הקמור על פי סדרם, במנגד לכיוון השעון. האלגוריתם הראשון, הידוע בשם "הסריקה של גראהם", רץ בזמן $O(n \lg n)$. האלגוריתם השני, המכונה "הצעדה של ג'ארביס", רץ בזמן $O(nh)$, כאשר h הוא מספר הקדקודים בקמור. כפי שניתן לראות באיור 35.6, כל קדקוד ב־$\mathrm{CH}(Q)$ הוא נקודה ב־Q. שני האלגוריתמים מנצלים תכונה זו ומחליטים אילו קדקודים של Q להשאיר כקדקודים של הקמור, ואילו קדקודים של Q לזרוק.

למעשה, קיימות שיטות אחדות לחישוב קמורים בזמן $O(n \lg n)$. שני האלגוריתמים שנציג, הסריקה של גראהם והצעדה של ג'ארביס, משתמשים בשיטה הקרויה "סריקה סיבובית" (rotational sweep), שבה מעבדים את הקדקודים בסדר הנקבע על פי הזוויות הקוטביות שהם יוצרים עם קדקוד ייחוס. מבין השיטות האחרות נמנה את הבאות:

- **שיטת הבנייה ההדרגתית** (the incremental method): בשיטה זו ממיינים את הנקודות משמאל לימין לקבלת סדרה $\langle p_1, p_2, \ldots, p_n \rangle$. בשלב ה־$i$, מעדכנים את הקמור $\mathrm{CH}(\{p_1, p_2, \ldots, p_{i-1}\})$ של $i-1$ הנקודות השמאליות ביותר על פי הנקודה ה־i משמאל, וכך מקבלים את $\mathrm{CH}(\{p_1, p_2, \ldots, p_i\})$. כפי שיתבקש הקורא להראות בתרגיל 35.3-6, ניתן לממש שיטה זו כך שתרוץ בזמן כולל של $O(n \lg n)$.
- **שיטת הפרד־ומשול** (the divide-and-conquer method): בזמן $\Theta(n)$ מחלקים את קבוצת n הנקודות לשתי תת־קבוצות, אחת המכילה את $\lceil n/2 \rceil$ הנקודות השמאליות ביותר והשנייה — את $\lfloor n/2 \rfloor$ הנקודות הימניות ביותר, ומחשבים באופן רקורסיבי את הקמורים של שתי תת־קבוצות. לאחר מכן משתמשים בשיטה חכמה לצירוף שני הקמורים בזמן $O(n)$.
- **שיטת גיזום־וחיפוש** (the prune-and-search method): שיטה זו דומה לאלגוריתם החציון מסעיף 10.3, שרץ בזמן לינארי במקרה הגרוע. בשיטה זו מוצאים את החלק העליון (או "השרשרת העליונה") של הקמור על־ידי כך שחוזרים ומסלקים חלק קבוע מן הנקודות שנותרו בקמור עד שנשארת רק השרשרת העליונה שלו. לאחר מכן מוצאים באותו אופן את השרשרת התחתונה של הקמור. שיטה זו היא המהירה ביותר אסימפטוטית: עבור קמור בן h נקודות, זמן הריצה הוא $O(n \lg h)$ בלבד.

חישוב הקמור של קבוצת נקודות הוא כשלעצמו בעיה מעניינת. יתר על כן, קיימים אלגוריתמים לפתרון בעיות אחרות בגיאומטריה חישובית המתחילים בחישוב קמור. לדוגמה, נתבונן בגרסה הדו־ממדית של **בעיית זוג הנקודות הרחוקות ביותר** (the farthest-pair problem): נתונה קבוצה של n נקודות במישור, וברצוננו למצוא את שתי הנקודות שהמרחק ביניהן מקסימלי. כפי שיתבקש הקורא להוכיח בתרגיל 35.3-3, שתי נקודות אלה הן בהכרח קדקודים של הקמור. אף כי לא נוכיח זאת כאן, ניתן למצוא את שני הקדקודים הרחוקים ביותר זה מזה במצולע קמור בעל n קדקודים בזמן $O(n)$. לכן, על־ידי חישוב הקמור של n נקודות הקלט בזמן $O(n \lg n)$ ולאחר

מכן מציאת זוג הקדקודים המרוחק ביותר מבין קדקודי המצולע הקמור המתקבל, נוכל למצוא את זוג הנקודות הרחוקות ביותר זו מזו בכל קבוצה בת n נקודות בזמן $O(n \lg n)$.

הסריקה של גראהם

הסריקה של גראהם (Graham's scan) פותרת את בעיית הקמור על־ידי כך שהיא מתחזקת מחסנית S של נקודות מועמדות. כל נקודה של קבוצת הקלט Q נדחפת למחסנית פעם אחת, והנקודות שאינן קדקודים של $\mathrm{CH}(Q)$ נשלפות בסופו של דבר מן המחסנית. כאשר האלגוריתם עוצר, המחסנית S מכילה בדיוק את קדקודי $\mathrm{CH}(Q)$, בסדר הופעתם על שפת הקמור, במנוגד לכיוון השעון.

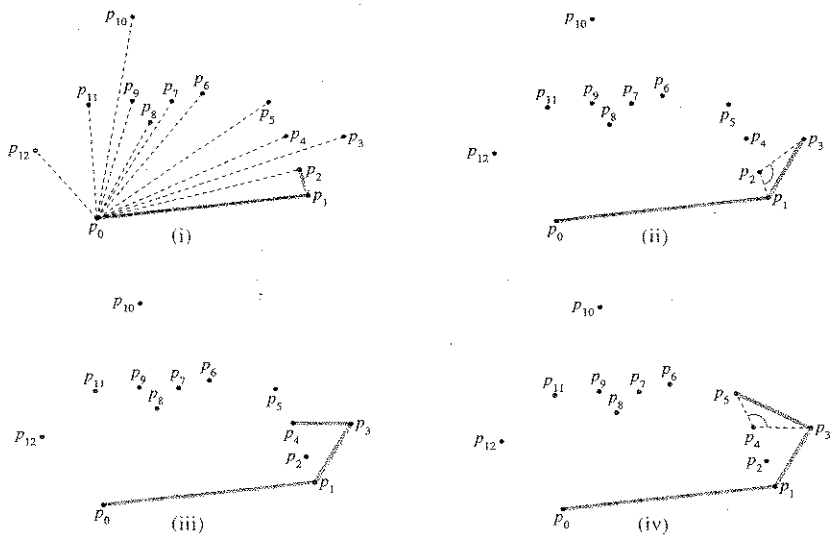
השגרה GRAHAM-SCAN מקבלת כקלט קבוצה Q של נקודות, כאשר $|Q| \geq 3$. השגרה קוראת לפונקציה $\mathrm{TOP}(S)$, אשר מחזירה את הנקודה הנמצאת בראש המחסנית S בלי לשנות את S, ולפונקציה $\mathrm{NEXT\text{-}TO\text{-}TOP}(S)$, המחזירה את הנקודה הנמצאת בתא שמתחת לראש המחסנית S, בלי לשנות את S. כפי שנוכיח מיד, המחסנית S שמחזירה GRAHAM-SCAN מכילה, מלמטה למעלה, בדיוק את קדקודי $\mathrm{CH}(Q)$, במנוגד לכיוון השעון.

```
GRAHAM-SCAN(Q)
1  let p_0 be the point in Q with the minimum y-coordinate,
       or the leftmost such point in case of a tie
2  let ⟨p_1, p_2, ..., p_m⟩ be the remaining points in Q,
       sorted by polar angle in counterclockwise order around p_0
       (if more than one point has the same angle, remove all but
       the one that is farthest from p_0)
3  top[S] ← 0
4  PUSH(P_0, S)
5  PUSH(P_1, S)
6  PUSH(P_2, S)
7  for i ← 3 to m
8      do while the angle formed by points NEXT-TO-TOP(S)
                 TOP(S), and p_i makes a nonleft turn
9          do POP(S)
10         PUSH(p_i, S)
11 return S
```

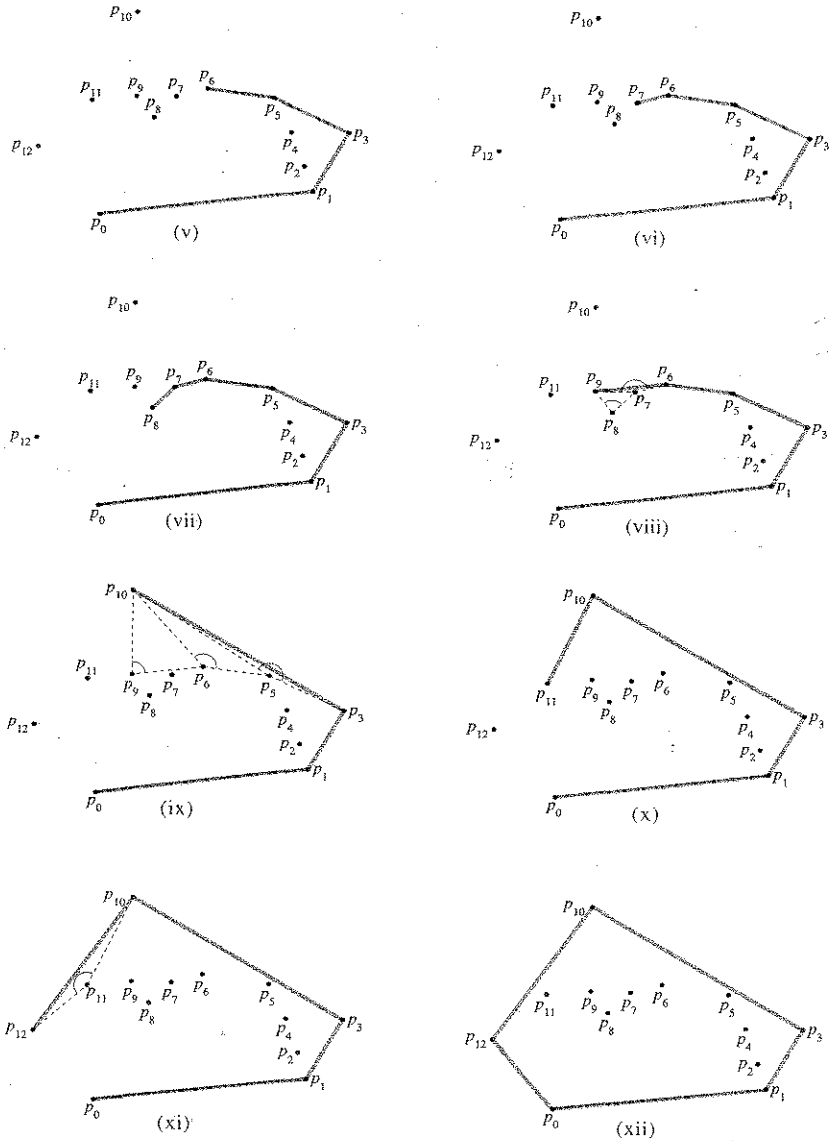
באיור 35.7 מדגמת התקדמותה של GRAHAM-SCAN. שורה 1 בוחרת בנקודה בעלת קואורדינטת ה־y הנמוכה ביותר בקדקוד p_0, ובמקרה של שוויון, בחרת הנקודה השמאלית ביותר הזאת. מכיוון שאין ב־Q נקודה הנמצאת מתחת ל־p_0 וכל הנקודות האחרות בעלות אותה קואורדינטת y נמצאות מימינה, p_0 היא קדקוד של $\mathrm{CH}(Q)$. שורה 2 ממיינת את שאר הנקודות ב־Q על פי הזוויות הקוטביות שלהן ביחס ל־p_0 באותה שיטה ששימשה בתרגיל 35.1-2 — השוואת מכפלות וקטוריות. אם קיימות שתי נקודות או יותר בעלות אותה זווית קוטבית ביחס ל־p_0, כל הנקודות הללו, פרט לרחוקה ביותר, הן צירופים קמורים של p_0 והנקודה הרחוקה ביותר, ולכן אין שום צורך להביא אותן בחשבון. נסמן ב־m את מספר הנקודות הנותרות, לא כולל p_0. הזווית הקוטבית של כל נקודה ב־Q ביחס ל־p_0, כשהיא נמדדת ברדיאנים, נמצאת בקטע החצי פתוח $[0, \pi/2)$. מכיוון שהזוויות הקוטביות גדלות במנוגד לכיוון השעון, הנקודות ממוינות בסדר מנוגד לכיוון השעון ביחס ל־p_0. נסמן ב־$\langle p_1, p_2, \ldots, p_m \rangle$ את קבוצת הנקודות הממוינות המתקבלת. נשים

לב כי הנקודות p_1 ו־p_m הן קדקודים של $\mathrm{CH}(Q)$ (ראה תרגיל 35.3-1). באיור 35.7(i) מופיעות הנקודות מאיור 35.6 בצירוף הזוויות הקוטביות הממוינות של $\langle p_1, p_2, \ldots, p_{12} \rangle$ ביחס ל־p_0.

החל משורה 3 משתמשת השגרה במחסנית S. שורות 3-6 מאתחלות את המחסנית כך שתכיל, מלמטה למעלה, את שלוש הנקודות הראשונות p_0, p_1 ו־p_2. באיור 35.7(i) מופיעה המחסנית S במצב ההתחלתי. לולאת ה־**for** בשורות 7-10 מבצעת איטרציה אחת עבור כל נקודה בתת־סדרה $\langle p_3, p_4, \ldots, p_m \rangle$. הכוונה היא שלאחר עיבוד הנקודה p_i, המחסנית S תכיל, מלמטה למעלה, את קדקודי $\mathrm{CH}(\{p_0, p_1, \ldots, p_i\})$ בסדר מנוגד למגמת השעון. לולאת ה־**while** בשורות 8-9 מוציאה מן המחסנית נקודות אשר נמצא כי אינן קדקודים של הקמור. כאשר עוברים על הקמור במנוגד לכיוון השעון, יש לפנות שמאלה בכל קדקוד. לכן, בכל פעם שלולאת ה־**while** מוצאת קדקוד שבו אין מבצעים פנייה שמאלה, קדקוד זה נשלף מן המחסנית. ועל־ידי כך שמחפשים פנייה שאינה שמאלה ולא פשוט פנייה ימינה, מונעים אפשרות של זווית שטוחה בקדקוד של הקמור המתקבל. זהו בדיוק הדבר שאנו זקוקים לו, מכיוון שאסור שקדקוד של מצולע קמור יהיה צירוף קמור של קדקודים אחרים של המצולע.) לאחר ששלפנו את כל הקדקודים שהפנייה בהם אינה שמאלה כאשר מתקדמים לעבר הנקודה p_i, דוחפים את p_i למחסנית. איורים 35.7(ii)-(xi) מראים את מצב המחסנית S לאחר כל איטרציה של לולאת ה־**for**. לבסוף, GRAHAM-SCAN מחזירה את המחסנית S בשורה 11. באיור 35.7(xii) מופיע הקמור המתקבל.



איור 35.7 פעולתה של GRAHAM-SCAN על הקבוצה Q מאיור 35.6. בכל שלב מופיע באפור הקמור הנוכחי המאוחסן במחסנית S. **(i)** הזוויות הקוטביות הממוינות של $\langle p_1, p_2, \ldots, p_{12} \rangle$ יחסית ל־p_0 והמחסנית ההתחלתית S, המכילה את הנקודות p_0, p_1 ו־p_2. **(ii)-(xi)** המחסנית S לאחר כל איטרציה של לולאת ה־**for** בשורות 7-10. הקווים המקוטקווים מייצגים פניות שאינן שמאלה, הנורמות לשלִיפת נקודות מן המחסנית. לדוגמה, בחלק (xi) של האיור, הפנייה ימינה בזווית $\angle p_9 p_8 p_7$ גורמת לשלִיפתה של p_8, ולאחר מכן, הפנייה ימינה בזווית $\angle p_8 p_7 p_6$ גורמת לשלִיפתה של p_7. **(xii)** הקמור שמחזירה השגרה, הזהה לקמור שבאיור 35.6.



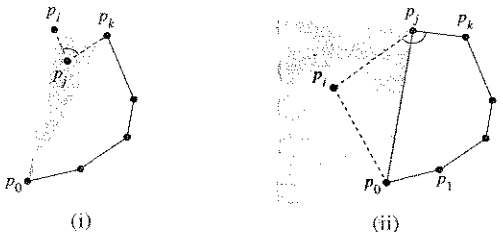
איור 35.7 (המשך)

המשפט הבא מוכיח באופן פורמלי את נכונותה של GRAHAM-SCAN.

משפט 35.2
אם מריצים את GRAHAM-SCAN על קבוצה Q של נקודות, כאשר $|Q| \geq 3$, אזי נקודה מ־Q נמצאת במחסנית S בעת העצירה אם ורק אם נקודה זו היא קדקוד של $\mathrm{CH}(Q)$.

הוכחה כאמור לעיל, קדקוד שהוא צירוף קמור של p_0 ושל איזשהו קדקוד אחר ב־Q איננו קדקוד של $\mathrm{CH}(Q)$. קדקוד כזה אינו נכלל בסדרה $\langle p_1, p_2, \ldots, p_m \rangle$, ולכן לעולם אינו מופיע במחסנית S.

לב ההוכחה טמון בשני המצבים המוצגים באיור 35.8. חלק (i) של האיור עוסק בפניות שאינן פניות שמאלה, וחלק (ii) עוסק בפניות שמאלה.



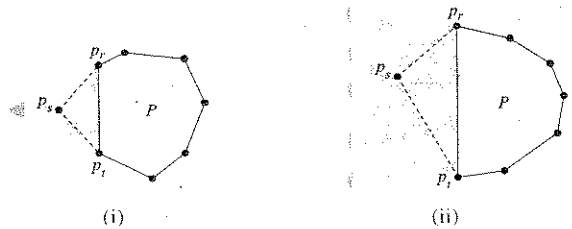
איור 35.8 שני המצבים הבסיסיים בהוכחת נכונותה של GRAHAM-SCAN. **(i)** חלק זה מראה שנקודה הנשלפת מהמחסנית על־ידי GRAHAM-SCAN אינה קדקוד של $\mathrm{CH}(Q)$. אם נקודה p_j נשלפת מן המחסנית מכיוון שהזווית $\angle p_k p_j p_i$ מבצעת פנייה שאינה שמאלה, אזי המשולש המוצלל $\Delta p_0 p_k p_i$ מכיל את הנקודה p_j. לכן, הנקודה p_j איננה קדקוד של $\mathrm{CH}(Q)$. **(ii)** אם נקודה p_i נדחפת למחסנית, אזי הפנייה המתבצעת בזווית $\angle p_k p_j p_i$ חייבת להיות פנייה שמאלה. מכיוון ש־p_i עוקבת ל־p_j בסידור הנקודות על פי זוויות קוטביות ובשל האופן שבו בוחרים את p_0, הנקודה p_i חייבת להימצא באזור המוצלל. אם הנקודות שבמחסנית יוצרות מצולע קמור לפני הדחיפה, הן יוצרות בהכרח מצולע קמור גם לאחר הדחיפה.

תחילה נראה שכל נקודה הנשלפת מן המחסנית S איננה קדקוד של $\mathrm{CH}(Q)$. נניח שהנקודה p_j נשלפת מן המחסנית מכיוון שהזווית $\angle p_k p_j p_i$ מבצעת פנייה שאינה שמאלה, כמתואר באיור 35.8(i). מכיוון שאנו סורקים את הנקודות בסדר עולה של זוויותיהן הקוטביות ביחס לנקודה p_0, קיים משולש $\Delta p_0 p_i p_k$ כך שהנקודה p_j נמצאת בתוך המשולש או על הקטע $\overline{p_i p_k}$. בכל אחד מהמקרים, הנקודה p_j אינה יכולה להיות קדקוד של $\mathrm{CH}(Q)$.

עתה נראה שכל נקודה הנמצאת במחסנית S בעת העצירה היא קדקוד של $\mathrm{CH}(Q)$. נתחיל בהוכחת הטענה הבאה: GRAHAM-SCAN משמרת את האינווריאנטה שהנקודות הנמצאות במחסנית S הן תמיד קדקודים של מצולע קמור בסדר מנוגד לכיוון השעון.

טענה זו מתקיימת מיד לאחר ביצועה של שורה 6, שכן הנקודות p_0, p_1 ו־p_2 יוצרות מצולע קמור. עתה נבחן כיצד משתנה המחסנית במהלך הביצוע של GRAHAM-SCAN. נקודות נשלפות מן המחסנית או נדחפות אליה. במקרה הראשון, נסתמך על תכונה גיאומטרית פשוטה: אם מסלקים קדקוד ממצולע קמור, המצולע המתקבל גם הוא קמור. לכן, שליפת נקודה מהמחסנית S משמרת את האינווריאנטה.

לפני שנדון במקרה שבו נקודה נדחפת למחסנית, הבה נבחן תכונה גיאומטרית נוספת, המתוארת באיורים 35.9(i)-(ii). יהי P מצולע קמור, ותהי $\overline{p_r p_t}$ צלע כלשהי של P. נתבונן באזור התחום על-ידי $\overline{p_r p_t}$ והמשכי שתי הצלעות הסמוכות. (הזוויות היחסיות של הצלעות הסמוכות קובעות אם זהו אזור חסום, כמו האזור המוצלל בחלק (i) של איור 35.9, או בלתי חסום, כמו האזור המוצלל בחלק (ii) של האיור.) אם נסיף ל-P נקודה p_s כלשהי הנמצאת באזור זה כקדקוד חדש, ונחליף את הצלע $\overline{p_r p_t}$ בצלעות $\overline{p_r p_s}$ ו-$\overline{p_s p_t}$, המצולע המתקבל יהיה גם הוא קמור.



איור 35.9 הוספת נקודה מן האזור המוצלל למצולע קמור P מניבה מצולע קמור אחר. האזור המוצלל תחום על-ידי הצלע $\overline{p_r p_t}$ והמשכי שתי הצלעות הסמוכות. **(i)** האזור המוצלל חסום. **(ii)** האזור המוצלל אינו חסום.

עתה נתבונן בנקודה p_i הנדחפת ל-S. נחזור לאיור 35.8(ii) ונניח כי p_j הוא הקדקוד שבראש המחסנית S ממש לפני דחיפתה של p_i, ותהי p_k הנקודה הקודמת ל-p_j ב-S. אנו טוענים כי p_i חייבת להימצא באזור המוצלל שבאיור 35.8(ii), המתאים בדיוק לאזורים המוצללים באיור 35.9. מאחר שהזווית $\angle p_k p_j p_i$ מבצעת פנייה שמאלה, p_i נמצאת בהכרח בצד המוצלל של המשך הצלע $\overline{p_k p_j}$. מאחר ש-p_i עוקבת ל-p_j בסידור על פי הזוויות הקוטביות, היא נמצאת בהכרח בצד המוצלל של הצלע $\overline{p_0 p_j}$. יתירה מזו, בשל האופן שבו בחרנו את p_0, הנקודה p_i חייבת להימצא בצד המוצלל של המשך הצלע $\overline{p_0 p_1}$. לכן, הנקודה p_i נמצאת באזור המוצלל, ולפיכך, לאחר שדחפנו את p_i למחסנית S, הנקודות ב-S יוצרות מצולע קמור. בכך נשלמה הוכחת הטענה.

אם כן, הנקודות מ-Q הנמצאות במחסנית S עם סיומה של GRAHAM-SCAN הן קדקודיו של מצולע קמור. כבר הראינו שכל הנקודות שאינן ב-S אינן קדקודים של $\mathrm{CH}(Q)$, או באופן שקול, שכל הקדקודים של $\mathrm{CH}(Q)$ נמצאים ב-S. מאחר ש-S מכילה אך ורק קדקודים מ-Q והנקודות ב-S יוצרות מצולע קמור, הן יוצרות בהכרח את $\mathrm{CH}(Q)$. ■

עתה נראה שזמן הריצה של GRAHAM-SCAN הוא $O(n \lg n)$, כאשר $n = |Q|$. שורה 1 מתבצעת בזמן $\Theta(n)$. שורה 2 מתבצעת בזמן $O(n \lg n)$, כאשר משתמשים במיון-מיזוג או מיון-ערימה למיון הזוויות הקוטביות ובשיטת המכפלות הווקטוריות מסעיף 35.1 להשוואת הזוויות. (את מחיקת כל הנקודות בעלות אותה זווית קוטבית, פרט לרחוקה ביותר, ניתן לבצע בזמן כולל של $O(n)$.) שורות 3-6 מתבצעות בזמן $O(1)$. מאחר ש-$m \le n - 1$, לולאת ה-**for** בשורות 7-10 מתבצעת $n - 3$ פעמים לכל היותר. מכיוון שהפעולה PUSH מתבצעת בזמן $O(1)$, כל איטרציה מתבצעת בזמן $O(1)$, לא כולל הזמן המוקדש לביצוע לולאת ה-**while** שבשורות 8-9. לפיכך, הזמן הכולל לביצוע לולאת ה-**for**, לא כולל ביצוע לולאת ה-**while** המקוננת, הוא $O(n)$.

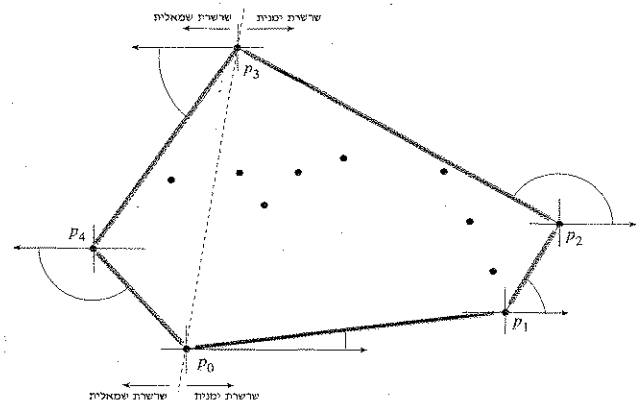
נשתמש בניתוח לשיעורין בשיטת הצבירה כדי להראות שהזמן הכולל לביצוע לולאת ה-**while** הוא $O(n)$. עבור $i = 0, 1, \ldots, m$, כל נקודה p_i נדחפת למחסנית S בדיוק פעם אחת. כמו בניתוח השגרה MULTIPOP בסעיף 18.1, נשים לב כי מתבצעת פעולת POP אחת לכל היותר עבור כל פעולת PUSH. לפחות שלוש נקודות — p_0, p_1 ו-p_m — לעולם אינן נשלפות מן המחסנית, ולכן למעשה, מתבצעות בסך הכל $m - 2$ פעולות POP לכל היותר. כל איטרציה של לולאת ה-**while** מבצעת פעולת POP אחת, ולכן בסך הכל מתבצעות לכל היותר $m - 2$ איטרציות של לולאת ה-**while**. מכיוון שהבדיקה בשורה 8 מתבצעת בזמן $O(1)$, כל קריאה ל-POP מתבצעת בזמן $O(1)$, ו-$m \le n - 1$, הרי שהזמן הכולל לביצוע לולאת ה-**while** הוא $O(n)$. זמן הריצה של GRAHAM-SCAN הוא אפוא $O(n \lg n)$.

הצעדה של ג'ארביס

הצעדה של ג'ארביס (Jarvis's march) מחשבת את הקמור של קבוצת נקודות Q בשיטה המכונה **עטיפת חבילות** או **עטיפת מתנות**. האלגוריתם רץ בזמן $O(nh)$, כאשר h הוא מספר הקדקודים של $\mathrm{CH}(Q)$. כאשר h הוא $o(\lg n)$, הצעדה של ג'ארביס מהירה יותר אסימפטוטית מן הסריקה של גראהם.

אינטואיטיבית, הצעדה של ג'ארביס מדמה פעולה שבה כורכים נייר עטיפה הדוק סביב הקבוצה Q. מתחילים בהדבקת קצה אחד של הנייר לנקודה הנמוכה ביותר בקבוצה, דהיינו, אותה נקודה p_0 שבה מתחילים את הסריקה של גראהם. נקודה זו היא קדקוד של הקמור. מושכים את הנייר ימינה כדי להדקו ואז מושכים אותו למעלה עד שהוא נוגע באיזשהי נקודה. גם נקודה זו חייבת להיות קדקוד של הקמור. ממשיכים לכרוך את הנייר סביב קבוצת הקדקודים באותו אופן, תוך כדי שמירה על הנייר מתוח, עד שחוזרים לנקודת ההתחלה p_0.

באופן פורמלי יותר, הצעדה של ג'ארביס בונה סדרה $H = \langle p_0, p_1, \ldots, p_{h-1} \rangle$ של קדקודי $\mathrm{CH}(Q)$. מתחילים עם p_0. כפי שמראה איור 35.10, הקדקוד הבא של הקמור, p_1, הוא בעל הזווית הקוטבית הקטנה ביותר ביחס ל-p_0. (במקרה של שוויון, בוחרים בנקודה המרוחקת ביותר מ-p_0.) באותו אופן, p_2 היא בעלת הזווית הקוטבית הקטנה ביותר ביחס ל-p_1, וכן הלאה. כאשר מגיעים לקדקוד הגבוה ביותר, נאמר p_k (במקרה של שוויון, בוחרים את הקדקוד המרוחק ביותר), מקבלים את **השרשרת הימנית** (right chain) של $\mathrm{CH}(Q)$ (ראה איור 35.10). כדי לבנות את **השרשרת השמאלית** (left chain), מתחילים ב-p_k ובוחרים כ-p_{k+1} את הנקודה בעלת הזווית הקוטבית הקטנה ביותר ביחס ל-p_k, אבל הפעם מתבוננים בזוויות הנוצרות עם *הקרן השלילית של* ציר ה-x. ממשיכים ביצירת השרשרת השמאלית על פי הזוויות הקוטביות עם הקרן השלילית של ציר ה-x עד שחוזרים לקדקוד המקור p_0.



איור 35.10 פעולת הצעדה של גיארביס. הקדקוד הראשון שנבחר הוא הנקודה הנמוכה ביותר p_0. הקדקוד הבא, p_1, הוא הנקודה בעלת הזווית הקוטבית הקטנה ביותר ביחס ל־p_0. לאחר מכן, p_2 היא הנקודה בעלת הזווית הקוטבית הקטנה ביותר ביחס ל־p_1. השרשרת הימנית מגיעה עד לנקודה הגבוהה ביותר, p_3. לאחר מכן בונים את השרשרת השמאלית על־ידי כך שמוצאים זוויות קוטביות קטנות ביותר ביחס לקרן השלילית של ציר ה־x.

ניתן היה לממש את הצעדה של גיארביס על־ידי סריקה אחת סביב הקמור, כלומר, בלי לבנות בנפרד את השרשרת הימנית והשרשרת השמאלית. מימושים כאלה עוקבים בדרך כלל אחר הזווית של הצלע האחרונה שצורפה לקמור ודורשים שסדרת הזוויות של צלעות הקמור תהיה סדרה עולה ממש (בתחום 0 עד 2π רדיאנים). היתרון שבבנייה נפרדת של השרשרות טמון בכך שאין צורך לחשב זוויות במפורש: די בשיטות להשוואת זוויות שהוצגו בסעיף 35.1.

אם היא ממומשת כהלכה, הצעדה של גיארביס רצה בזמן $O(nh)$. עבור כל אחד מ־h הקדקודים של $\mathrm{CH}(Q)$, אנו מוצאים את הקדקוד בעל הזווית הקוטבית המינימלית. כל השוואה בין זוויות קוטביות מתבצעת בזמן $O(1)$ אם משתמשים בשיטות מסעיף 35.1. כפי שראינו בסעיף 10.1, אם כל השוואה מתבצעת בזמן $O(1)$, ניתן לחשב את המינימום מבין n ערכים בזמן $O(n)$. זמן הריצה של הצעדה של גיארביס הוא אפוא $O(nh)$.

תרגילים

35.3-1

הוכח כי בשגרה GRAHAM-SCAN, הנקודות p_1 ו־p_m חייבות להיות קדקודים של $\mathrm{CH}(Q)$.

35.3-2

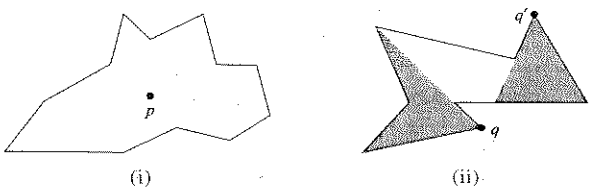
נתבונן במודל חישובי התומך בפעולות חיבור, השוואה וכפל, ואשר קיים עבורו חסם תחתון של $\Omega(n \lg n)$ על מיון n מספרים. הוכח כי במודל זה, $\Omega(n \lg n)$ הוא חסם תחתון עבור חישוב קדקודי הקמור של קבוצה בת n נקודות, על פי סדרם.

35.3-3

בהינתן קבוצת נקודות Q, הוכח שזוג הנקודות המרוחקות ביותר זו מזו חייבות להיות קדקודים של $\mathrm{CH}(Q)$.

35.3-4

בהינתן מצולע P ונקודה q על שפתו, **הצל** (shadow) של q הוא קבוצת הנקודות r שעבורן הקטע $\overline{qr}$ נמצא כולו על שפת המצולע P או בתוכו. מצולע P הוא מצולע בעל **צורת כוכב** (star shaped) אם בפנים המצולע P קיימת נקודה p השייכת לצל של כל נקודה על שפת המצולע P. קבוצת כל הנקודות p האלה נקראת **הגרעין** (kernel) של P. (ראה איור 35.11.) בהינתן מצולע P בעל n קדקודים ובעל צורת כוכב, המוגדר על־ידי קדקודיו בסדר מנוגד לכיוון השעון, הראה כיצד מחשבים את $\mathrm{CH}(P)$ בזמן $O(n)$.



איור 35.11 ההגדרה של מצולע בעל צורת כוכב, עבור תרגיל 35.3-4. **(i)** מצולע בעל צורת כוכב. הקטע המחבר את הנקודה p עם כל נקודה q על שפת המצולע חותך את השפה בנקודה q בלבד. **(ii)** מצולע שאינו בעל צורת כוכב. האזור המוצלל משמאל הוא הצל של q, והאזור המוצלל מימין הוא הצל של q'. מאחר ששני האזורים זרים זה לזה, הגרעין של המצולע ריק.

35.3-5

בגרסה המקוונת של בעיית הקמור, אנו מקבלים את n הנקודות המרכיבות את הקבוצה Q בזו אחר זו. לאחר שמתקבלת נקודה, עלינו לחשב את הקמור של הנקודות שהופיעו עד כה. כמובן, יכולנו להריץ את הסריקה של גראהם פעם אחת עבור כל נקודה, כך שזמן הריצה הכולל יהיה $O(n^2 \lg n)$. הראה כיצד ניתן לפתור את הגרסה המקוונת של בעיית הקמור בזמן כולל של $O(n^2)$.

* 35.3-6

הראה כיצד ניתן לממש את שיטת הבנייה ההדרגתית לחישוב הקמור של n נקודות כך שזמן הריצה יהיה $O(n \lg n)$.

35.4 מציאת זוג הנקודות הקרובות ביותר

עתה נתבונן בבעיה של מציאת זוג הנקודות הקרובות ביותר בקבוצה Q בת $n \geq 2$ נקודות. "קרבה" מתייחסת כאן למרחקים אוקלידיים רגילים: המרחק בין הנקודות $p_1 = (x_1, y_1)$ ו־$p_2 = (x_2, y_2)$ הוא $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. שתי נקודות בקבוצה Q עשויות להתלכד, ואז המרחק ביניהן הוא אפס. בעיה זו מופיעה, למשל, במערכות בקרת תנועה. מערכת לבקרת תנועה אווירית או ימית צריכה לדעת אילו הם שני כלי התחבורה הקרובים ביותר, כדי למנוע התנגשויות אפשריות.

אלגוריתם ישיר למציאת זוג הנקודות הקרובות ביותר פשוט בוחן את כל $\binom{n}{2} = \Theta(n^2)$ זוגות הנקודות. בסעיף זה נתאר אלגוריתם הפרד־ומשול אשר זמן ריצתו מתואר על־ידי נוסחת הנסיגה המוכרת $T(n) = 2T(n/2) + O(n)$. אלגוריתם זה פותר אפוא את הבעיה בזמן $O(n \lg n)$ בלבד.

אלגוריתם הפרד־ומשול

כל הפעלה רקורסיבית של האלגוריתם מקבלת כקלט תת־קבוצה $P \subseteq Q$ ושני מערכים X ו־Y, שכל אחד מהם מכיל את כל הנקודות בתת־קבוצה P. הנקודות במערך X ממוינות בסדר מונוטוני עולה של קואורדינטות ה־x שלהן. באופן דומה, הנקודות במערך Y ממוינות בסדר מונוטוני עולה של קואורדינטות ה־y שלהן. נשים לב שכדי להגיע לחסם $O(n \lg n)$ על זמן הריצה, איננו יכולים להרשות לעצמנו למיין בכל קריאה רקורסיבית, אילו עשינו זאת, נוסחת הנסיגה המתארת את זמן הריצה היתה $T(n) = 2T(n/2) + O(n \lg n)$, שפתרונה $T(n) = O(n \lg^2 n)$. בהמשך נראה כיצד ניתן להשתמש ב"מיזוג מקדים" כדי לשמר את המיון בלי למיין ממש בכל קריאה רקורסיבית.

קריאה רקורסיבית נתונה עם הקלטים P, X ו־Y בודקת תחילה אם $|P| \le 3$. אם כך הדבר, קריאה זו פשוט מבצעת את השיטה הישירה שתוארה לעיל: היא בודקת את כל $\binom{|P|}{2}$ זוגות הנקודות ומחזירה את הזוג הקרוב ביותר. אם $|P| > 3$, הקריאה הרקורסיבית פועלת בשיטת הפרד־ומשול כדלקמן.

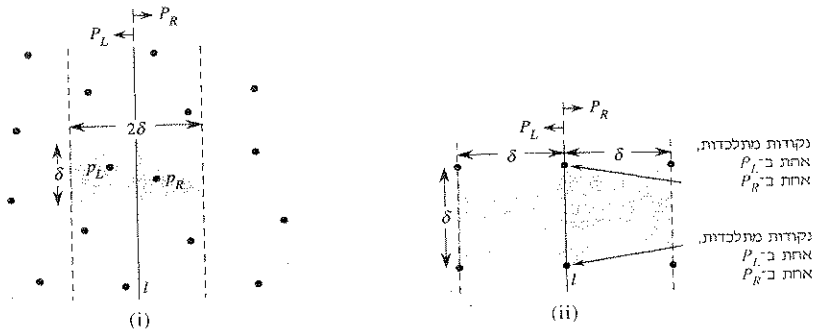
הפרד: היא מוצאת ישר אנכי l החוצה את קבוצת הנקודות P לשתי קבוצות P_L ו־P_R כך ש־$|P_L| = \lceil |P|/2 \rceil$, $|P_R| = \lfloor |P|/2 \rfloor$, כל הנקודות ב־$P_L$ נמצאות משמאל לישר l, וכל הנקודות ב־P_R נמצאות מימין לישר l. המערך X מחולק למערכים X_L ו־X_R, המכילים את הנקודות של P_L ו־P_R, בהתאמה, ממוינות בסדר מונוטוני עולה של קואורדינטות ה־x שלהן. באותו אופן, המערך Y מחולק למערכים Y_L ו־Y_R, המכילים את הנקודות של P_L ו־P_R, בהתאמה, ממוינות בסדר מונוטוני עולה של קואורדינטות ה־y שלהן.

משול: לאחר שחילקה את P ל־P_L ול־P_R, היא מבצעת שתי קריאות רקורסיביות, אחת כדי למצוא את זוג הנקודות הקרובות ביותר ב־P_L ואחת כדי למצוא את הזוג הקרוב ביותר ב־P_R. הקלטים לקריאה הראשונה הם תת־קבוצה P_L והמערכים X_L ו־Y_L, והקלטים לקריאה השנייה הם P_R, X_R ו־Y_R. יהיו δ_L ו־δ_R המרחקים בין הזוגות הקרובים ביותר שהוחזרו עבור P_L ו־P_R, בהתאמה. יהי $\delta = \min(\delta_L, \delta_R)$.

צרף: הזוג הקרוב ביותר הוא הזוג המרוחק מרחק δ שנמצא על־ידי אחת הקריאות הרקורסיביות או שהוא זוג נקודות שאחת מהן נמצאת ב־P_L והשנייה ב־P_R. האלגוריתם קובע אם ישנו זוג נקודות כאלה שהמרחק ביניהן קטן מ־δ. נשים לב שאם קיים זוג נקודות שהמרחק ביניהן קטן מ־δ, אזי שתי הנקודות חייבות להימצא בטווח של δ יחידות מן הישר l. לכן, כפי שניתן לראות באיור 35.12(i), שתי הנקודות חייבות להימצא ברצועה האנכית ברוחב 2δ שמרכזה הישר l. כדי למצוא זוג כזה, אם הוא קיים, האלגוריתם מבצע את הפעולות הבאות:

1. הוא יוצר מערך Y', שהוא המערך Y לאחר שהוצאו ממנו כל הנקודות שאינן נמצאות ברצועה האנכית ברוחב 2δ שמרכזה הישר l. המערך Y', בדיוק כמו המערך Y, ממוין על פי קואורדינטות ה־y של הנקודות.
2. עבור כל נקודה p במערך Y', האלגוריתם מנסה למצוא נקודות ב־Y' שמרחקן מ־p אינו עולה על δ יחידות. כפי שנראה מיד, די לבחון את 7 הנקודות העוקבות ל־p ב־Y'. האלגוריתם מחשב את המרחק בין p לבין כל אחת מ־7 הנקודות האלה, ושומר את המרחק δ' הקטן ביותר שמצא בין כל זוגות הנקודות ב־Y'.

3. אם $\delta' < \delta$, אזי הרצועה האנכית אכן מכילה זוג קרוב יותר מזה שנמצא על־ידי הקריאות הרקורסיביות. האלגוריתם מחזיר את זוג הנקודות הזה ואת המרחק ביניהן δ'. אחרת, האלגוריתם מחזיר את זוג הנקודות הקרובות ביותר ואת המרחק ביניהן δ, שנמצאו על־ידי הקריאות הרקורסיביות.



איור 35.12 מושגי מפתח בהוכחה שהאלגוריתם למציאת זוג הנקודות הקרובות ביותר יכול להסתפק בבדיקת 7 הנקודות העוקבות לכל נקודה במערך Y'. **(i)** אם הנקודות $p_L \in P_L$ ו־$p_R \in P_R$ מרוחקות זו מזו פחות מ־δ יחידות, הן חייבות להימצא בתוך מלבן בגודל $\delta \times 2\delta$ שמרכזו הישר l. **(ii)** האופן שבו יכולות 4 נקודות, שהמרחק בין כל שתיים מהן הוא לפחות δ יחידות, להימצא כולן בתוך ריבוע של $\delta \times \delta$. משמאל מופיעות 4 נקודות השייכות ל־P_L, ומימין — 4 נקודות השייכות ל־P_R. המלבן בגודל $\delta \times 2\delta$ יכול להכיל 8 נקודות אם הנקודות שעל הישר l הן למעשה זוגות של נקודות מתלכדות אשר אחת מהן שייכת ל־P_L והשנייה ל־P_R.

בתיאור שלעיל הושמטו כמה פרטי מימוש הנחוצים להשגת זמן ריצה של $O(n \lg n)$. לאחר שנכיח את נכונותו של האלגוריתם, נראה כיצד מממשים אותו כך שיושג החסם המבוקש על זמן הריצה.

נכונות

נכונותו של אלגוריתם זה למציאת זוג הנקודות הקרובות ביותר ברורה מאליה, פרט לשני היבטים. ראשית, על־ידי כך שאנו מתחילים "לעלות" מן הרקורסיה כאשר $|P| \le 3$, אנו מונעים מצב שבו ננסה לחלק קבוצת נקודות המכילה נקודה אחת בלבד. ההיבט השני הוא שניתן להסתפק בבדיקת 7 הנקודות העוקבות לכל נקודה p במערך Y', נוכיח עתה תכונה זו.

נניח שברמה מסוימת של הרקורסיה, זוג הנקודות הקרובות ביותר הוא $p_L \in P_L$ ו־$p_R \in P_R$. לפיכך, המרחק δ' בין p_L ו־p_R קטן ממש מ־δ. הנקודה p_L חייבת להימצא על הישר l או משמאלו במרחק של פחות מ־δ יחידות ממנו. באותו אופן, הנקודה p_R נמצאת על הישר l או מימינו במרחק של פחות מ־δ יחידות ממנו. יתר על כן, המרחק האנכי בין p_L ו־p_R אינו עולה על δ יחידות. לכן, כפי שמראה איור 35.12(i), p_L ו־p_R נמצאות בתוך מלבן של $\delta \times 2\delta$ שמרכזו הישר l. (מלבן זה עשוי להכיל נקודות נוספות.)

עתה נראה שבתוך מלבן זה בגודל $\delta \times 2\delta$ יכולות להימצא לכל היותר 8 נקודות של P. נתבונן בריבוע בגודל $\delta \times \delta$, המהווה את המחצית השמאלית של מלבן זה. מכיוון שכל הנקודות ב־P_L מרוחקות זו מזו לפחות δ יחידות, לכל היותר 4 נקודות יכולות להימצא בתוך ריבוע זה, איור

35.12(ii) מראה כיצד. באותו אופן, בריבוע בגודל $\delta \times \delta$ המהווה את המחצית הימנית של המלבן יכולות להימצא לכל היותר 4 נקודות של P_R. לכן, במלבן $2\delta \times \delta$ יכולות להימצא לכל היותר 8 נקודות של P. (נשים לב שמאחר שנקודות על הישר l יכולות להיות נקודות מ־P_L או נקודות מ־P_R, הרי שעל l יכולות להיות עד 4 נקודות. גבול זה מתקבל כאשר קיימים שני זוגות של נקודות מתלכדות, כך שכל זוג מורכב מנקודה אחת מ־P_L ונקודה אחת מ־P_R, וזוג אחד נמצא בנקודת החיתוך של l עם הצלע העליונה של המלבן, והזוג השני – בנקודת החיתוך של l עם הצלע התחתונה של המלבן.)

לאחר שהראינו כי בתוך המלבן יכולות להימצא לכל היותר 8 נקודות מ־P, קל לראות שניתן להסתפק בבדיקתן של 7 הנקודות העוקבות לכל נקודה במערך Y'. נניח עדיין כי הזוג הקרוב ביותר הן הנקודות p_L ו־p_R, ועוד נניח בלי הגבלת הכלליות כי p_L קודמת ל־p_R במערך Y'. אזי, גם אם p_L מופיעה במערך Y' מוקדם ככל האפשר ו־p_R מופיעה מאוחר ככל האפשר, p_R נמצאת באחד מ־7 המיקומים העוקבים ל־p_L. בכך נשלמה הוכחת נכונותו של האלגוריתם למציאת הזוג הקרוב ביותר.

מימוש וזמן ריצה

כפי שציינו, מטרתנו היא שנוסחת הנסיגה עבור זמן הריצה תהיה $T(n) = 2T(n/2) + O(n)$, כאשר $T(n)$ הוא, כמובן, זמן הריצה עבור קבוצה בת n נקודות. הקושי העיקרי הוא להבטיח שהמערכים X_L, X_R, Y_L ו־Y_R, המועברים לקריאות הרקורסיביות, יהיו ממוינים על פי הקואורדינטה המתאימה, וכן שהמערך Y' יהיה ממוין על פי קואורדינטת y. (נשים לב כי אם המערך X שמקבלת קריאה רקורסיבית כבר ממוין, אזי חלוקת הקבוצה P לתת־קבוצות P_L ו־P_R ניתנת לביצוע בנקל בזמן לינארי.)

הבחנת המפתח כאן היא שבכל קריאה, ברצוננו ליצור תת־קבוצה ממוינת של מערך ממוין. לדוגמה, קריאה מסוימת מקבלת את התת־קבוצה P ואת המערך Y, ממוין על פי קואורדינטת y. לאחר חלוקת P ל־P_L ו־P_R, הקריאה צריכה ליצור את המערכים Y_L ו־Y_R, כשהם ממוינים על פי קואורדינטת y. יתירה מזו, יש ליצור את המערכים הללו בזמן לינארי. על השיטה שבה עושים זאת ניתן להסתכל כעל היפוכה של השגרה MERGE ממיון־מיזוג שהוצג בסעיף 1.3.1: כאן מפצלים מערך ממוין לשני מערכים ממוינים. הפסאודו־קוד שלהלן מציג את הרעיון.

```
1  length[Y_L] ← length[Y_R] ← 0
2  for i ← 1 to length[Y]
3      do if Y[i] ∈ P_L
4          then length[Y_L] ← length[Y_L] + 1
5               Y_L[length[Y_L]] ← Y[i]
6          else length[Y_R] ← length[Y_R] + 1
7               Y_R[length[Y_R]] ← Y[i]
```

אנו פשוט בוחנים את הנקודות במערך Y על פי הסדר. אם נקודה $Y[i]$ שייכת ל־P_L, מוסיפים אותה לסוף המערך Y_L; אחרת, מוסיפים את הנקודה לסוף המערך Y_R. פסאודו־קוד דומה יוצר את המערכים X_L, X_R ו־Y'.

השאלה היחידה שנותרה היא כיצד למיין את הנקודות מלכתחילה. עושים זאת פשוט על־ידי **מיון מקדים** (presorting); כלומר, ממיינים אותן פעם אחת ולתמיד *לפני* הקריאה הרקורסיבית הראשונה. מערכים ממוינים אלה מועברים לקריאה הרקורסיבית הראשונה, ומשלב זה ואילך מפצלים ומקטינים אותם כנדרש במהלך הקריאות הרקורסיביות. המיון המקדים מוסיף לזמן הריצה הכולל זמן של $O(n \lg n)$, אבל עכשיו כל צעד של הרקורסיה, לא כולל הקריאות

הרקורסיביות המתבצעות במהלכו, מתבצע בזמן לינארי. לפיכך, אם $T(n)$ הוא זמן הריצה של כל אחד מהצעדים הרקורסיביים, ו־$T'(n)$ הוא זמן הריצה של האלגוריתם כולו, נקבל כי $T'(n) = T(n) + O(n \lg n)$, ו־$T(n)$ נתון על־ידי:

$$T(n) = \begin{cases} 2T(n/2) + O(n) & \text{אם } n > 3 \\ O(1) & \text{אם } n \le 3 \end{cases}$$

לכן, $T(n) = O(n \lg n)$ ו־$T'(n) = O(n \lg n)$.

תרגילים

35.4-1

פרופסור סמותריס מציע שיטה המאפשרת לאלגוריתם למציאת הזוג הקרוב ביותר לבדוק רק את 5 הנקודות העוקבות לכל נקודה במערך Y'. הרעיון הוא להכניס תמיד לקבוצה p_L את הנקודות שעל הישר l. כך לא יוכלו להיות על הישר l זוגות של נקודות מתלכדות שאחת מהן שייכת ל־p_L והאחרת שייכת ל־p_R. לכן, במלבן בגודל $2\delta \times \delta$ יוכלו להימצא לכל היותר 6 נקודות. מהו הליקוי בשיטתו של הפרופסור?

35.4-2

הראה כיצד ניתן להבטיח, בלי להגדיל את זמן הריצה האסימפטוטי של האלגוריתם, שקבוצת הנקודות המועברת לקריאה הרקורסיבית הראשונה לא תכיל שום נקודות מתלכדות. הוכח שבמקרה זה די לבדוק את 6 (ולא 7) הנקודות העוקבות לכל נקודה במערך Y'.

35.4-3

את המרחק בין שתי נקודות ניתן להגדיר גם בדרכים שאינן אוקלידיות. במישור, **מרחק־L_m** (L_m-distance) בין הנקודות p_1 ו־p_2 מוגדר על־ידי $((x_1 - x_2)^m + (y_1 - y_2)^m)^{1/m}$. מרחק אוקלידי הוא אפוא מרחק־L_2. שנה את האלגוריתם למציאת הזוג הקרוב ביותר כך שישתמש במרחק־L_1, הידוע גם כ**מרחק מנהטן** (Manhattan distance).

35.4-4

בהינתן שתי נקודות p_1 ו־p_2 במישור, מרחק־L_∞ ביניהן מוגדר על־ידי $\max(|x_1 - x_2|, |y_1 - y_2|)$. שנה את האלגוריתם למציאת הזוג הקרוב ביותר כך שישתמש במרחק־L_∞.

בעיות

35-1 שכבות קמורות

בהינתן קבוצה Q של נקודות במישור, אנו מגדירים את **השכבות הקמורות** (convex layers) של Q באופן אינדוקטיבי: השכבה הקמורה הראשונה של Q מורכבת מאותן נקודות ב־Q שהן קדקודים של $\mathrm{CH}(Q)$. עבור $i > 1$, נגדיר את Q_i כקבוצת הנקודות ב־Q שאינן שייכות לשכבות הקמורות $1, 2, \ldots, i-1$. השכבה הקמורה ה־i של Q היא אפוא $\mathrm{CH}(Q_i)$ אם $Q_i \neq \phi$, ואינה מוגדרת אם $Q_i = \phi$.

א. כתוב אלגוריתם שזמן ריצתו $O(n^2)$ למציאת השכבות הקמורות של קבוצה בת n נקודות.

ב. הוכח כי דרוש זמן של $\Omega(n \lg n)$ לחישוב השכבות הקמורות של קבוצה בת n נקודות בכל מודל חישובי שבו דרוש זמן של $\Omega(n \lg n)$ למיון n מספרים ממשיים.

35-2 שכבות מקסימליות

תהי Q קבוצה בת n נקודות במישור. אנו אומרים שנקודה (x, y) **שולטת** (dominates) על נקודה (x', y') אם $x \geq x'$ ו־$y \geq y'$. נקודה ב־Q שאינה נשלטת על־ידי אף אחת מהנקודות האחרות ב־Q נקראת **מקסימלית** (maximal). יש לשים לב כי Q עשויה להכיל נקודות מקסימליות רבות, שניתן לארגן אותן ב**שכבות מקסימליות** (maximal layers) באופן הבא. השכבה המקסימלית הראשונה L_1 היא שכבת הנקודות המקסימליות ב־Q. עבור $i > 1$, השכבה המקסימלית ה־i, L_i, היא קבוצת הנקודות המקסימליות ב־$Q - \bigcup_{j=1}^{i-1} L_j$.

נניח כי Q מכילה k שכבות מקסימליות לא ריקות, ועבור $i = 1, 2, \ldots, k$, תהי y_i קואורדינטת y של הנקודה השמאלית ביותר ב־L_i. לעת עתה נניח שלא קיימות ב־Q שתי נקודות בעלות אותה קואורדינטת x או אותה קואורדינטת y.

א. הראה כי $y_1 > y_2 > \cdots > y_k$.

נתבונן בנקודה (x, y) הנמצאת משמאל לכל נקודה אחרת ב־Q ואשר קואורדינטת ה־y שלה שונה מזו של כל נקודה אחרת ב־Q. תהי $Q' = Q \cup \{(x, y)\}$.

ב. יהי j האינדקס המינימלי שעבורו $y < y_j$, אלא אם כן $y < y_k$, ואז מציבים $j = k + 1$. הראה שהשכבות המקסימליות של Q' הן כדלקמן:

- אם $j \leq k$, אזי השכבות המקסימליות של Q' זהות לשכבות המקסימליות של Q, פרט לכך ש־L_j מכילה גם את (x, y) כנקודה השמאלית ביותר החדשה שלה.
- אם $j = k + 1$, אזי k השכבות המקסימליות הראשונות של Q' זהות לאלה של Q, אך בנוסף להן, Q' מכילה גם שכבה מקסימלית לא ריקה $(k + 1)$־ית: $L_{k+1} = \{(x, y)\}$.

ג. תאר אלגוריתם שזמן ריצתו $O(n \lg n)$ לחישוב השכבות המקסימליות של קבוצה Q בת n נקודות. (**רמז:** העבר קו סריקה מימין לשמאל.)

ד. האם מתעוררים קשיים כלשהם אם מרשים נקודות קלט בעלות אותה קואורדינטת x או אותה קואורדינטת y? הצע דרך לפתרון בעיות כאלה.

35-3 שדים ומכשפי שדים

קבוצה של n מכשפי שדים נלחמת ב־n שדים. כל מכשף שדים מצויד ברובה פרוטונים היורה זרם פרוטונים לעבר השד ומשמיד אותו. זרם הפרוטונים נע בקו ישר ומסתיים כאשר הוא פוגע בשד. מכשפי השדים מסכימים ביניהם על האסטרטגיה הבאה: כל מכשף יבחר לעצמו שד, כך שייווצרו n זוגות של שד ומכשף שדים, ואז יירו כל מכשפי השדים זרמי פרוטונים בבת אחת, כל אחד לעבר השד שלו. כפי שכולנו יודעים, הצטלבותם של זרמים מסוכנת מאוד, ולפיכך מכשפי השדים צריכים לבחור את בני זוגם השדים כך שהזרמים לא יצטלבו.

הנח שמיקומו של כל שד וכל מכשף שדים הוא נקודה קבועה במישור, ולא קיימות שלוש נקודות כאלה הנמצאות כולן על ישר אחד.

א. הראה כי קיים ישר העובר דרך מכשף שדים אחד ושד אחד כך שמספר מכשפי השדים בצד אחד של הישר שווה למספר השדים באותו צד. תאר כיצד ניתן למצוא ישר כזה בזמן $O(n \lg n)$.

ב. כתוב אלגוריתם שזמן ריצתו $O(n^2 \lg n)$ לזיווג מכשפי שדים ושדים באופן המונע הצטלבותם של זרמים.

35-4 התפלגויות בעלות קמורים דלילים

נתבונן בבעיה של חישוב הקמור של קבוצת נקודות במישור הנבחרת על פי איזושהי התפלגות אקראית ידועה. לעתים, תוחלת הגודל של קמור של n נקודות שנבחרו על פי התפלגות כזאת היא $O(n^{1-\epsilon})$ עבור איזשהו קבוע $\epsilon > 0$. התפלגות כזאת מכונה **התפלגות בעלת קמור דליל** (sparse-hulled distribution). התפלגויות בעלות קמורים דלילים כוללות:

- נקודות הנבחרות על פי התפלגות אחידה מעיגול ברדיוס יחידה. תוחלת גודלו של הקמור היא $\Theta(n^{1/3})$.
- נקודות הנבחרות על פי התפלגות אחידה מן הפנים של מצולע קמור בעל k צלעות, עבור קבוע k כלשהו. תוחלת גודלו של הקמור היא $\Theta(\lg n)$.
- נקודות הנבחרות על פי התפלגות נורמלית דו־ממדית. תוחלת גודלו של הקמור היא $\Theta(\sqrt{\lg n})$.

א. בהינתן שני מצולעים קמורים בעלי n_1 ו־n_2 קדקודים, בהתאמה, הראה כיצד מחשבים את הקמור של כל $n_1 + n_2$ הנקודות בזמן $O(n_1 + n_2)$. (המצולעים אינם בהכרח זרים.)

ב. הראה שאת הקמור של קבוצה בת n נקודות הנבחרות באופן בלתי תלוי על פי התפלגות בעלת קמור דליל ניתן לחשב בתוחלת זמן של $O(n)$. (**רמז:** מצא באופן רקורסיבי את הקמורים של $n/2$ הנקודות הראשונות ושל $n/2$ הנקודות הנותרות ואז צרף את התוצאות.)

הערות ביבליוגרפיות לפרק 35

פרק זה בקושי מגרד את פני השטח של נושא האלגוריתמים והשיטות בגיאומטריה חישובית. בין הספרים העוסקים בגיאומטריה חישובית נמנים Preparata & Shamos [160] ו־Edelsbrunner [60].

אף כי הגיאומטריה נלמדה כבר בעת העתיקה, פיתוחם של אלגוריתמים לפתרון בעיות גיאומטריות הוא חדש יחסית. Preparata & Shamos מציינים כי המושג המוקדם ביותר בדבר מורכבותה של בעיה הועלה על־ידי E. Lemoine בשנת 1902. Lemoine חקר בניות אוקלידיות — כאלה שניתן לבצען באמצעות סרגל ומחוגה — והמציא קבוצה של חמש פעולות יסוד: הצבת רגל אחת של המחוגה על נקודה נתונה, הצבת רגל אחת של המחוגה על ישר נתון, שרטוט מעגל, העברת הסרגל דרך נקודה נתונה, ושרטוט ישר. Lemoine התעניין בשאלה כמה פעולות יסוד דרושות לביצוע בנייה נתונה. למספר זה קרא "מדד הפְּשָטוּת" של הבנייה.

האלגוריתם שבסעיף 35.2, הקובע אם ישנם קטעים נחתכים, הוא של Shamos & Hoey [176].

הגרסה המקורית של הסריקה של גראהם מופיעה אצל Graham [91]. אלגוריתם עטיפת החבילה הוא של Jarvis [112]. באמצעות מודל חישובי של עץ החלטה הוכיח Yao [205] את התחסם התחתון $\Omega(n \lg n)$ על זמן הריצה של אלגוריתם כלשהו למציאת קמור. כאשר מביאים בחשבון את מספר הקדקודים בקמור, h, אלגוריתם הגיזום־והחיפוש של Kirkpatrick & Seidel [120] שרץ בזמן $O(n \lg h)$ הוא אופטימלי אסימפטוטית.

האלגוריתם בשיטת הפרד־ומשול למציאת הזוג הקרוב ביותר, שרץ בזמן $O(n \lg n)$, הוא של Shamos ומופיע אצל Preparata & Shamos [160]. Preparata & Shamos גם מראים שאלגוריתם זה הוא אופטימלי אסימפטוטית במודל עץ־החלטה.

36 שלמות ב־NP

כל האלגוריתמים שחקרנו עד כה היו **בעלי זמן ריצה פולינומיאלי** (polynomial-time): זמן הריצה שלהם במקרה הגרוע על קלטים בגודל n הוא $O(n^k)$, עבור איזשהו קבוע k. לפעמים נקצר ונקרא לאלגוריתמים כאלה פשוט **אלגוריתמים פולינומיאליים**. באופן טבעי מתעוררת השאלה אם כל הבעיות ניתנות לפתרון בזמן פולינומיאלי. התשובה היא לא. לדוגמה, קיימות בעיות כגון "בעיית העצירה" המפורסמת של טיורינג (Turing), שלא ניתן כלל לפותרן באמצעות מחשב, ואין זה משנה כמה זמן עומד לרשותנו. ישנן גם בעיות שניתן לפותרן, אך לא בזמן $O(n^k)$ עבור אף אחד מהקבועים k. ככלל, בעיות שניתן לפותרן על־ידי אלגוריתמים פולינומיאליים נתפשות כבעיות הניתנות לפתרון סביר, או בעיות סבירות, ואילו בעיות אשר פתרונן דורש זמן על־פולינומיאלי נתפשות כבעיות שאינן ניתנות לפתרון סביר, או בעיות בלתי־סבירות.

אבל הנושא של פרק זה הוא מחלקה מעניינת של בעיות המכונות "בעיות NP־שלמות", שמעמדן אינו ידוע. מצד אחד, עבור אף אחת מן הבעיות ה־NP־שלמות לא נתגלה עדיין אלגוריתם פולינומיאלי, אך מצד שני, איש עדיין לא הצליח להוכיח אפילו עבור אחת מהן חסם תחתון על־פולינומיאלי. שאלה זו, המוכרת כשאלת $P \neq NP$, שהועלתה לראשונה ב־1971, היתה ועודנה אחת מבעיות המחקר הפתוחות העמוקות והמורכבות ביותר במדעי המחשב התיאורטיים.

רוב החוקרים בתחום מדעי המחשב התיאורטיים מאמינים שהבעיות ה־NP־שלמות אינן ניתנות לפתרון סביר. הסיבה לכך היא שאם ניתן לפתור רק בעיה NP־שלמה אחת בזמן פולינומיאלי, אזי לכל בעיה NP־שלמה קיים אלגוריתם זמן־פולינומיאלי. בהתחשב במגוון הרחב של בעיות NP־שלמות שנחקרו עד היום, ללא התקדמות כלשהי לקראת פתרון זמן־פולינומיאלי, תהיה בכך משום הפתעה אמיתית אם יתברר כי כולן ניתנות לפתרון בזמן פולינומיאלי.

כדי שאדם יהיה מתכנן אלגוריתמים טוב, עליו להבין את יסודות התיאוריה של שלמות ב־NP. אם ניתן לקבוע שבעיה מסוימת הינה NP־שלמה, זוהי ראיה טובה לכך שהבעיה אינה ניתנת לפתרון סביר. במקרה כזה, ייטיב מהנדס תוכנה לעשות אם ישקיע מאמצים בפיתוח אלגוריתם קירוב (ראה פרק 37) במקום לחפש אלגוריתם מהיר לפתרון מדויק של הבעיה. יתירה מזו, בעיות טבעיות ומעניינות רבות, אשר במבט ראשון אינן נראות קשות יותר ממיון, חיפוש בגרף או זרימה ברשת, הן למעשה בעיות NP־שלמות. לכן חשוב להכיר את מחלקת הבעיות המיוחדת הזאת.

בפרק זה נחקור אותם היבטים של שלמות ב־NP שיש להם השלכות הישירות ביותר על ניתוח אלגוריתמים. בסעיף 36.1 נציג הגדרה פורמלית של מושג ה"בעיה" ונגדיר את מחלקת הסיבוכיות P של בעיות הכרעה הניתנות לפתרון בזמן פולינומיאלי. כמו כן נראה כיצד משתלבים מושגים אלה במסגרת תורת השפות הפורמליות. בסעיף 36.2 נגדיר את המחלקה NP — מחלקת בעיות ההכרעה שפתרונותיהן ניתנים לאימות בזמן פולינומיאלי. כמו כן נציג באופן פורמלי את שאלת $P \neq NP$.

בסעיף 36.3 נראה כיצד ניתן לחקור קשרים בין בעיות באמצעות "רדוקציות" זמן־פולינומיאלי. נגדיר שלמות ב־NP ונציג בקווים כלליים הוכחה לכך שבעיה אחת, המכונה "ספיקות מעגל", היא NP־שלמה. לאחר שמצאנו בעיה NP־שלמה אחת, נראה בסעיף 36.4 כיצד ניתן להוכיח ביתר קלות את השלמות ב־NP של בעיות אחרות בשיטת הרדוקציות. את השיטה נדגים על־ידי כך שנוכיח כי שתי בעיות של ספיקות נוסחאות הן שלמות ב־NP. בסעיף 36.5 נוכיח את שלמותן ב־NP של מגוון בעיות אחרות.

36.1 זמן פולינומיאלי

את חקירת השלמות ב־NP נפתח בכך שנסח באופן פורמלי את המושג של בעיות הניתנות לפתרון בזמן פולינומיאלי. בעיות אלה נתפשות בדרך כלל כבעיות סבירות, אבל בשל סיבות פילוסופיות ולאו דווקא מתמטיות, אנו יכולים להציע שלושה טיעונים התומכים בתפישה זו.

ראשית, אף כי הגיוני הדבר לראות בעיה שפתרונה דורש זמן של $\Theta(n^{100})$ כבעיה בלתי־סבירה, הרי שקיימות רק בעיות מעשיות מעטות מאוד שהזמן הנדרש לפתרונן הוא בסדר גודל של פולינום ממעלה כה גבוהה. הבעיות החישוביות הניתנות לפתרון בזמן פולינומיאלי שנתקלים בהן באופן מעשי דורשות בדרך כלל זמן קצר בהרבה.

שנית, עבור מודלים חישוביים סבירים רבים, בעיה שניתן לפותרה בזמן פולינומיאלי במודל אחד, ניתן לפותרה בזמן פולינומיאלי גם במודל אחר. לדוגמה, מחלקת הבעיות הניתנות לפתרון בזמן פולינומיאלי על־ידי מכונת גישה־אקראית סדרתית, המודל ששימש אותנו ברובו של ספר זה, זהה למחלקת הבעיות הניתנות לפתרון בזמן פולינומיאלי על־ידי מכונות טיורינג מופשטות[1]. היא זהה גם למחלקת הבעיות הניתנות לפתרון בזמן פולינומיאלי על מחשב מקבילי, גם אם מספר המעבדים גדל פולינומיאלית עם גודל הקלט.

שלישית, מחלקת הבעיות הניתנות לפתרון בזמן פולינומיאלי היא בעלת תכונות סגירות נאות, שכן פולינומים סגורים תחת חיבור, כפל והרכבה. לדוגמה, אם הפלט של אלגוריתם פולינומיאלי אחד מוזן כקלט לאלגוריתם פולינומיאלי אחר, האלגוריתם המורכב המתקבל גם הוא פולינומיאלי. אם אלגוריתם שהוא פולינומיאלי מכל היבט אחר, מבצע מספר קבוע של קריאות לשגרות זמן־פולינומיאלי, זמן הריצה של האלגוריתם המורכב גם הוא פולינומיאלי.

בעיות מופשטות

כדי להבין את מחלקת הבעיות הפתירות בזמן פולינומיאלי, עלינו לגבש תחילה תפישה פורמלית של מושג ה"בעיה". אנו מגדירים **בעיה מופשטת** (abstract problem) Q כיחס בינרי בין קבוצה I של **מופעים** (instances) של בעיה וקבוצה S של **פתרונות** (solutions) לבעיה. לדוגמה, נתבונן בבעיה SHORTEST-PATH של מציאת המסלול הקצר ביותר בין שני קדקודים נתונים בגרף בלתי־מכוון ובלתי־משוקלל $G = (V, E)$. מופע של SHORTEST-PATH הוא שלשה המורכבת מגרף ושני קדקודים. פתרון הוא סדרת קדקודים בגרף, כאשר הסדרה הריקה אולי מציינת כי לא קיים שום מסלול. הבעיה SHORTEST-PATH עצמה היא היחס הקושר כל מופע של גרף ושני קדקודים למסלול קצר ביותר המחבר את שני הקדקודים בגרף. מכיוון שמסלולים קצרים ביותר אינם בהכרח יחידים, למופע נתון של בעיה עשויים להיות כמה פתרונות.

1. דיון מקיף במודל מכונת־טיורינג אפשר למצוא אצל Hopcroft & Ullman [104] או אצל Lewis & Papadimitriou [139].

$O(T(n))$ ואת הפלט שלה ניתן לחשב בזמן $O(T(n))$ אז קיים מספר קבוע k כך שלכל n ולכל קלט x באורך n מתקיים $T(n) \leq n^k$.

2. אם $T(n)$ ו־$S(n)$ הן פונקציות כך שלכל n ולכל קלט x באורך n מתקיים $T(n) \leq S(n)$, אז כל שפה שניתן להכריע בזמן $T(n)$ ניתן להכריע גם במקום $S(n)$.

3. אם $T(n)$ ו־$S(n)$ הן פונקציות כך שלכל n ולכל קלט x באורך n מתקיים $T(n) \leq S(n)$, אז כל שפה שניתן להכריע בזמן $T(n)$ ניתן להכריע גם במקום $S(n)$.

תזכורת שפה L נמצאת ב־P אם קיים אלגוריתם A ומספר קבוע k כך שלכל $x \in \{0,1\}^*$ האלגוריתם A עוצר אחרי $O(n^k)$ צעדים ומחזיר $L(x)$.

קבוע k

משפט 36.1

תהי Q בעיית הכרעה משפטית על קבוצת משפטים I ויהיו e_1, e_2 קידודים משפטיים של I. אם $Q \in \mathrm{P}$ אז גם $e_1(Q) \in \mathrm{P}$ ו־$e_2(Q) \in \mathrm{P}$.

עתה נוכל להגדיר באופן פורמלי את **מחלקת הסיבוכיות P** (complexity class P) בקבוצת הבעיות החלטיות שניתן לפתור בזמן פולינומיאלי.

הגדרה 36.2

מחלקת הסיבוכיות P היא קבוצת כל בעיות ההחלטה שניתן לפתור בזמן פולינומיאלי על ידי אלגוריתם דטרמיניסטי. כלומר,

$$\mathrm{P} = \bigcup_{k=0}^{\infty} \mathrm{DTIME}(n^k)$$

כאשר $\mathrm{DTIME}(n^k)$ היא קבוצת כל בעיות ההחלטה שניתן לפתור על ידי אלגוריתם דטרמיניסטי בזמן $O(n^k)$.

דוגמה 36.3

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v שייכת ל־P. אלגוריתם דייקסטרה פותר בעיה זו בזמן $O(V^2)$ כאשר V הוא מספר הקודקודים בגרף. אלגוריתם זה רץ בזמן פולינומיאלי ולכן בעיית המסלול הקצר ביותר שייכת ל־P.

דוגמה 36.4

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף.

דוגמה 36.5

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.6

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.7

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.8

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.9

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.10

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.11

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.12

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.13

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.14

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.15

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.16

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.17

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.18

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.19

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.20

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.21

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.22

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.23

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.24

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.25

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.26

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.27

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.28

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.29

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.30

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.31

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.32

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.33

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.34

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.35

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.36

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.37

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.38

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.39

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.40

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.41

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.42

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.43

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.44

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.45

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.46

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.47

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.48

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.49

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.50

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.51

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.52

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.53

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.54

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.55

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.56

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.57

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.58

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.59

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.60

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.61

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.62

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.63

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.64

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.65

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.66

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.67

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.68

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.69

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.70

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.71

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.72

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשתות בגרף. לאחר מכן ניתן לבדוק האם המסלול שנמצא קצר מ־k או לא.

דוגמה 36.73

בעיית המסלול הקצר ביותר בגרף G בין שני קודקודים u ו־v עם משקלים שלמים על הקשתות ומספר שלם k שייכת ל־P. אלגוריתם בלמן־פורד פותר בעיה זו בזמן $O(VE)$ כאשר V הוא מספר הקודקודים ו־E הוא מספר הקשת

$n = |e_2(i)|$. כדי לפתור את הבעיה $e_2(Q)$ על הקלט $e_2(i)$, אנו מחשבים תחילה את $e_1(i)$ ואז מריצים את האלגוריתם עבור $e_1(Q)$ על $e_1(i)$. כמה זמן נמשך תהליך זה? המרת הקידודים מתבצעת בזמן $O(n^c)$, ולכן $|e_1(i)| = O(n^c)$, שכן הפלט של מחשב סדרתי אינו יכול להיות ארוך יותר מזמן הריצה. פתרון הבעיה על $e_1(i)$ מתבצע בזמן $O(|e_1(i)|^k) = O(n^{ck})$ שהוא זמן פולינומיאלי, מאחר ש־c ו־k הם קבועים. ■

אם כן, בין אם המופעים של בעיה מופשטת מקודדים בבסיס בינרי או בבסיס 3, אין לכך השפעה על "סיבוכיותה" של הבעיה, דהיינו, על השאלה אם היא פתירה בזמן פולינומיאלי או לא; אבל אם המופעים מקודדים בקידוד אונרי, הסיבוכיות עשויה להשתנות. כדי שנוכל לערוך דיון בלתי־תלוי בקידוד, נניח בדרך כלל שמופעי בעיה מקודדים באיזושהי שיטה סבירה ותמציתית, אלא אם כן נציין במפורש אחרת. ביתר דיוק, נניח כי הקידוד של מספר שלם קשור פולינומיאלית לייצוגו הבינרי, וכי הקידוד של קבוצה סופית קשור פולינומיאלית לקידודה כרשימת איבריה, כשהם מופרדים זה מזה בפסיקים ומוקפים בצומדיים. (ASCII היא שיטת קידוד אחת כזאת.) כשבידינו שיטת קידוד "סטנדרטית" כזאת, נוכל ליצור קידודים סבירים של עצמים מתמטיים אחרים, כגון m־יות, גרפים ונוסחאות. נסמן קידוד סטנדרטי של עצם על־ידי הקפת העצם בסוגריים זוויתיים. כך, $\langle G \rangle$ יסמן את הקידוד הסטנדרטי של גרף G.

כל עוד נשתמש בקידוד הקשור פולינומיאלית לקידוד סטנדרטי זה, נוכל לדון ישירות בבעיות מופשטות בלי להתייחס לקידוד מסוים כלשהו, ביודענו כי בחירת הקידוד אינה משפיעה על השאלה אם בעיה מופשטת פתירה בזמן פולינומיאלי או לא. מכאן ואילך נניח בדרך כלל שכל המופעים של בעיה הם מחרוזות בינריות שקודדו באמצעות הקידוד הסטנדרטי, אלא אם כן נציין במפורש אחרת. כמו כן, נותר ברוב המקרים על ההבחנה בין בעיות מופשטות לבעיות קונקרטיות. עם זאת, על הקורא להיזהר מפני בעיות המתעוררות ביישומים מעשיים, שבהם השימוש בקידוד הסטנדרטי אינו מובן מאליו ויש חשיבות לשיטת הקידוד.

מסגרת של שפות פורמליות

אחד ההיבטים הנוחים של ההתמקדות בבעיות הכרעה הוא, שהן מאפשרות שימוש קל במנגנונים של תורת השפות הפורמליות. בנקודה זו כדאי לחזור על כמה הגדרות בתורה זו. **אלפבית** (alphabet) Σ הוא קבוצה סופית של סימנים. **שפה** (language) L מעל אלפבית Σ היא קבוצה כלשהי של מחרוזות סופיות המורכבות מסימנים השייכים ל־Σ. לדוגמה, אם $\Sigma = \{0, 1\}$, אזי הקבוצה $L = \{10, 11, 101, 111, 1011, 1101, 10001, \ldots\}$ היא שפת הייצוגים הבינריים של המספרים הראשוניים. את **המחרוזת הריקה** (empty string) מסמנים על־ידי ε, ואת **השפה הריקה** (empty laguage) על־ידי ϕ. שפת כל המחרוזות הסופיות מעל Σ מסומנת על־ידי Σ^*. לדוגמה, אם $\Sigma = \{0, 1\}$, אזי $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000 \ldots\}$ היא קבוצת כל המחרוזות הבינריות. כל שפה L מעל Σ היא תת־קבוצה של Σ^*.

קיים מגוון של פעולות על שפות. פעולות על קבוצות, כגון **איחוד** (union) ו**חיתוך** (intersection), נובעות ישירות מהגדרות של תורת הקבוצות. את **המשלים** (complement) של L מגדירים על־ידי $\overline{L} = \Sigma^* - L$. **השרשור** (concatenation) של שתי שפות L_1 ו־L_2 הוא השפה:

$$L = \{x_1x_2 : x_1 \in L_1 \text{ וגם } x_2 \in L_2\}$$

הסגור (closure), או **כוכב קליני** (Kleene star), של שפה L הוא השפה:

$$L^* = \{\varepsilon\} \cup L \cup L^2 \cup L^3 \cup \cdots$$

כאשר L^k היא השפה המתקבלת משרשור L לעצמה k פעמים.

מנקודת המבט של תורת השפות, קבוצת המופעים של בעיית הכרעה Q כלשהי היא פשוט הקבוצה Σ^*, כאשר $\Sigma = \{0, 1\}$. מאחר ש־Q מאופיינת לחלוטין על־ידי אותם מופעי הבעיה היוצרים את התשובה 1 (כן), הרי שניתן לראות את Q כשפה L מעל $\Sigma = \{0, 1\}$ שעבורה:

$$L = \{x \in \Sigma^* : Q(x) = 1\}$$

לדוגמה, השפה המתאימה לבעיית ההכרעה PATH היא:

$$\text{PATH} = \{\langle G, u, v, k\rangle : G = (V, E) \text{ הוא גרף בלתי־מכוון},\ u, v \in V,\ k \geq 0 \text{ הוא מספר שלם},\ \text{וקיים ב־}G \text{ מסלול מ־}u \text{ אל } v \text{ שאורכו לכל היותר } k\}.$$

(מטעמי נוחות, נשתמש לפעמים באותו שם — PATH — במקרה זה — כדי לציין בעיית הכרעה וגם את השפה המתאימה לה.)

המסגרת של תורת השפות הפורמליות מאפשרת לנו לבטא באופן תמציתי את הקשר בין בעיות הכרעה לבין אלגוריתמים הפותרים אותן. אנו אומרים שאלגוריתם A **מקבל** (accepts) מחרוזת $x \in \{0, 1\}^*$ אם, בהינתן קלט x, האלגוריתם מפיק את הפלט $A(x) = 1$. השפה **המתקבלת** (accepted) על ידי אלגוריתם A היא הקבוצה $L = \{x \in \{0, 1\}^* : A(x) = 1\}$, כלומר, קבוצת המחרוזות שהאלגוריתם מקבל. אלגוריתם A **דוחה** (rejects) מחרוזת x אם $A(x) = 0$.

גם אם אלגוריתם A מקבל שפה L, האלגוריתם לא בהכרח ידחה מחרוזת $x \notin L$ הנמסרת לו כקלט. למשל, האלגוריתם עלול להיכנס ללולאה אינסופית. שפה L **מוכרעת** (decided) על־ידי אלגוריתם A אם כל מחרוזת בינרית מתקבלת או נדחית על־ידי האלגוריתם. שפה L **מתקבלת בזמן פולינומיאלי** (accepted in polynomial time) על־ידי אלגוריתם A אם עבור כל מחרוזת $x \in L$ באורך n, האלגוריתם מקבל את x בזמן $O(n^k)$ עבור איזשהו קבוע k. שפה L **מוכרעת בזמן פולינומיאלי** (decided in polynomial time) על־ידי אלגוריתם A אם עבור כל מחרוזת $x \in \{0, 1\}^*$ באורך n, האלגוריתם מכריע אם $x \in L$ בזמן $O(n^k)$, עבור איזשהו קבוע k. לכן, כדי לקבל שפה, אלגוריתם צריך להתייחס רק למחרוזות ב־L, אבל כדי להכריע שפה, עלינו לקבל או לדחות כל מחרוזת ב־$\{0,1\}^*$.

לדוגמה, השפה PATH יכולה להתקבל בזמן פולינומיאלי. אחד מאלגוריתמי הזמן־הפולינומיאלי המקבלים את השפה מחשב את המסלול הקצר ביותר מ־u אל v ב־G באמצעות חיפוש־לרוחב, ואז משווה ל־k את המרחק המתקבל. אם המרחק אינו עולה על k, האלגוריתם מפיק את הפלט 1 ועוצר. אחרת, האלגוריתם אינו עוצר לעולם. אלא שאלגוריתם זה אינו מכריע את PATH, שכן הוא אינו מפיק במפורש את הפלט 0 עבור מופעים שבהם המסלול הקצר ביותר ארוך מ־k. אלגוריתם הכרעה עבור PATH חייב לדחות במפורש מחרוזות בינריות אשר אינן שייכות ל־PATH. עבור בעיית הכרעה כמו PATH, קל לתכנן אלגוריתם הכרעה כזה. עבור בעיות אחרות, כגון בעיית העצירה של טיורינג, קיים אלגוריתם מקבל אבל לא קיים אלגוריתם הכרעה.

באופן בלתי־פורמלי ניתן להגדיר **מחלקת סיבוכיות** (complexity class) כקבוצת שפות אשר השייכות אליה נקבעת על פי **מדד סיבוכיות** (complexity measure), כגון זמן ריצה, של אלגוריתם הקובע אם מחרוזת נתונה x שייכת לשפה L. ההגדרה הפורמלית של מחלקת סיבוכיות היא טכנית מעט יותר; הקורא המעוניין יכול לעיין במאמרם המקורי של Hartmanis & Stearns [95].

בתוך מסגרת זו של תורת השפות הפורמליות, נוכל לתת הגדרה חלופית של מחלקת הסיבוכיות P:

$$\text{P} = \{L \subseteq \{0, 1\}^* : \text{קיים אלגוריתם } A \text{ המכריע את } L \text{ בזמן פולינומיאלי}\}$$

למעשה, P היא גם מחלקת השפות המתקבלות בזמן פולינומיאלי.

משפט 36.2

$\mathrm{P} = \{L : L$ מתקבלת על־ידי אלגוריתם זמן־פולינומיאלי$\}$

הוכחה מאחר שמחלקת השפות המוכרעות על־ידי אלגוריתמי זמן־פולינומיאלי היא תת־קבוצה של מחלקת השפות המתקבלות על־ידי אלגוריתמי זמן־פולינומיאלי, עלינו להראות רק שאם L מתקבלת על־ידי אלגוריתם זמן־פולינומיאלי, היא גם מוכרעת על־ידי אלגוריתם זמן־פולינומיאלי. תהי L השפה המתקבלת על־ידי איזשהו אלגוריתם זמן־פולינומיאלי A. אנו נשתמש בטיעון "הדמיה" קלאסי כדי לבנות אלגוריתם זמן־פולינומיאלי אחר, A', אשר מכריע את L. מכיוון ש־A מקבל את L בזמן $O(n^k)$ עבור איזשהו קבוע k, קיים גם קבוע c כך ש־A מקבל את L ב־$T = cn^k$ צעדים לכל היותר. עבור כל מחרוזת קלט x, האלגוריתם A' מדמה את פעולתו של A במהלך פרק הזמן T. עם סיום פרק הזמן T, האלגוריתם A' בוחן את התנהגותו של A. אם A קיבל את x, אזי A' מקבל את x על־ידי הפקת הפלט 1. אם A לא קיבל את x, אזי A' דוחה את x על־ידי הפקת הפלט 0. הנקודה של A' עבור הדמיית A אינה מגדילה את זמן הריצה ביותר מאשר גורם פולינומיאלי, ולכן A' הוא אלגוריתם זמן־פולינומיאלי אשר מכריע את L. ■

יש לשים לב כי ההוכחה של משפט 36.2 איננה הוכחה בונה. עבור שפה נתונה $L \in \mathrm{P}$, ייתכן שאיננו יודעים בפועל חסם על זמן הריצה של האלגוריתם A המקבל את L. אף על פי כן, אנו יודעים שחסם כזה קיים, ולכן אנו יודעים שקיים אלגוריתם A' אשר יכול לבדוק את החסם, גם אם ייתכן שלא נוכל למצוא בקלות את האלגוריתם A'.

תרגילים

36.1-1

הגדר את בעיית האופטימיזציה LONGEST-PATH-LENGTH כיחס הקושר בין כל מופע של גרף בלתי־מכוון ושני קדקודים לבין אורכו של המסלול הפשוט הארוך ביותר בין שני הקדקודים. הגדר את בעיית ההכרעה:

$\text{LONGEST-PATH} = \{\langle G, u, v, k\rangle : G(V, E)$ הוא גרף בלתי־מכוון, $u, v \in V$, $k \geq 0$ הוא מספר שלם, וקיים מסלול פשוט מ־u ל־v ב־G שאורכו לפחות $k\}$.

הראה שניתן לפתור את בעיית האופטימיזציה LONGEST-PATH-LENGTH בזמן פולינומיאלי אם ורק אם LONGEST-PATH $\in \mathrm{P}$.

36.1-2

כתוב הגדרה פורמלית עבור הבעיה של מציאת המעגל הפשוט הארוך ביותר בגרף בלתי־מכוון. מצא בעיית הכרעה קשורה. מצא את השפה המתאימה לבעיית ההכרעה.

36.1-3

מצא קידוד פורמלי של גרפים מכוונים כמחרוזות בינריות תוך שימוש בייצוג על־ידי מטריצת סמיכויות, ולאחר מכן תוך שימוש בייצוג על־ידי רשימות סמיכות. הראה כי שני הייצוגים קשורים פולינומיאלית.

36.1-4

האם אלגוריתם התכנון־הדינמי עבור גרסת 0-1 של בעיית התרמיל שהתבקשת לכתוב בתרגיל 17.2-2 הוא אלגוריתם זמן־פולינומיאלי? הסבר את תשובתך.

36.1-5

נניח כי קיימת שפה L שקיים עבורה אלגוריתם המקבל כל מחרוזת $x \in L$ בזמן פולינומיאלי, אבל רץ בזמן על־פולינומיאלי אם $x \notin L$. הראה כי ניתן להכריע את L בזמן פולינומיאלי.

36.1-6

הראה כי אלגוריתם, שהוא אלגוריתם זמן־פולינומיאלי מכל בחינה אחרת, אשר מבצע לכל היותר מספר קבוע של קריאות לשגרות זמן־פולינומיאלי, רץ בזמן פולינומיאלי, אבל אם מספר הקריאות לשגרות זמן־פולינומיאלי הוא פולינומיאלי, התוצאה עשויה להיות אלגוריתם שרץ בזמן אקספוננציאלי.

36.1-7

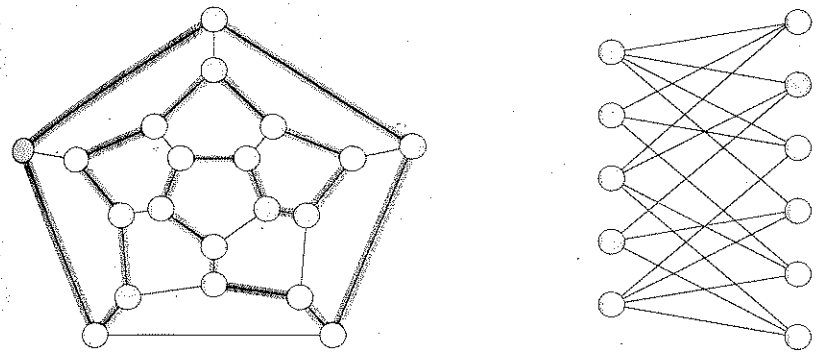
הראה שהמחלקה P, כאשר מסתכלים עליה כעת קבוצת שפות, סגורה תחת איחוד, חיתוך, שרשור, משלים וכוכב קליני (Kleene star). כלומר, אם $L_1, L_2 \in \mathrm{P}$, אזי $L_1 \cup L_2 \in \mathrm{P}$, וכו'.

36.2 אימות בזמן פולינומיאלי

עתה נתבונן באלגוריתמים ה"מאמתים" שייכות לשפה. לדוגמה, נניח שעבור מופע נתון $\langle G, u, v, k\rangle$ של בעיית ההכרעה PATH, נתון לנו גם מסלול p מ־u אל v. ניתן לבדוק בקלות אם אורכו של p הוא לכל היותר k, ואם כך הדבר, ניתן להסתכל על p כעל "אישור" לכך שהמופע הנתון אכן שייך ל־PATH. במקרה של בעיית ההכרעה PATH, ערכו של אישור זה אינו רב. אחרי ככלות הכל, PATH שייכת ל־P — למעשה ניתן לפתור את PATH בזמן לינארי — ולכן, אימות השייכות לשפה על סמך אישור נתון לא יתבצע בזמן קצר יותר מאשר פתרון הבעיה מההתחלה. עתה נבחן בעיה שעבורה איננו מכירים שום אלגוריתם הכרעה שרץ בזמן פולינומיאלי, ולמרות זאת, בהינתן אישור, האימות נעשה בקלות.

מעגלים המילטוניים

בעיית מציאתו של מעגל המילטוני בגרף בלתי־מכוון נחקרת כבר מאה שנה ויותר. באופן פורמלי, **מעגל המילטוני** (hamiltonian cycle) בגרף בלתי־מכוון $G = (V, E)$ הוא מעגל פשוט המכיל כל קדקוד ב־V. גרף המכיל מעגל המילטוני מכונה גרף **המילטוני** (hamiltonian); אחרת, הגרף מכונה **לא־המילטוני** (nonhamiltonian). Bondy & Murty [31] מצטטים מכתב מאת W. R. Hamilton המתאר משחק מתמטי על דריסרון (איור 36.1(i)) שבו אחד המשתתפים תוקע חמש סיכות בחמישה קדקודים עוקבים כלשהם, והמשתתף האחר צריך להשלים את המסלול ליצירת מעגל המכיל את כל הקדקודים. הדריסרון הוא גרף המילטוני ובאיור 36.1(i) מוצג מעגל המילטוני אחד בו. אבל לא כל הגרפים הם המילטוניים. לדוגמה, באיור 36.1(ii) מופיע גרף דו־צדדי בעל מספר אי־זוגי של קדקודים. (בתרגיל 36.2-2 ותבקש לראות שכל הגרפים מסוג זה הם לא־המילטוניים.)



(i) (ii)

איור 36.1 **(i)** גרף המייצג את הקדקודים, הצלעות והפיאות של תריסרון. הצלעות המוצללות יוצרות מעגל המילטוני. **(ii)** גרף דו־צדדי בעל מספר אי־זוגי של קדקודים. כל הגרפים מסוג זה הם לא המילטוניים.

את **בעיית המעגל ההמילטוני** (the hamiltonian-cycle problem), "האם לגרף G יש מעגל המילטוני?" ניתן להגדיר כשפה פורמלית:

$$\text{HAM-CYCLE} = \{\langle G\rangle : G \text{ הוא גרף המילטוני}\}$$

כיצד יכול אלגוריתם להכריע את השפה HAM-CYCLE? בהינתן מופע $\langle G\rangle$ של הבעיה, אלגוריתם הכרעה אפשרי אחד הוא זה הערוך רשימה של כל התמורות של קדקודי G ואז בודק כל תמורה כדי לראות אם היא מהווה מסלול המילטוני. מהו זמן הריצה של אלגוריתם זה? אם משתמשים בקידוד ה"סביר" של גרף על־ידי מטריצת הסמיכויות שלו, המספר m של קדקודים בגרף הוא $\Omega(\sqrt{n})$, כאשר $n = |\langle G\rangle|$ הוא אורך הקידוד של G. קיימות $m!$ תמורות אפשריות של הקדקודים, ולכן זמן הריצה הוא $\Omega(m!) = \Omega(\sqrt{m}!) = \Omega(2^{\sqrt{m}})$, שאיננו $O(n^k)$ עבור קבוע k כלשהו. לכן, אלגוריתם נאיבי זה אינו רץ בזמן פולינומיאלי, ולמעשה, בעיית המעגל ההמילטוני היא בעיה NP־שלמה, כפי שנוכיח בסעיף 36.5.

אלגוריתמי אימות

אולם נתבונן בבעיה קלה במעט. נניח שחבר מספר לך כי גרף נתון G הוא המילטוני, ומציע להוכיח זאת על־ידי כך שימנה את הקדקודים על פי סדרם לאורך המעגל ההמילטוני. ברור שיהיה קל למדי לאמת את ההוכחה: פשוט מאמתים כי המעגל המוצע הוא המילטוני על־ידי כך שבודקים אם קדקודיו הם תמורה של קדקודי V ואם כל אחת מן הקשתות העוקבות לאורך המעגל אכן קיימת בגרף. את אלגוריתם האימות הזה בודאי ניתן לממש כך שירוץ בזמן $O(n^2)$, כאשר n הוא אורך הקידוד של G. הוכחה שגרף מכיל מעגל המילטוני ניתנת אפוא לאימות בזמן פולינומיאלי.

נגדיר **אלגוריתם אימות** (verification algorithm) כאלגוריתם A המקבל כקלט שני ארגומנטים: ארגומנט אחד הוא מחרוזת קלט רגילה x, והארגומנט השני הוא מחרוזת בינרית y הנקראת **אישור** (certification). אלגוריתם A בעל שני ארגומנטים **מאמת** (verifies) מחרוזת קלט x אם

קיים אישור y כך ש־$A(x, y) = 1$. **השפה המאומתת** (verified language) על־ידי אלגוריתם אימות A היא:

$$L = \{x \in \{0,1\}^* : \text{קיים } y \in \{0,1\}^* \text{ כך ש־} A(x, y) = 1\}$$

אינטואיטיבית, אלגוריתם A מאמת שפה L אם עבור כל מחרוזת $x \in L$, קיים אישור y ש־A יכול להשתמש בו כדי להוכיח כי $x \in L$. יתירה מזו, עבור כל מחרוזת $x \notin L$, אסור שיהיה אישור המוכיח כי $x \in L$. לדוגמה, בבעיית המעגל ההמילטוני, האישור הוא רשימת הקדקודים המרכיבים את המעגל ההמילטוני. אם גרף הוא המילטוני, המעגל ההמילטוני עצמו מספק מידע שדי בו כדי לאמת עובדה זו. ולהיפך, אם גרף הוא לא־המילטוני, לא קיימת רשימת קדקודים שיכולה להטעות את אלגוריתם האימות ולגרום לו להאמין שהגרף המילטוני, שכן אלגוריתם האימות בודק בקפדנות את "המעגל" המוצע כדי לוודא שהוא אכן המילטוני.

מחלקת הסיבוכיות NP

מחלקת הסיבוכיות NP היא מחלקת השפות שניתן לאמתן באמצעות אלגוריתם זמן־פולינומיאלי.[4] ביתר דיוק, שפה L שייכת ל־NP אם ורק אם קיים אלגוריתם זמן־פולינומיאלי A בעל שני קלטים וקבוע c כך שמתקיים:

$$L = \{x \in \{0,1\}^* : \text{קיים אישור } y \text{ המקיים } |y| = O(|x|^c) \text{ כך ש־} A(x, y) = 1\}$$

אנו אומרים שהאלגוריתם A **מאמת** את השפה L **בזמן פולינומיאלי**.

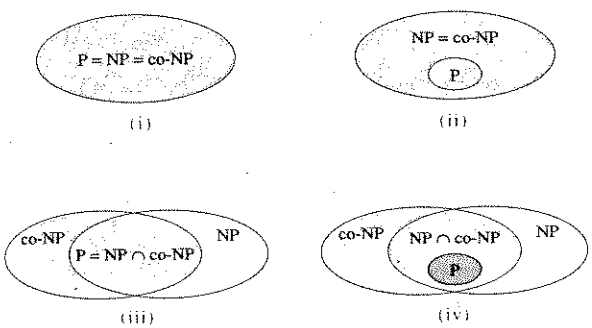
מן הדיון שלעיל בבעיית המעגל ההמילטוני עולה כי HAM-CYCLE $\in$ NP. (תמיד נחמד לדעת שקבוצה חשובה איננה ריקה.) יתר על כן, אם $L \in$ P, אזי $L \in$ NP, שכן אם קיים אלגוריתם זמן־פולינומיאלי המכריע את L, ניתן להפוך אותו בקלות לאלגוריתם אימות בעל שני קלטים שפשוט מתעלם מכל אישור ומקבל בדיוק אותן מחרוזות שלגביהן הוא קובע כי הן שייכות ל־L. לכן, P $\subseteq$ NP.

לא ידוע אם P = NP, אבל רוב החוקרים מאמינים כי P ו־NP הן מחלקות שונות. אינטואיטיבית, המחלקה P מורכבת מאותן בעיות הניתנות לפתרון במהירות. המחלקה NP מורכבת מבעיות שפתרון עבורן ניתן לאימות במהירות. ייתכן שלמדת מניסיונך כי לעתים קרובות קשה הרבה יותר לפתור בעיה מההתחלה, מאשר לאמת פתרון המוצג בבהירות, במיוחד אם יש לעבוד במסגרת אילוצי זמן. ככלל, תיאורטיקנים במדעי המחשב מאמינים בדרך כלל שניתן להחיל אנלוגיה זו על המחלקות P ו־NP, ולכן כי NP מכילה שפות שאינן שייכות ל־P.

קיימות עדות משכנעות יותר לכך ש־P $\neq$ NP — קיומן של שפות "ישלמות ב־NP". אנו נחקור מחלקה זו בסעיף 36.3.

שאלות יסודיות רבות אחרות, מעבר לשאלת P $\neq$ NP, נותרו עדיין בלתי פתורות. למרות עבודה מאומצת של חוקרים רבים, איש עדיין אינו יודע אפילו אם המחלקה NP סגורה תחת פעולת המשלים. כלומר, האם $L \in$ NP גורר $\overline{L} \in$ NP? נוכל להגדיר את **מחלקת הסיבוכיות co-NP** כקבוצת השפות L כך ש־$\overline{L} \in$ NP. את השאלה אם NP סגורה תחת פעולת המשלים ניתן לנסח מחדש כשאלה אם NP = co-NP. מאחר ש־P סגורה תחת פעולת המשלים (ראה תרגיל 36.1-7), נובע מכך כי P $\subseteq$ NP $\cap$ co-NP. אבל גם הפעם, לא ידוע אם NP $\cap$ co-NP = P או שקיימת איזושהי שפה השייכת ל־(NP $\cap$ co-NP) – P. איור 36.2 מתאר ארבעה תרחישים אפשריים.

4. "NP" הן ראשי התיבות של "nondeterministic polynomial-time". במקורה נחקרה המחלקה NP בהקשר של אלגוריתמים לא־דטרמיניסטיים, אולם בספר זה בחרנו להשתמש במושג הפשוט יותר, אך השקול, של אימות. אצל Hopcroft & Ullman [104] ניתן למצוא הצגה טובה של שלמות ב־NP במונחי מודלים חישוביים לא־דטרמיניסטיים.



איור 36.2 ארבעה תרחישים אפשריים ליחסים בין מחלקות סיבוכיות. בכל תרשים, אזור המוקף כולו על־ידי אזור אחר מצביע על תת־קבוצה ממש. **(i)** P = NP = co-NP. רוב החוקרים חושבים שזוהי האפשרות הפחות סבירה מכולן. **(ii)** אם NP סגורה תחת פעולת המשלים, אזי NP = co-NP, אולם אין ידוע שמתקיים P = NP. **(iii)** P = NP ∩ co-NP, אבל NP אינה סגורה תחת פעולת המשלים. **(iv)** NP ≠ co-NP וגם P ≠ NP ∩ co-NP. רוב החוקרים מאמינים כי זהי האפשרות הסבירה ביותר.

אם כן, הבנתנו את היחסים המדויקים בין P ל־NP לוקה בחסר באופן מצער. אף על פי כן, כאשר נחקור את תורת השלמות ב־NP נגלה שמנקודת מבט מעשית, החיסרון שבאי יכולתנו להוכיח כי בעיות הן סבירות אינו כה גדול כפי שניתן היה לשער.

תרגילים

36.2-1

נתבונן בשפה $\text{GRAPH-ISOMORPHISM} = \{\langle G_1, G_2\rangle : G_1$ ו־G_2 הם גרפים איזומורפיים$\}$. הוכח כי GRAPH-ISOMORPHISM $\in$ NP על־ידי כך שתתאר אלגוריתם זמן־פולינומיאלי לאימות השפה.

36.2-2

הוכח שאם G הוא גרף דו־צדדי בלתי־מכוון בעל מספר אי־זוגי של קדקודים, אזי G הוא לא־המילטוני.

36.2-3

הראה שאם HAM-CYCLE $\in$ P, אזי הבעיה של מניית קדקודיו של מעגל המילטוני על פי סדרם ניתנת לפתרון בזמן פולינומיאלי.

36.2-4

הוכח שמחלקת השפות NP סגורה תחת איחוד, חיתוך, שרשור וכוכב קליני (Kleene star). דון בסגור של NP תחת פעולת המשלים.

36.2-5

הראה שניתן להכריע כל שפה ב־NP באמצעות אלגוריתם שרץ בזמן $2^{O(n^k)}$ עבור איזשהו קבוע k.

36.2-6

מסלול המילטוני (hamiltonian path) בגרף הוא מסלול פשוט העובר דרך כל קדקוד בדיוק פעם אחת. הראה שהשפה:

$$\text{HAM-PATH} = \{\langle G, u, v\rangle : \text{קיים בגרף } G \text{ מסלול המילטוני מ־} u \text{ אל } v\}$$

שייכת ל־NP.

36.2-7

הראה שאת בעיית המסלול ההמילטוני עבור גרפים מכוונים ללא מעגלים ניתן לפתור בזמן פולינומיאלי. כתוב אלגוריתם יעיל הפותר את הבעיה.

36.2-8

תהי ϕ נוסחה בוליאנית המורכבת ממשתני הקלט הבוליאניים $x_1, x_2, \ldots, x_k$ מסימני הפעולות NOT ($\neg$), AND ($\wedge$) ו־OR ($\vee$), ומסוגריים. הנוסחה ϕ היא **טאוטולוגיה** (tautology) אם ערכה 1 עבור כל השמה של 0־ים ו־1־ים במשתני הקלט. הגדר את TAUTOLOGY כשפת הנוסחאות הבוליאניות המהוות טאוטולוגיות. הראה כי TAUTOLOGY $\in$ co-NP.

36.2-9

הוכח כי co-NP $\subseteq$ P.

36.2-10

הוכח שאם co-NP $\neq$ NP, אזי P $\neq$ NP.

36.2-11

יהי G גרף בלתי־מכוון קשיר המכיל לפחות 3 קדקודים, ויהי G^3 הגרף המתקבל על־ידי חיבור כל זוגות הקדקודים המחוברים על־ידי מסלול ב־G שאורכו לכל היותר 3. הוכח כי G^3 הוא גרף המילטוני. (**רמז:** בנה עץ פורש עבור G, והשתמש בטיעון אינדוקטיבי.)

36.3 שלמות ב־NP ורדוקציות

הסיבה המשכנעת ביותר אולי לכך שתיאורטיקנים במדעי המחשב מאמינים כי P $\neq$ NP היא קיומה של מחלקת הבעיות "השלמות ב־NP", או כפי שהן נקראות לפעמים "הבעיות ה־NP־שלמות". מחלקה זו היא בעלת התכונה המפתיעה שאם בעיה NP־שלמה *אחת* כלשהי ניתנת לפתרון בזמן פולינומיאלי, אזי לכל בעיה ב־NP יש פתרון זמן־פולינומיאלי, דהיינו, P = NP. אך למרות שנים רבות של מחקר, לא התגלה מעולם שום אלגוריתם זמן־פולינומיאלי עבור בעיה NP־שלמה כלשהי.

השפה HAM-CYCLE היא אחת הבעיות ה־NP־שלמות. אילו יכולנו להכריע את HAM-CYCLE בזמן פולינומיאלי, אזי היינו יכולים לפתור כל בעיה ב־NP בזמן פולינומיאלי. למעשה, אם יסתבר כי P – NP אינה ריקה, נוכל לומר בוודאות כי HAM-CYCLE $\notin$ P – NP.

השפות ה־NP־שלמות הן, במובן מסוים, השפות "הקשות" ביותר ב־NP. בסעיף זה נראה כיצד להשוות את "הקושי" היחסי של שפות תוך שימוש במושג מדויק הנקרא "רדוקציה בזמן פולינומיאלי". נגדיר תחילה באופן פורמלי שפות NP־שלמות, ולאחר מכן נביא הוכחה לכך ששפה

אחת כזאת, הנקראת CIRCUIT-SAT, היא NP־שלמה. בסעיף 36.5 נשתמש במושג הרדוקציה ונראה שבעיות רבות אחרות גם הן NP־שלמות.

רדוקציות

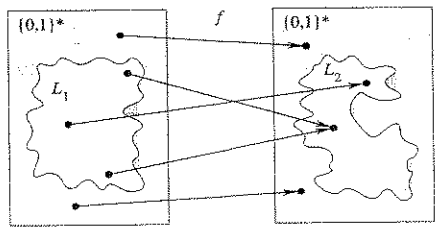
אינטואיטיבית, בעיה Q ניתנת לרדוקציה לבעיה אחרת Q' אם ניתן "לנסח מחדש" בקלות כל מופע של Q כמופע של Q' שפתרונו מספק פתרון למופע של Q. לדוגמה, הבעיה של פתרון משוואות לינאריות בעלם x ניתנת לרדוקציה לבעיה של פתרון משוואות ריבועיות. בהינתן מופע $ax + b = 0$, אנו ממירים אותו למשוואה הריבועית $0x^2 + ax + b = 0$, שפתרונה מספק פתרון למשוואה $ax + b = 0$. לפיכך, אם בעיה Q ניתנת לרדוקציה לבעיה אחרת Q', אזי במובן מסוים Q "אינה קשה יותר לפתרון" מאשר Q'.

אם נחזור למסגרת השפות הפורמליות עבור בעיות הכרעה, נאמר כי שפה L_1 **ניתנת לרדוקציה בזמן פולינומיאלי** (polynomial-time reducible) לשפה L_2, ונכתוב זאת $L_1 \leq_P L_2$, אם קיימת פונקציה $f : \{0,1\}^* \to \{0,1\}^*$ הניתנת לחישוב בזמן פולינומיאלי, כך שעבור כל $x \in \{0,1\}^*$ מתקיים:

$$x \in L_1 \text{ אם ורק אם } f(x) \in L_2 \qquad (36.1)$$

לפונקציה f נקרא **פונקציית הרדוקציה** (reduction function), ואלגוריתם זמן־פולינומיאלי F המחשב את f ייקרא **אלגוריתם רדוקציה** (reduction algorithm).

איור 36.3 מדגים את הרעיון של רדוקציית זמן־פולינומיאלי משפה L_1 לשפה אחרת L_2. כל שפה היא תת־קבוצה של $\{0,1\}^*$. פונקציית הרדוקציה f מספקת מיפוי זמן־פולינומיאלי כך שאם $x \in L_1$ אזי $f(x) \in L_2$. יתר על כן, אם $x \notin L_1$, אזי $f(x) \notin L_2$. לכן, פונקציית הרדוקציה ממפה כל מופע x של בעיית ההכרעה המיוצגת על־ידי השפה L_1 למופע $f(x)$ של הבעיה המיוצגת על־ידי L_2. תשובה לשאלה האם $f(x) \in L_2$ מספקת ישירות תשובה לשאלה האם $x \in L_1$.



איור 36.3 המחשה של רדוקציית זמן־פולינומיאלי משפה L_1 לשפה L_2 באמצעות פונקציית רדוקציה f. עבור כל קלט $x \in \{0,1\}^*$, לשאלה האם $x \in L_1$ יש אותה תשובה כמו לשאלה האם $f(x) \in L_2$.

רדוקציות זמן־פולינומיאלי מספקות כלי חזק להוכחת שייכותן של שפות שונות ל־P.

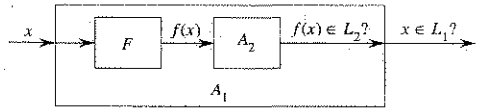
למה 36.3

אם $L_1, L_2 \subseteq \{0,1\}^*$ הן שפות המקיימות $L_1 \leq_P L_2$, אזי $L_2 \in P$ גורר $L_1 \in P$.

הוכחה יהי A_2 אלגוריתם זמן־פולינומיאלי המכריע את L_2, ויהי F אלגוריתם רדוקציה שרץ בזמן פולינומיאלי ומחשב את פונקציית הרדוקציה f. אנו נבנה אלגוריתם זמן־פולינומיאלי A_1 המכריע את L_1.

איור 36.4 מדגים את בנייתו של A_1. עבור קלט נתון $x \in \{0,1\}^*$, האלגוריתם A_1 משתמש ב־F כדי להמיר את x ל־$f(x)$ ואז משתמש ב־A_2 כדי לבדוק אם $f(x) \in L_2$. הפלט של A_2 הוא הערך שהתקבל כפלט מ־A_1.

נכונותו של A_1 נובעת מהתנאי (36.1). האלגוריתם רץ בזמן־פולינומיאלי משום ש־F וגם A_2 רצים בזמן פולינומיאלי (ראה תרגיל 36.1-6). ■



איור 36.4 הוכחת למה 36.3. האלגוריתם F הוא אלגוריתם רדוקציה שמחשב בזמן פולינומיאלי את פונקציית הרדוקציה f מ־L_1 ל־L_2. A_2 הוא אלגוריתם זמן־פולינומיאלי המכריע את L_2. באיור מתואר אלגוריתם A_1 המכריע אם $x \in L_1$ על־ידי כך שהוא משתמש ב־F כדי להמיר כל קלט x ל־$f(x)$ ואז משתמש ב־A_2 כדי להכריע אם $f(x) \in L_2$.

שלמות ב־NP

רדוקציות זמן־פולינומיאלי מספקות אמצעי פורמלי שבעזרתו אפשר להראות כי בעיה אחת קשה לפחות כמו בעיה אחרת, עד כדי גורם זמן־פולינומיאלי. כלומר, אם $L_2 \leq_P L_1$, אזי L_1 קשה יותר מ־L_2 בגורם פולינומיאלי לכל היותר, וזוהי הסיבה לכך שהשימוש בסימן "$\leq$" לציון רדוקציה מתאים. עתה אנו יכולים להגדיר את קבוצת השפות ה־NP־שלמות, שהן הבעיות הקשות ביותר ב־NP.

שפה $L \subseteq \{0,1\}^*$ היא **NP־שלמה** או **שלמה ב־NP** (NP-complete), אם:

1. $L \in NP$, וגם
2. $L' \leq_P L$ עבור כל $L' \in NP$.

אם שפה L מקיימת את תכונה 2 אולם לא בהכרח את תכונה 1, נאמר כי L **NP־קשה**, או **קשה ב־NP** (NP-hard). כמו כן נגדיר את NPC כמחלקת השפות ה־NP־שלמות.

כפי שמראה המשפט הבא, שלמות ב־NP מצויה בלב ליבה של ההכרעה אם P אכן שווה ל־NP.

משפט 36.4

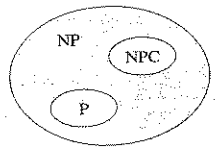
אם בעיה NP־שלמה כלשהי פתירה בזמן פולינומיאלי, אזי $P = NP$. באופן שקול, אם בעיה כלשהי ב־NP אינה פתירה בזמן פולינומיאלי, אזי שום בעיה NP־שלמה אינה פתירה בזמן פולינומיאלי.

הוכחה נניח כי $L \in P$ וכן כי $L \in NPC$. מתכונה 2 של השלמות ב־NP נובע שעבור כל $L' \in NP$ מתקיים $L' \leq_P L$. לכן, על פי למה 36.3, מתקיים גם $L' \in P$, ובכך הוכחה הטענה הראשונה במשפט.

להוכחת הטענה השנייה, נשים לב שהיא השלילה הלוגית של הטענה הראשונה. ■

זוהי הסיבה לכך שחקר שאלת $P \neq NP$ מתרכז סביב הבעיות ה־NP־שלמות. רוב החוקרים במדעי המחשב התיאורטיים מאמינים כי $P \neq NP$, ופירוש הדבר שהיחסים בין P, NP ו־NPC הם

אלה המתוארים באיור 36.5. אולם עדיין קיימת האפשרות שמישהו יגלה אלגוריתם זמן־פולינומיאלי עבור בעיה NP־שלמה, ובכך יוכיח כי P = NP. אף על פי כן, מכיוון שעד כה טרם התגלה אלגוריתם זמן־פולינומיאלי עבור בעיה NP־שלמה כלשהי, הרי שהוכחה כי בעיה היא NP־שלמה מספקת ראיה מצוינת לכך שהבעיה היא בלתי סבירה בסיבוכיותה.



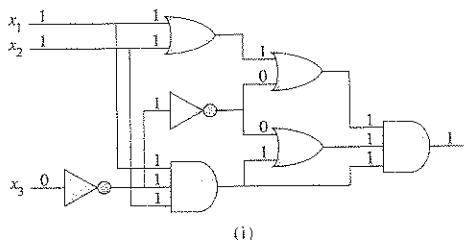
איור 36.5 הדרך שבה רואים רוב החוקרים במדעי המחשב התיאורטיים את היחסים בין P, NP ו־NPC. P וגם NPC מוכלות לחלוטין ב־NP ו־$\mathrm{NPC} = \phi$ ו־$\mathrm{NPC} \cap \mathrm{P} = \phi$.

ספיקות מעגלים

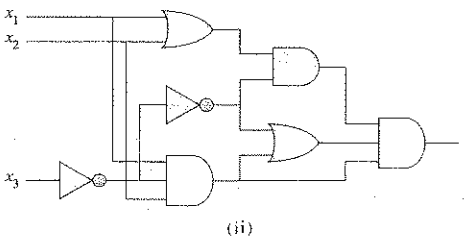
הגדרנו את המושג "בעיה NP־שלמה", אבל עד כה טרם הוכחנו למעשה כי בעיה כלשהי היא NP־שלמה. ברגע שנוכיח כי בעיה אחת לפחות היא NP־שלמה, נוכל להשתמש ברדוקציות זמן־פולינומיאלי ככלי להוכחת השלמות ב־NP של בעיות אחרות. נתמקד אפוא עתה בהוכחת קיומה של בעיה NP־שלמה: בעיית ספיקותם של מעגלים.

לרוע המזל, ההוכחה הפורמלית כי הבעיה של ספיקות מעגלים היא NP־שלמה דורשת פרטים טכניים החורגים מתחומו של ספר זה. תחת זאת נתאר באופן לא פורמלי הוכחה המסתמכת על הבנה בסיסית של מעגלים צירופיים בוליאניים. חומר זה נסקר בתחילתו של פרק 29.

באיור 36.6 מופיעים שני מעגלים צירופיים בוליאניים, כל אחד בעל שלושה קלטים ופלט יחיד. **השמת ערכי אמת** (truth assignment) עבור מעגל צירופי בוליאני היא קבוצה של ערכי קלט בוליאניים. אנו אומרים שמעגל צירופי בוליאני בעל פלט יחיד הוא **ספיק** (satisfiable) אם קיימת עבורו **השמה מספקת** (satisfying assignment): השמת ערכי אמת הגורמת לכך שערכו של פלט המעגל יהיה 1. לדוגמה, עבור המעגל שבאיור 36.6(i), ההשמה $\langle x_1 = 1, x_2 = 1, x_3 = 0 \rangle$ היא השמה מספקת, ולכן המעגל ספיק. עבור המעגל שבאיור 36.6(ii) לא קיימת השמה של ערכים עבור x_1 x_2 ו־x_3 הגורמת לכך שפלט המעגל יהיה 1; פלט המעגל הוא תמיד 0, ולכן המעגל בלתי־ספיק.



איור 36.6 שני מופעים של בעיית ספיקות המעגלים. **(i)** ההשמה $\langle x_1 = 1, x_2 = 1, x_3 = 0 \rangle$ עבור הקלטים של המעגל גורמת לכך שפלט המעגל יהיה 1. לכן המעגל ספיק. **(ii)** שום השמה עבור הקלטים של המעגל אינה יכולה לגרום לכך שפלט המעגל יהיה 1. לכן המעגל בלתי־ספיק.



איור 36.6 (המשך)

הבעיה של ספיקות מעגלים (circuit-satisfiability problem) היא זו: "בהינתן מעגל צירופי בוליאני המורכב משערי AND, OR ו־NOT, האם המעגל ספיק?" אבל כדי לנסח שאלה זו באופן פורמלי, עלינו להסכים על קידוד סטנדרטי של מעגלים. ניתן ליצור קידוד דמוי־גרף הממפה מעגל נתון כלשהו C למחרוזת בינרית $\langle C \rangle$ שאורכה אינו עולה בהרבה על גודלו של המעגל עצמו. כשפה פורמלית, נוכל אפוא להגדיר:

$$\text{CIRCUIT-SAT} = \{\langle C \rangle : C \text{ הוא מעגל צירופי בוליאני ספיק}\}$$

לבעיית הספיקות של מעגלים יש חשיבות רבה בתחום של אופטימיזציית חומרה בעזרת מחשב. אם מעגל תמיד מפיק 0, ניתן להחליפו במעגל פשוט יותר שאין בו כלל שערים לוגיים והוא מפיק כפלט את הקבוע 0. לאלגוריתם זמן־פולינומיאלי עבור בעיה זו יהיה שימוש מעשי רב.

בהינתן מעגל C, נוכל לנסות לקבוע אם הוא ספיק פשוט על־ידי בדיקת כל ההשמות האפשריות עבור ערכי הקלט. לרוע המזל, אם ישנם k קלטים, אזי מספר ההשמות האפשריות הוא 2^k. כאשר גודלו של C פולינומיאלי ב־k, בדיקת כל ההשמות האפשריות מובילה לאלגוריתם שרץ בזמן על־פולינומיאלי. כאמור, למעשה קיימות ראיות טובות לכך שלא קיים אלגוריתם זמן־פולינומיאלי הפותר את בעיית הספיקות של מעגלים, משום שבעיה זו היא NP־שלמה. את הוכחת העובדה הזאת נחלק לשני חלקים, המבוססים על שני חלקי ההגדרה של שלמות ב־NP.

למה 36.5
בעיית הספיקות של מעגלים שייכת למחלקה NP.

הוכחה אנו נציג אלגוריתם זמן־פולינומיאלי A בעל שני קלטים שיכול לאמת את השפה CIRCUIT-SAT. אחד הקלטים ל־A הוא (קידוד סטנדרטי של) מעגל צירופי בוליאני C. הקלט האחר הוא אישור המתאים להשמה של ערכים בוליאניים על תילי C.

האלגוריתם A בנוי כדלקמן. עבור כל שער לוגי במעגל, האלגוריתם בודק אם הערך שהשים האישור על תיל הפלט מחושב נכונה כפונקציה של הערכים על תילי הקלט. לאחר מכן, אם הפלט של המעגל כולו הוא 1, האלגוריתם מפיק את הפלט 1, שכן הערכים שהושמו לקלטים של C יוצרים השמה מספקת. אחרת, A מפיק את הפלט 0.

בכל פעם שהאלגוריתם A מקבל כקלט מעגל ספיק C, קיים אישור שאורכו פולינומיאלי בגודלו של C אשר גורם ל־A להפיק את הפלט 1. בכל פעם ש־A מקבל כקלט מעגל בלתי־ספיק, אף אישור אינו יכול להטעות את A ולגרום לו להאמין שהמעגל ספיק. A רץ בזמן פולינומיאלי: מימוש טוב ירוץ אפילו בזמן לינארי. לכן, ניתן לאמת את CIRCUIT-SAT בזמן פולינומיאלי, כלומר, $\text{CIRCUIT-SAT} \in \text{NP}$. ■

החלק השני בהוכחת שלמותה ב־NP של CIRCUIT-SAT הוא ההוכחה שהשפה היא NP־קשה. כלומר, עלינו להראות שכל שפה ב־NP ניתנת לרדוקציה ל־CIRCUIT-SAT בזמן־פולינומיאלי. ההוכחה המדויקת והמלאה של עובדה זו גדושה פרטים טכניים סבוכים, ולכן אנו נסתפק בתיאור כללי שלה, המבוסס על מידה מסוימת של הבנה באופן הפעולה של חומרת מחשב.

תכנית מחשב מאוחסנת בזיכרון המחשב כסדרה של הוראות. הוראה טיפוסית היא קידוד של פעולה לביצוע, כתובות של אופרנדים בזיכרון, וכתובת שבה תאוחסן התוצאה. מקום מיוחד בזיכרון, הנקרא **מונה התכנית** (program counter), מנהל מעקב איזו הוראה יש לבצע בצעד הבא. בכל פעם שנשלפת הוראה לשם ביצוע, ערך המונה גדל ב־1 באופן אוטומטי, ודבר זה גורם לכך שהמחשב יבצע את ההוראות באופן סדרתי. עם זאת, ביצוע הוראה עשוי לגרום לכתיבת ערך במונה התכנית, ואז עשוי להשתנות הביצוע הסדרתי הרגיל, כדי לאפשר למחשב לבצע לולאות או הסתעפויות מותנות.

בכל נקודה במהלך ביצועה של תכנית, מצב החישוב כולו מיוצג בזיכרון המחשב. (מבחינתנו, הזיכרון כולל את התכנית עצמה, את מונה התכנית, שטח עבודה ואת כל סיביות המצב השונות שמנהל המחשב לצורך "הנהלת חשבונות".) כל מצב מסוים של זיכרון המחשב נקרא **תצורה** (configuration). ניתן לראות ביצוע של הוראה כמיפוי תצורה אחת לאחרת. מה שחשוב הוא, שאת חומרת המחשב המבצעת מיפוי זה ניתן לממש כמעגל צירופי בוליאני, שנסמנו ב־M בהוכחת הלמה הבאה.

למה 36.6

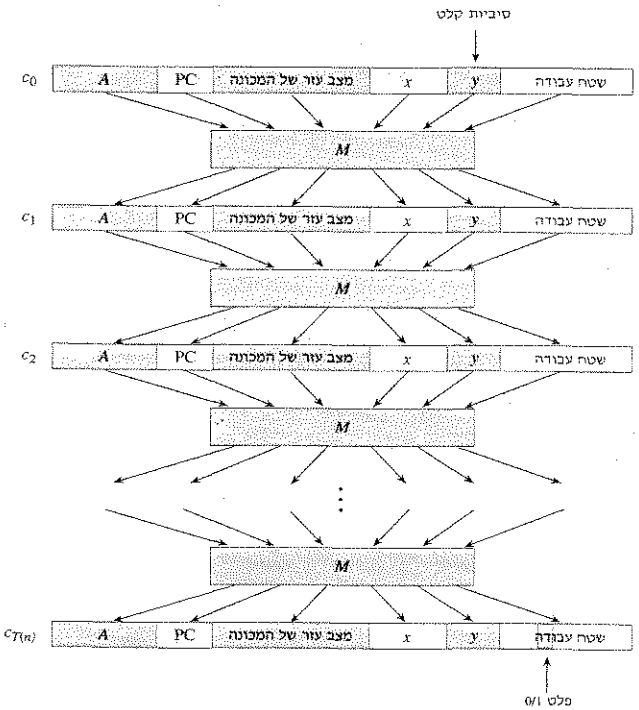
בעיית הספיקות של מעגלים היא NP־קשה.

הוכחה תהי L שפה כלשהי ב־NP. נתאר אלגוריתם זמן־פולינומיאלי F שמחשב פונקציית רדוקציה f הממפה כל מחרוזת בינרית x למעגל $C = f(x)$ כך ש־$x \in L$ אם ורק אם $C \in$ CIRCUIT-SAT.

מכיוון ש־$L \in$ NP, בהכרח קיים אלגוריתם A המאמת את L בזמן פולינומיאלי. האלגוריתם F שנבנה ישתמש באלגוריתם A בעל שני הקלטים לחישוב פונקציית הרדוקציה f.

נסמן ב־$T(n)$ את זמן הריצה במקרה הגרוע של האלגוריתם A על מחרוזות קלט באורך n, יהי $k \geq 1$ קבוע שעבורו $T(n) = O(n^k)$ ואורכו של האישור הוא $O(n^k)$. למעשה, זמן הריצה של A פולינומיאלי בגודל הכולל של הקלט, המורכב הן ממחרוזת הקלט והן מן האישור, אבל מאחר שאורך האישור פולינומיאלי באורך n של מחרוזת הקלט, זמן הריצה פולינומיאלי ב־n.)

הרעיון הבסיסי בהוכחה הוא לייצג את החישוב שמבצע A כסדרה של תצורות. כפי שמראה איור 36.7, ניתן לחלק כל תצורה לחלקים: התכנית עבור A, מונה התכנית (המסומן ב־PC, עבור program counter) ומצב עזר של המכונה, הקלט x, האישור y ושטח עבודה. החל מן התצורה ההתחלתית c_0, כל תצורה c_i ממופה לתצורה שאחריה c_{i+1} באמצעות המעגל הצירופי M, המממש את חומרת המחשב. עם סיום הרצתו של A, הפלט שלו — 0 או 1 — נכתב במקום המיועד לכך בשטח העבודה, ואם אנו מניחים ש־A עוצר מיד אחר כך, הרי שערך זה שוב אינו משתנה. לכן, אם האלגוריתם מבצע לכל היותר $T(n)$ צעדים, הפלט מופיע כאחת הסיביות בתצורה $c_{T(n)}$.



איור 36.7 סדרת התצורות שיוצר אלגוריתם A המוּרָץ על קלט x ואישור y. כל תצורה מייצגת את מצב המחשב עבור צעד אחד של החישוב. לבד מ־A, x ו־y, כל תצורה כוללת גם את מונה התכנית (PC), מצב עזר של המכונה ושטח עבודה. פרט לאישור y, התצורה ההתחלתית c_0 קבועה. כל תצורה ממופה אל התצורה הבאה באמצעות מעגל צירופי בוליאני M. הפלט הוא סיבית מסוימת בשטח העבודה.

אלגוריתם הרדוקציה F בונה מעגל צירופי יחיד אשר מחשב את כל התצורות שיוצרת תצורה התחלתית נתונה. הרעיון הוא להדביק ביחד $T(n)$ עותקים של המעגל M. הפלט של המעגל ה־i, היוצר את התצורה c_i, מוזן ישירות כקלט למעגל ה־$(i+1)$. כך, במקום שפלטי התצורות יאוחסנו באוגרי מצב, הם פשוט מתקבלים כערכים על התילים המחברים עותקים של M.

הבה ניזכר מה צריך לעשות אלגוריתם הרדוקציה הפולינומיאלי F. בהינתן קלט x, עליו לחשב מעגל $C = f(x)$ שהוא ספיק אם ורק אם קיים אישור y כך ש־$A(x, y) = 1$ כאשר F מקבל קלט x, הוא מחשב תחילה את $n = |x|$ ובונה מעגל צירופי C' המורכב מ־$T(n)$ עותקים של M. הקלט ל־C' הוא תצורה התחלתית המתאימה לחישוב על $A(x, y)$, והפלט הוא התצורה $c_{T(n)}$.

המעגל $C = f(x)$ שמחשב F מתקבל באמצעות שינוי קטן של C'. ראשית, משימים ישירות על התילים את הערכים הידועים של הקלטים של C' המתאימים לתכנית עבור A, את הערך ההתחלתי של מונה התכנית, את הקלט x ואת המצב ההתחלתי של הויכרון. לכן, הקלטים הנותרים היחידים למעגל הם אלה המתאימים לאישור y. שנית, מתעלמים מכל הפלטים של המעגל פרט לסיבית האחת של $c_{T(n)}$ המתאימה לפלט של A. המעגל C הבנוי באופן זה מחשב

את $C(y) = A(x, y)$ עבור כל קלט y באורך $O(n^k)$. אלגוריתם הרדוקציה F, כאשר הוא מקבל מחרוזת קלט x, מחשב מעגל C כזה ומציג אותו כפלט.

עדיין עלינו להוכיח שתי תכונות. ראשית, עלינו להראות ש־F מחשב נכונה פונקציית רדוקציה f. כלומר, עלינו להראות ש־C ספיק אם ורק אם קיים אישור y כך ש־$A(x, y) = 1$. שנית, עלינו להראות ש־F רץ בזמן פולינומיאלי.

כדי להראות ש־F מחשב נכונה פונקציית רדוקציה, הבה נניח שקיים אישור y באורך $O(n^k)$ כך ש־$A(x, y) = 1$. אזי, אם מזינים את סיביות y כקלטים ל־C, הפלט של C הוא $C(y) = A(x, y) = 1$. לכן, אם קיים אישור, C ספיק. ובכיוון ההפוך, נניח כי C ספיק. מכאן, שקיים קלט y ל־C כך ש־$C(y) = 1$, ומכך אנו מסיקים כי $A(x, y) = 1$. לכן, F מחשב נכונה פונקציית רדוקציה.

כדי להשלים את ההוכחה עלינו רק להראות שזמן ריצתו של F פולינומיאלי ב־$n = |x|$. ראשית, נשים לב כי מספר הסיביות הדרושות לייצוגה של תצורה הוא פולינומיאלי ב־n. התכנית עבור A עצמו היא בעלת גודל קבוע בלי תלות באורך הקלט x. אורך הקלט x הוא n, ואורך האישור y הוא $O(n^k)$. מכיוון שהאלגוריתם מבצע לכל היותר $O(n^k)$ צעדים, כמות שטח העבודה הדרוש עבור A גם היא פולינומיאלית ב־n. (אנו מניחים שזהו שטח רצוף בזיכרון. בתרגיל 36.3-4 תתבקש להרחיב את הטיעון למצב שבו תאי הזיכרון ש־A ניגש אליהם מפוזרים על פני שטח גדול בהרבה בזיכרון, ותבנית הפיזור המסוימת עשויה להשתנות עבור כל קלט x.)

גודלו של המעגל הצירופי M המממש את חומרת המחשב הוא פולינומיאלי באורכה של תצורה, ואורך זה פולינומיאלי ב־$O(n^k)$ ולכן פולינומיאלי ב־n. (רובו של מעגל זה משמש למימוש הלוגיקה של מערכת הזיכרון.) המעגל C מורכב מ־$t = O(n^k)$ עותקים של M. לכל היותר, ולכן גודלו פולינומיאלי ב־n. בהינתן x, ניתן לבנות את C בזמן פולינומיאלי באמצעות אלגוריתם הרדוקציה F, משום שכל צעד של הבנייה מתבצע בזמן פולינומיאלי. ■

הראינו אפוא שהשפה CIRCUIT-SAT קשה לפחות כמו כל שפה אחרת ב־NP, ומכיוון שהיא שייכת ל־NP, הרי שהיא NP־שלמה.

משפט 36.7

בעיית הספיקות של מעגלים היא NP־שלמה.

הוכחה נובעת ישירות מלמות 36.5 ו־36.6 ומהגדרת השלמות ב־NP. ■

תרגילים

36.3-1

הראה שהיחס $\le_p$ הוא יחס טרנזיטיבי על שפות. כלומר, הראה שאם $L_1 \le_p L_2$ ו־$L_2 \le_p L_3$, אזי $L_1 \le_p L_3$.

36.3-2

הוכח כי $L \le_p \bar{L}$ אם ורק אם $\bar{L} \le_p L$.

36.3-3

הראה שהשמה מספקת יכולה לשמש בתור אישור בהוכחה חלופית של למה 36.5. איזה אישור יוצר הוכחה קלה יותר?

36.3-4

הוכחת למה 36.6 מניחה ששטח העבודה עבור האלגוריתם A תופס אזור רצוף בגודל פולינומיאלי. היכן בהוכחה מנצלים הנחה זו? הראה שאין בהנחה זו משום הגבלת הכלליות.

36.3-5

שפה L היא **שלמה** (complete) עבור מחלקת שפות C ביחס לרדוקציות זמן־פולינומיאלי אם $L \in C$ ו־$L' \le_p L$ עבור כל $L' \in C$. הראה ש־ϕ ו־$\{0, 1\}^*$ הן השפות היחידות ב־P שאינן שלמות עבור P ביחס לרדוקציות זמן־פולינומיאלי.

36.3-6

הראה ש־L שלמה עבור NP אם ורק אם $\bar{L}$ שלמה עבור co-NP.

36.3-7

אלגוריתם הרדוקציה F בהוכחת למה 36.6 בונה את המעגל $C = f(x)$ על סמך ידע של A, x ו־k. פרופסור סארטר מעיר ש־F מקבל כקלט את המחרוזת x, אבל באשר ל־A ו־k, רק קיומם ידוע ל־F (שכן השפה L שייכת ל־NP), אך לא ערכיהם בפועל. הפרופסור מסיק מכך ש־F אין שום אפשרות לבנות את המעגל C, ולכן השפה CIRCUIT-SAT אינה בהכרח NP־קשה. הסבר את הכשל בטיעונו של הפרופסור.

36.4 הוכחות שלמות ב־NP

השלמות ב־NP של בעיית הספיקות של מעגלים מסתמכת על הוכחה ישירה כי CIRCUIT-SAT $\le_p L$ עבור כל שפה $L \in$ NP. בסעיף זה נראה כיצד מוכיחים כי שפות הן שלמות ב־NP ללא רדוקציה ישירה של כל שפה ב־NP לשפה הנתונה. נדגים את השיטה הזאת באמצעות הוכחת השלמות ב־NP של בעיות שונות של ספיקות נוסחאות. בסעיף 36.5 מובאות דוגמאות רבות להוכחות בשיטה זו.

הלמה שלהלן היא הבסיס לשיטה שלנו להוכחת השלמות ב־NP של שפה נתונה.

למה 36.8

אם L היא שפה כך ש־$L' \le_p L$ עבור איזושהי $L' \in$ NPC, אזי L היא NP־קשה. יתר על כן, אם $L \in$ NP, אזי $L \in$ NPC.

הוכחה מכיוון ש־L' שלמה ב־NP, אזי כל $L'' \in$ NP מקיימת $L'' \le_p L'$. על־פי הנתון, $L' \le_p L$, ולכן מהטרזיטיביות (תרגיל 36.3-1) מקבלים כי $L'' \le_p L$. בכך הראינו כי L היא NP־קשה. אם $L \in$ NP, אזי מתקיים גם $L \in$ NPC. ■

במילים אחרות, על־ידי רדוקציה של שפה NP־שלמה ידועה L' ל־L, מבצעים במובלע רדוקציה של כל שפה ב־NP ל־L. למה 36.8 מספקת לנו אפוא שיטה להוכחה כי שפה L היא NP־שלמה:

1. הוכח כי $L \in$ NP.
2. בחר שפה NP־שלמה ידועה L'.
3. תאר אלגוריתם המחשב פונקציה f הממפה כל מופע של L' למופע של L.

4. הוכח שהפונקציה f מקיימת $x \in L'$ אם ורק אם $f(x) \in L$, עבור כל $x \in \{0,1\}^*$.

5. הוכח שהאלגוריתם המחשב את f רץ בזמן פולינומיאלי.

שיטה זו של רדוקציה משפה NP־שלמה ידועה אחת פשוטה בהרבה מאשר התהליך המסובך של מציאת רדוקציות מכל שפה ב־NP. ההוכחה כי CIRCUIT-SAT $\in$ NPC סיפקה לנו "דריסת רגלי". הידיעה כי בעיית הספיקות של מעגלים היא NP־שלמה מאפשרת לנו להוכיח ביתר קלות שבעיות אחרות הן NP־שלמות. יתירה מזו, ככל שתתארך רשימת הבעיות שידוע כי הן NP־שלמות, כך ייעשה השימוש בשיטה זו קל יותר ויותר.

ספיקות נוסחאות

נדגים את שיטת הרדוקציה באמצעות הוכחת השלמות ב־NP של בעיית ההכרעה אם נוסחה בוליאנית (ולא מעגל בוליאני) היא טפיקה. לבעיה זו מקום היסטורי מכובד בשל היותה הבעיה הראשונה שעבורה הוכיחו כי היא NP־שלמה.

את **בעיית הספיקות (של נוסחאות)** (formula) satisfiability problem)) ננסח במונחי השפה SAT כדלקמן: מופע של SAT הוא נוסחה בוליאנית ϕ המורכבת מהסימנים הבאים:

1. משתנים בוליאניים: $x_1, x_2, \ldots$.
2. קשָׁרים בוליאניים: כל פונקציה בוליאנית בעלת קלט אחד או שניים ופלט אחד, כגון $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT), $\rightarrow$ (גרירה), $\leftrightarrow$ (אם ורק אם), וכן
3. סוגריים.

כמו במעגלים צירופיים בוליאניים, **השמת ערכי אמת** (truth assignment) עבור נוסחה בוליאנית ϕ היא קבוצת ערכים עבור משתני ϕ, ו**השמה מספקת** (satisfying assignment) היא השמת ערכי אמת הגורמת לכך שהערך שמייצגת הנוסחה יהיה 1. נוסחה שעבורה קיימת השמה מספקת היא נוסחה **ספיקה** (satisfiable). בעיית הספיקות שואלת אם נוסחה בוליאנית נתונה היא ספיקה; במונחי שפות פורמליות:

$$\text{SAT} = \{\langle\phi\rangle : \phi \text{ היא נוסחה בוליאנית ספיקה}\}$$

לדוגמה, עבור הנוסחה:

$$\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$$

ההשמה $\langle x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1\rangle$ היא השמה מספקת, שכן:

$$\begin{aligned} \phi &= ((0 \rightarrow 0) \vee \neg((\neg 0 \leftrightarrow 1) \vee 1)) \wedge \neg 0 \\ &= (1 \vee \neg(1 \vee 1)) \wedge 1 \\ &= (1 \vee 0) \wedge 1 \\ &= 1 \end{aligned} \tag{36.2}$$

ולכן נוסחה ϕ זו שייכת ל־SAT.

האלגוריתם הנאיבי הקובע אם נוסחה בוליאנית בלשהי היא ספיקה אינו רץ בזמן פולינומיאלי. עבור נוסחה ϕ המכילה n משתנים קיימות 2^n השמות אפשריות. אם האורך של $\langle\phi\rangle$ פולינומיאלי

ב־n, אזי בדיקת כל ההשמות דורשת זמן על־פולינומיאלי. כפי שמוכיח המשפט שלהלן, לא קיים ככל הנראה אלגוריתם זמן־פולינומיאלי עבור בעיה זו.

משפט 36.9
ספיקות נוסחאות בוליאניות היא NP־שלמה.

הוכחה תחילה נטען כי SAT $\in$ NP. לאחר מכן נראה כי CIRCUIT-SAT $\leq_p$ SAT, על פי למה 36.8, ובכך יוכח המשפט.

כדי להוכיח כי SAT שייכת ל־NP, נראה כי אישור המורכב מהשמה מספקת עבור נוסחת קלט ϕ ניתן לאימות בזמן פולינומיאלי. אלגוריתם האימות פשוט מחליף כל משתנה בנוסחה בערך המתאים לו, ואז מחשב את ערכו של הביטוי, כמו שעשינו עבור משוואה (36.2) לעיל. זוהי משימה שקל לבצע אותה בזמן פולינומיאלי. אם ערכו של הביטוי הוא 1, הנוסחה ספיקה. אם כן, התנאי הראשון של למה 36.8 עבור שלמות ב־NP מתקיים.

כדי להוכיח ש־SAT היא NP־קשה, נראה כי CIRCUIT-SAT $\leq_p$ SAT. במילים אחרות, כל מופע של ספיקות מעגלים ניתן לרדוקציה בזמן פולינומיאלי למופע של ספיקות נוסחאות. באמצעות אינדוקציה ניתן לבטא כל מעגל צירופי בוליאני כנוסחה בוליאנית. פשוט מתבוננים בשער המפיק את פלט המעגל ובאופן אינדוקטיבי מבטאים כל אחד מהקלטים לשער כנוסחה. הנוסחה עבור המעגל כולו מתקבלת על־ידי כתיבת ביטוי שבו הפונקציה שמממש השער מופעלת על הנוסחאות של הקלטים שלו.

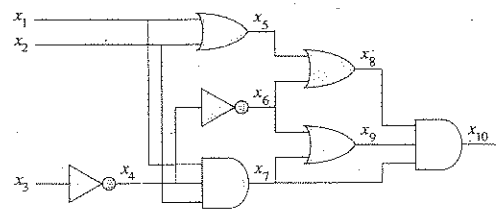
לרוע המזל, שיטה ישירה זו אינה מהווה רדוקציית זמן־פולינומיאלי. הנוסחה הנוצרת עלולה לגדול באופן אקספוננציאלי בשל תת־נוסחאות משותפות (ראה תרגיל 36.4-1). לכן, אלגוריתם הרדוקציה צריך להיות מתוחכם מעט יותר.

איור 36.8 מדגים את הרעיון הבסיסי של הרדוקציה מ־CIRCUIT-SAT ל־SAT עבור המעגל מאיור 36.6(i). עבור כל תיל x_i במעגל C, מופיע בנוסחה ϕ משתנה x_i את פעולתו של כל שער ניתן עתה לבטא כנוסחה שבה מופיעים המשתנים המייצגים את התילים המגיעים אל השער. לדוגמה, פעולת שער הפלט AND היא $x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)$.

הנוסחה ϕ שיוצר אלגוריתם הרדוקציה היא ה־AND של משתנה הפלט של המעגל ושל הקוניונקציה של הפסוקיות המתארות את פעולתו של כל שער. עבור המעגל שבאיור, הנוסחה היא:

$$\begin{aligned} \phi = {} & x_{10} \wedge (x_4 \leftrightarrow \neg x_3) \\ & \wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\ & \wedge (x_6 \leftrightarrow \neg x_4) \\ & \wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\ & \wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \\ & \wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \\ & \wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)) \end{aligned}$$

בהינתן מעגל C, אין שום קושי ליצור נוסחה ϕ כזאת בזמן פולינומיאלי.



איור 36.8 רדוקציה של ספיקות מעגלים לספיקות נוסחאות. הנוסחה שיוצר אלגוריתם הרדוקציה מכילה משתנה עבור כל תיל במעגל.

מדוע המעגל C ספיק בדיוק כאשר הנוסחה ϕ ספיקה? אם קיימת עבור C השמה מספקת, הרי שלכל תיל במעגל יש ערך מוגדר היטב, ופלט המעגל הוא 1. לכן, השמת ערכי התילים למשתנים ב־ϕ גורמת לכך שכל פסוקית ב־ϕ מקבלת את הערך 1, ולכן גם הקוניונקציה של כולן מקבלת את הערך 1. בכיוון ההפוך, אם קיימת השמה הגורמת לכך שערכה של ϕ יהיה 1, אזי על פי טיעון אנלוגי, המעגל C ספיק. הראינו אפוא כי $\text{CIRCUIT-SAT} \leq_p \text{SAT}$, ובכך הושלמה הוכחת המשפט. ■

ספיקות נוסחאות 3-CNF

באמצעות רדוקציה מבעיית ספיקות הנוסחאות ניתן להוכיח את השלמות ב־NP של בעיות רבות. אבל אלגוריתם הרדוקציה צריך לטפל בכל נוסחת קלט אפשרית, כך שהוא עלול להידרש לבחון מספר עצום של מקרים. לכן, לעתים קרובות רצוי לבצע רדוקציה משפה מצומצמת של נוסחאות בוליאניות, כך שיהיה צורך לבחון מספר מועט יותר של מקרים. כמובן, יש להיזהר מצמצום השפה עד כדי כך שתיעשה פתירה בזמן פולינומיאלי. אחת השפות הנותות היא 3-CNF-SAT, השפה המייצגת את בעיית ספיקותן של נוסחאות 3-CNF.

נגדיר את הספיקות של נוסחאות 3-CNF באמצעות המונחים הבאים: **ליטרל** (literal) בנוסחה בוליאנית הוא מופע של משתנה או של שלילתו. נוסחה בוליאנית כתובה **בצורה קוניונקטיבית נורמלית** (conjunctive normal form), ובקיצור **CNF**, אם היא מבוטאת כ־AND של פסוקיות שכל אחת מהן היא OR של ליטרל אחד או יותר. נוסחה בוליאנית נתונה בצורה **3-CNF** (3-conjunctive normal form) אם כל פסוקית מכילה בדיוק שלושה ליטרלים שונים.

לדוגמה, הנוסחה הבוליאנית:

$$(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

היא נוסחת 3-CNF. הראשונה בשלוש הפסוקיות שלה היא $(x_1 \vee \neg x_1 \vee \neg x_2)$, המכילה את שלושת הליטרלים x_1, $\neg x_1$ ו־$\neg x_2$.

בעיית ספיקותן של נוסחאות 3-CNF שואלת אם נוסחת 3-CNF בוליאנית נתונה ϕ היא ספיקה. המשפט שלהלן מראה כי אין זה סביר שקיים אלגוריתם זמן־פולינומיאלי שיכול לקבוע אם נוסחאות בוליאניות הן ספיקות, גם אם הנוסחאות כתובות בצורה נורמלית פשוטה זו.

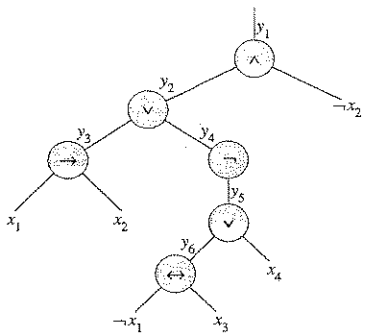
משפט 36.10
בעיית הספיקות של נוסחאות בוליאניות 3-CNF היא NP־שלמה.

הוכחה הטיעון שהשתמשנו בו בהוכחת משפט 36.9 כדי להראות כי $\text{SAT} \in \text{NP}$ מתאים באותה מידה גם כאן, וניתן להראות באמצעותו כי $\text{3-CNF-SAT} \in \text{NP}$. לכן, עלינו רק להראות כי 3-CNF-SAT היא NP־קשה. נוכיח זאת על־ידי כך שנראה כי $\text{SAT} \leq_p \text{3-CNF-SAT}$, ומכך תנבע ההוכחה על פי למה 36.8.

את אלגוריתם הרדוקציה ניתן לחלק לשלושה צעדים בסיסיים. כל צעד משנה בהדרגה את נוסחת הקלט ϕ ומקרב אותה לצורה המבוקשת 3-CNF.

הצעד הראשון דומה לזה שנקטנו בו במשפט 36.9, כשהוכחנו כי $\text{CIRCUIT-SAT} \leq_p \text{SAT}$. תחילה נבנה עבור נוסחת הקלט ϕ עץ "פיסוק" בינרי שבו הליטרלים מופיעים בעלים והקשרים בצמתים הפנימיים. באיור 36.9 מתואר עץ פיסוק כזה עבור הנוסחה:

$$\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2 \qquad (36.3)$$



איור 36.9 העץ המתאים לנוסחה: $\phi = ((x_1 \rightarrow x_2) \vee \neg(\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$

אם נוסחת הקלט מכילה למשל פסוקית שהיא ה־OR של כמה ליטרלים, ניתן להשתמש באסוציאטיביות כדי להוסיף לביטוי הצבת סוגריים מלאה, כך שלכל צומת פנימי בעץ המתקבל יהיו בן אחד או שניים. עתה ניתן לראות את עץ הפיסוק הבינרי כמעגל לחישוב הפונקציה.

נחקה את הרדוקציה בהוכחת משפט 36.9, ונגדיר משתנה y_i עבור הפלט של כל צומת פנימי. לאחר מכן נכתוב מחדש את הנוסחה המקורית ϕ כ־AND של משתנה השורש ושל קוניונקציה של פסוקיות המתארות את הפעולה של כל צומת. הביטוי המתקבל עבור הנוסחה (36.3) הוא:

$$\begin{aligned}\phi' = {} & y_1 \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\ & \wedge (y_2 \leftrightarrow (y_3 \vee y_4)) \\ & \wedge (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \\ & \wedge (y_4 \leftrightarrow \neg y_5) \\ & \wedge (y_5 \leftrightarrow (y_6 \vee x_4)) \\ & \wedge (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))\end{aligned}$$

נשים לב כי הנוסחה ϕ' המתקבלת באופן זה היא קוניונקציה של פסוקיות ϕ'_i שכל אחת מהן מכילה לכל היותר 3 ליטרלים. הדרישה הנוספת היחידה היא שכל פסוקית תהיה OR של ליטרלים.

בצעד השני של הרדוקציה ממירים כל פסוקית ϕ'_i לצורה קוניונקטיבית נורמלית. מחשבים את ערכה של ϕ'_i עבור כל השמה אפשרית של ערכי אמת למשתנים בפסוקית, ובונים את טבלת האמת שלה. כל שורה בטבלת האמת מורכבת מהשמה אפשרית של ערכי אמת למשתני הפסוקית ומערכה של הפסוקית תחת השמה זו. באמצעות השורות בטבלה שבהן ערכה של הפסוקית הוא 0, בונים נוסחה שקולה ל־$\neg\phi'_i$ הכתובה **בצורה דיסיונקטיבית נורמלית** (disjunctive normal form או DNF), דהיינו, דיסיונקציה של קוניונקציות. לאחר מכן ממירים נוסחה זו לנוסחה ϕ''_i הכתובה בצורה קוניונקטיבית נורמלית (CNF) תוך שימוש בחוקי דה־מורגן (5.2) כדי לשנות את כל הליטרלים למשלימיהם, וכדי להפוך פעולות OR לפעולות AND, ופעולות AND לפעולות OR.

y_1	y_2	x_2	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	1

איור 36.10 טבלת האמת של הפסוקית $(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$.

בדוגמה שלנו, אנו ממירים את הפסוקית $\phi'_1 = (y_1 \leftrightarrow (y_2 \wedge \neg x_2))$ לצורת CNF כדלקמן. טבלת האמת של ϕ'_1 נתונה באיור 36.10. נוסחת DNF השקולה ל־$\neg\phi'_1$ היא:

$$(y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2)$$

על ידי הפעלת חוקי דה־מורגן מתקבלת נוסחת CNF הבאה:

$$\begin{aligned}\phi''_1 = {} & (\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \\ & \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2)\end{aligned}$$

השקולה לפסוקית המקורית ϕ'_1.

עתה, כל פסוקית ϕ'_i בנוסחה ϕ' הומרה לנוסחת CNF שקולה ϕ''_i, ולכן ϕ' שקולה לנוסחת CNF ϕ'' המורכבת מקוניונקציה של הפסוקיות ϕ''_i. יתירה מזו, כל פסוקית של ϕ'' מכילה לכל היותר 3 ליטרלים.

בצעד השלישי והאחרון ברדוקציה ממשיכים ומשנים את הנוסחה כך שכל פסוקית תכיל *בדיוק* 3 ליטרלים. נוסחת 3-CNF הסופית ϕ''' בנויה מהפסוקיות של נוסחת ה־CNF ϕ''. בצעד זה גם משתמשים בשני משתני עזר שנקרא להם p ו־q. עבור כל פסוקית C_i ב־ϕ'' נכלול ב־ϕ''' את הפסוקיות הבאות:

- אם C_i מכילה בדיוק 3 ליטרלים שונים, פשוט כוללים אותה כפסוקית ב־ϕ'''.
- אם C_i מכילה 2 ליטרלים שונים, כלומר, אם $C_i = (l_1 \vee l_2)$, כאשר l_1 ו־l_2 הם ליטרלים, אזי כוללים ב־ϕ''' את הפסוקיות $(l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$. הליטרלים p ו־$\neg p$ נוספו רק כדי למלא את הדרישה התחבירית שכל פסוקית תכיל בדיוק 3 ליטרלים שונים. $(l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$ שקול ל־$(l_1 \vee l_2)$, בין אם $p = 0$ ובין אם $p = 1$.
- אם C_i מכילה ליטרל אחד בלבד l, אזי כוללים ב־ϕ''' את הפסוקיות $(l \vee \neg p \vee q) \wedge (l \vee \neg p \vee \neg q) \wedge (l \vee p \vee q) \wedge (l \vee p \vee \neg q)$. אפשר לראות מיד שעבור כל השמה אפשרית של ערכי אמת ב־p וב־q, ערך הקוניונקציה של ארבע הפסוקיות הללו הוא l.

על־ידי בדיקת כל אחד משלושת הצעדים הללו ניתן להיווכח כי ϕ''' ספיקה אם ורק אם ϕ ספיקה. כמו ברדוקציה מ־CIRCUIT-SAT ל־SAT, בניית ϕ' מ־ϕ בצעד הראשון משמרת את הספיקות. הצעד השני יוצר נוסחת CNF ϕ'' השקולה אלגברית ל־ϕ'. הצעד השלישי יוצר נוסחת 3-CNF ϕ''' השקולה למעשה ל־ϕ'', שכן כל השמה עבור המשתנים p ו־q יוצרת נוסחה השקולה אלגברית ל־ϕ''.

עלינו גם להראות שהרדוקציה ניתנת לחישוב בזמן פולינומיאלי. בניית ϕ' מ־ϕ מוסיפה לכל היותר משתנה אחד ופסוקית אחת עבור כל קשר ב־ϕ. בניית ϕ'' מ־ϕ' יכולה להוסיף ל־ϕ'' לכל היותר 8 פסוקיות עבור כל פסוקית של ϕ', שכן כל פסוקית של ϕ' מכילה לכל היותר 3 משתנים, וטבלת האמת עבור כל פסוקית מכילה לכל היותר $2^3 = 8$ שורות. בניית ϕ''' מ־ϕ'' מוסיפה ל־ϕ''' לכל היותר 4 פסוקיות עבור כל פסוקית של ϕ''. לכן, גודלה של הנוסחה המתקבלת ϕ''' פולינומיאלי באורכה של הנוסחה המקורית. את כל אחת מן הבניות ניתן לבצע בקלות בזמן פולינומיאלי. ■

תרגילים

36.4-1

בחן את הרדוקציה הישירה (שרצה בזמן על־פולינומיאלי) המופיעה בהוכחת משפט 36.9. תאר מעגל בגודל n אשר, כשממירים אותו לנוסחה בשיטה זו, מניב נוסחה שגודלה אקספוננציאלי ב־n.

36.4-2

הצג את נוסחת 3-CNF המתקבלת מנוסחה (36.3) באמצעות השיטה המתוארת במשפט 36.10.

36.4-3

פרופסור ג'אגר מציע להראות כי $\text{SAT} \leq_P \text{3-CNF-SAT}$ רק באמצעות שיטת טבלת האמת המופיעה בהוכחת משפט 36.10, בלי הצעדים האחרים. כלומר, הפרופסור מציע לקחת את הנוסחה הבוליאנית ϕ, ליצור את טבלת האמת של משתניה, לגזור מטבלת האמת נוסחת 3-DNF השקולה ל־ϕ, ואז לשלול אותה וליצור בעזרת חוקי דה־מורגן נוסחת 3-CNF השקולה ל־ϕ. הראה שאסטרטגיה זו אינה מניבה רדוקציית זמן־פולינומיאלי.

36.4-4

הראה שבעיית ההכרעה אם נוסחה בוליאנית היא טאוטולוגיה היא בעיה שלמה עבור co-NP. (**רמז:** ראה תרגיל 36.3-6.)

36.4-5

הראה שהבעיה של הכרעת ספיקותן של נוסחאות בוליאניות הכתובות בצורה דיסיונקטיבית נורמלית ניתנת לפתרון בזמן פולינומיאלי.

36.4-6

נניח שמישהו נותן לך אלגוריתם זמן־פולינומיאלי להכרעת ספיקות נוסחאות. תאר כיצד ניתן להשתמש באלגוריתם זה למציאת השמות מספקות בזמן פולינומיאלי.

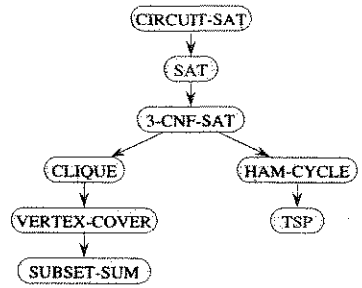
36.4-7

תהי 2-CNF-SAT קבוצת כל הנוסחאות הבוליאניות הספיקות הכתובות בצורה קוניונקטיבית נורמלית, כאשר כל פסוקית מכילה בדיוק 2 ליטרלים. הראה כי $\text{2-CNF-SAT} \in \text{P}$. בנה אלגוריתם יעיל ככל האפשר. (**רמז:** שים לב כי $x \vee y$ שקולה ל־$\neg x \rightarrow y$. בצע רדוקציה של 2-CNF-SAT לבעיה על גרפים מכוונים שניתן לפותרה ביעילות.)

36.5 בעיות NP־שלמות

בעיות NP־שלמות מתעוררות בתחומים רבים ושונים: לוגיקה בוליאנית, גרפים, אריתמטיקה, תכנון רשתות, קבוצות וחלוקות, אחסון ואחזור, תזמון ולוחות זמנים, תכנון מתמטי, אלגברה ותורת המספרים, משחקים וחידות, אוטומטים ותורת השפות, אופטימיזציה של תכניות, ועוד ועוד. בסעיף זה נשתמש בשיטת הרדוקציה כדי להוכיח את שלמותן ב־NP של מגוון בעיות בתורת הגרפים ובחלוקת קבוצות.

התרשים באיור 36.11 מציג בקווים כלליים את המבנה של הוכחות השלמות ב־NP המובאות בסעיף זה ובסעיף 36.4. השלמות ב־NP של כל שפה המופיעה בתרשים מוכחת באמצעות רדוקציה מן השפה המצביעה עליה. בשורש נמצאת CIRCUIT-SAT, שאת שלמותה ב־NP הוכחנו במשפט 36.7.



איור 36.11 המבנה של הוכחות השלמות ב־NP המובאות בסעיפים 36.4 ו־36.5. כל ההוכחות מושגות בסופו של דבר בשיטת הרדוקציה מהשלמות ב־NP של CIRCUIT-SAT.

36.5.1 בעיית הקליקה

קליקה (clique) בגרף בלתי־מכוון $G = (V, E)$ היא תת־קבוצה $V' \subseteq V$ של קדקודים, כך שכל זוג קדקודים מחובר על־ידי קשת ב־E. במילים אחרות, קליקה היא תת־גרף שלם של G. **הגודל** (size) של קליקה הוא מספר הקדקודים שהיא מכילה. **בעיית הקליקה** (the clique problem) היא בעיית האופטימיזציה של מציאת קליקה מקסימלית בגודלה בגרף. כבעיית הכרעה, אנו פשוט שואלים אם קיימת בגרף קליקה בגודל נתון k. ההגדרה הפורמלית היא:

$$\text{CLIQUE} = \{\langle G, k\rangle : k \text{ הוא גרף המכיל קליקה בגודל } G\}$$

אלגוריתם נאיבי הקובע אם גרף $G = (V, E)$ בעל $|V|$ קדקודים מכיל קליקה בגודל k יערוך רשימה של כל תת־קבוצות של V שגודלן k, ויבדוק כל אחת מהן כדי לקבוע אם היא מהווה קליקה. זמן הריצה של אלגוריתם כזה הוא $\Omega\left(k^2\binom{|V|}{k}\right)$, שהוא פולינומיאלי אם k קבוע. אבל ככלל, ייתכן שערכו של k נמצא ביחס ישר ל־$|V|$, ובמקרה זה האלגוריתם ירוץ בזמן על־פולינומיאלי. כפי שניתן היה לחשוד, אין זה סביר שקיים אלגוריתם יעיל לפתרון בעיית הקליקה.

משפט 36.11
בעיית הקליקה היא NP־שלמה.

הוכחה כדי להראות כי $\text{CLIQUE} \in \text{NP}$, עבור גרף נתון $G = (V, E)$, אנו משתמשים בקבוצת הקדקודים $V' \subseteq V$ בקליקה כאישור עבור G. את הבדיקה אם V' היא אכן קליקה ניתן לבצע בזמן פולינומיאלי על־ידי כך שבודקים, עבור כל זוג $u, v \in V'$, אם הקשת (u, v) שייכת ל־E.

עתה נראה שבעיית הקליקה היא NP־קשה על־ידי כך שנוכיח כי $3\text{-CNF-SAT} \leq_p \text{CLIQUE}$. העובדה שניתן להוכיח תוצאה זו מפתיעה במידה מסוימת, שכן במבט ראשון נראה שאין קשר בין נוסחאות לוגיות לבין גרפים.

אלגוריתם הרדוקציה מתחיל עם מופע של 3-CNF-SAT. תהי $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ נוסחת 3-CNF בוליאנית המכילה k פסוקיות. עבור $r = 1, 2, \ldots, k$, כל פסוקית C_r מכילה בדיוק שלושה ליטרלים שונים, l_1^r, l_2^r ו־l_3^r. אנו נבנה גרף G כך ש־G מכיל קליקה בגודל k אם ורק אם ϕ ספיקה.

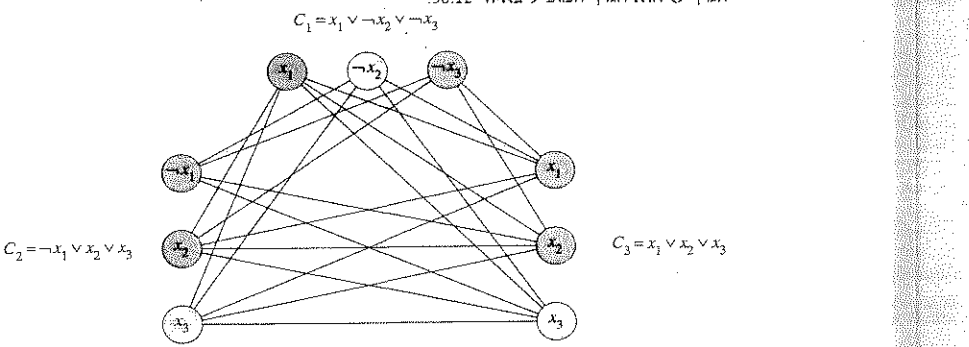
את הגרף $G = (V, E)$ נבנה כדלקמן: עבור כל פסוקית $C_r = (l_1^r \vee l_2^r \vee l_3^r)$ ב־ϕ, נוסיף ל־V שלשת קדקודים v_1^r, v_2^r ו־v_3^r. נחבר בקשת שני קדקודים v_i^r ו־v_j^s אם מתקיימים שני התנאים הבאים:

- v_i^r ו־v_j^s שייכים לשלשות שונות, דהיינו $r \neq s$, וגם
- הליטרלים המתאימים להם **קונסיסטנטיים**, כלומר, l_i^r אינו שלילתו של l_j^s.

בהינתן ϕ, קל לחשב גרף כזה בזמן פולינומיאלי. להלן דוגמה לבנייה כזאת. בהינתן הנוסחה:

$$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

הגרף G הוא הגרף המופיע באיור 36.12.



איור 36.12 הגרף G המתקבל עבור נוסחת 3-CNF $\phi = C_1 \wedge C_2 \wedge C_3$, שבה $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$, $C_2 = (\neg x_1 \vee x_2 \vee x_3)$ ו־$C_3 = (x_1 \vee x_2 \vee x_3)$, בעת הרדוקציה של 3-CNF-SAT ל־CLIQUE. השמה מספקת עבור נוסחה זו היא $(x_1 = 0, x_2 = 0, x_3 = 1)$, שכן x_2 מספק את C_1 ו־$\neg x_3$ מספק את C_2 ו־C_3. השמה זו מתאימה לקליקה שיוצרים הקדקודים הבהירים יותר באיור.

עלינו להראות כי טרנספורמציה זו של ϕ ל־G היא רדוקציה. ראשית, נניח שקיימת השמה המספקת את ϕ. אזי, כל פסוקית C_r מכילה לפחות ליטרל אחד l_i^r שמקבל בהשמה זו את הערך 1, וכל ליטרל כזה מתאים לקדקוד v_i^r. בחירת ליטרל "אמת" אחד כזה מכל פסוקית מניבה

שאין במצאות ב-G. באיור 36.13 מוצגים גרף והמשלים שלו ומודגמת הדוקציה מ-CLIQUE ל-VERTEX-COVER.

נניח כי בעיית ההכרעה עבור קבוצות קודקודים היא NP-קשה ע"י רדוקציה מ-CLIQUE. כלומר, $\text{CLIQUE} \leq_p \text{VERTEX-COVER}$. דוקציה זו מבוססת על מושג "המשלים" של גרף. במילים אחרות, בהינתן גרף $G = (V, E)$ נגדיר כעת את $\overline{G} = (V, \overline{E})$ כגרף המכיל את **המשלים** (complement) של G. אם מגדירים את $\overline{E}$ להיות $\{(u, v) : (u, v) \notin E\}$ אזי $\overline{G}$ הוא הגרף המכיל בדיוק את הקשתות שאין ב-G. במילים אחרות, עבור כל זוג קודקודים u ו-v ב-V, הקשת (u, v) נמצאת ב-$\overline{G}$ אם ורק אם (u, v) אינה נמצאת ב-G.

הדוקציה מקבלת כקלט גרף G ומספר שלם k עבור בעיית הקליקה. היא מחשבת את $\overline{G}$ ומחזירה את הקלט $\langle \overline{G}, |V| - k \rangle$ לבעיית כיסוי הקודקודים. כדי להשלים את הוכחת הנכונות של הדוקציה, נראה כי לגרף G יש קליקה בגודל k אם ורק אם ל-$\overline{G}$ יש כיסוי קודקודים בגודל $|V| - k$.

נניח כי $V' \subseteq V$ היא קליקה בגודל k ב-G. נטען כי $V - V'$ היא כיסוי קודקודים ב-$\overline{G}$. יהי (u, v) קשת כלשהי ב-$\overline{E}$. אזי $(u, v) \notin E$, ומכאן ש-$u \notin V'$ או $v \notin V'$ (או שניהם). אחרת, שתי הקודקודים u ו-v שייכים ל-V', ולכן $(u, v) \in E$ מכיוון ש-V' היא קליקה. מכאן נובע כי כל קשת ב-$\overline{E}$ חלה על קודקוד אחד לפחות ב-$V - V'$, ולכן $V - V'$ היא כיסוי קודקודים ב-$\overline{G}$. להיפך, נניח כי $V' \subseteq V$ היא כיסוי קודקודים ב-$\overline{G}$. עבור כל $u, v \in V$, אם $(u, v) \in \overline{E}$ אזי $u \in V - V'$ או $v \in V - V'$ (או שניהם). שקילות זו שקולה לכך שאם $u \notin V'$ ו-$v \notin V'$ אזי $(u, v) \in E$. מכאן נובע כי V' היא קליקה בגודל $|V| - |V - V'| = k$.

■

36.5.2 בעיית הכיסוי על-ידי קדקודים

בעיית הכיסוי על-ידי קדקודים (the vertex-cover problem) היא למצוא עבור גרף נתון כיסוי קטן ביותר על-ידי קדקודים שנתון. ניתן להגדיר בעיה זו כבעיית אופטימיזציה זו באופן הטבעי: מהו הכיסוי על-ידי קדקודים בגודל מינימלי עבור גרף נתון? כדי להפוך בעיה זו לבעיית הכרעה, נצטרך להגדיר גודל מקסימלי עבור כיסוי. נגדיר את הבעיה כך:

$$\text{VERTEX-COVER} = \{\langle G, k \rangle : \text{לגרף } G \text{ יש כיסוי על-ידי קדקודים שגודלו } k\}.$$

המשפט הבא מראה שבעיה זו היא NP-שלמה.

משפט 36.12

בעיית הכיסוי על-ידי קדקודים היא NP-שלמה.

הוכחה תחילה נראה כי $\text{VERTEX-COVER} \in \text{NP}$. נניח שתנתנים גרף $G = (V, E)$ ומספר שלם k. האישור שנבחר הוא הכיסוי $V' \subseteq V$ עצמו. אלגוריתם האימות מאשר כי $|V'| = k$ ולאחר מכן בודק, עבור כל קשת $(u, v) \in E$, אם $u \in V'$ או $v \in V'$ או שניהם. אם כל קשת נבדקת מוצאת לפחות קודקוד אחד בכיסוי, אזי הקלט נתקבל.

יש לבחון מספר פולינומיאלי.

משפט 36.12

בעיית הכיסוי על-ידי קדקודים היא NP-שלמה.

הוכחה תחילה נראה כי $\text{VERTEX-COVER} \in \text{NP}$. נניח שתנתנים גרף $G = (V, E)$ ומספר שלם k. האישור שנבחר הוא הכיסוי $V' \subseteq V$ עצמו. אלגוריתם האימות מאשר כי $|V'| = k$ ולאחר מכן בודק, עבור כל קשת $(u, v) \in E$, אם $u \in V'$ או $v \in V'$ או שניהם. אם כל קשת נבדקת מוצאת לפחות קודקוד אחד בכיסוי, אזי הקלט נתקבל.

יש לבחון מספר פולינומיאלי.

משפט 36.12

בעיית הכיסוי על-ידי קדקודים היא NP-שלמה.

הוכחה תחילה נראה כי $\text{VERTEX-COVER} \in \text{NP}$. נניח שתנתנים גרף $G = (V, E)$ ומספר שלם k. האישור שנבחר הוא הכיסוי $V' \subseteq V$ עצמו. אלגוריתם האימות מאשר כי $|V'| = k$ ולאחר מכן בודק, עבור כל קשת $(u, v) \in E$, אם $u \in V'$ או $v \in V'$ או שניהם. אם כל קשת נבדקת מוצאת לפחות קודקוד אחד בכיסוי, אזי הקלט נתקבל.

יש לבחון מספר פולינומיאלי.

משפט 36.12

בעיית הכיסוי על-ידי קדקודים היא NP-שלמה.

הוכחה תחילה נראה כי $\text{VERTEX-COVER} \in \text{NP}$. נניח שתנתנים גרף $G = (V, E)$ ומספר שלם k. האישור שנבחר הוא הכיסוי $V' \subseteq V$ עצמו. אלגוריתם האימות מאשר כי $|V'| = k$ ולאחר מכן בודק, עבור כל קשת $(u, v) \in E$, אם $u \in V'$ או $v \in V'$ או שניהם. אם כל קשת נבדקת מוצאת לפחות קודקוד אחד בכיסוי, אזי הקלט נתקבל.

יש לבחון מספר פולינומיאלי.

איור 36.13 הדוקציה של CLIQUE ל-VERTEX-COVER. (i) גרף בלתי-מכוון $G = (V, E)$ המכיל קליקה $V' = \{w, x, y, z\}$, כאשר $k = 4$. (ii) הגרף $\overline{G}$ שנוצר על-ידי הדוקציה. הכיסוי $V - V' = \{u, v, s\}$ הוא כיסוי קודקודים ב-$\overline{G}$.

מכיוון שהדוקציה מבוצעת בזמן פולינומיאלי, הוכחנו כי CLIQUE $\leq_p$ VERTEX-COVER. מכאן נובע כי VERTEX-COVER היא NP-שלמה.

36.5 בעיות NP-שלמות

כפי שציינו לעיל, הידיעה כי בעיה מסוימת היא NP-שלמה מספקת מידע חשוב לגבי אפשרות מציאת אלגוריתם יעיל לפתרון הבעיה. אם בעיה היא NP-קשה, אזי אין סיכוי רב שקיים אלגוריתם יעיל לפתרונה. אם בעיה היא NP-שלמה, אזי ניתן להשתמש בדוקציה כדי לפתור כל בעיה אחרת ב-NP בזמן פולינומיאלי. אלגוריתמים פולינומיאליים הם שבירים במובן זה שהם נוטים להיות בעלי זמני ריצה פולינומיאליים נמוכים, כגון $\Theta(n)$, $\Theta(n^2)$ או $\Theta(n^3)$. לעתים קרובות, שינוי קטן בהגדרת הבעיה גורם לכך שאין אלגוריתם יעיל הפותר אותה. לעתים קרובות, בעיות NP-שלמות מעניינות מכיוון שלמרות שרק מעט מאוד בעיות ידועות כפתרונות בזמן פולינומיאלי, אף אחד לא הוכיח שאי-אפשר לפתור אותן בזמן פולינומיאלי. פתרון בזמן פולינומיאלי לבעיה NP-שלמה אחת יספק פתרונות בזמן פולינומיאלי לכל בעיה NP-שלמה אחרת, דבר שנראה לא סביר מאוד. בנוסף, בעיות NP-שלמות מעניינות מכיוון שחלקן דומות, על פני השטח, לבעיות הידועות כפתרונות בזמן פולינומיאלי. שינוי קטן בבעיה יכול לגרום לכך שיהיה קשה למצוא אלגוריתם פולינומיאלי. עם זאת, ידע בנוגע לאלגוריתמים פולינומיאליים לעתים קרובות יכול לשפוך אור על בעיה NP-שלמה ולמצוא במקרים מסוימים אלגוריתמים יעילים לפתרון חלקים חשובים מהבעיה.

איור 36.13 הדוקציה של CLIQUE ל-VERTEX-COVER. (i) גרף בלתי-מכוון $G = (V, E)$ המכיל קליקה $V' = \{w, x, y, z\}$, כאשר $k = 4$. (ii) הגרף $\overline{G}$ שנוצר על-ידי הדוקציה. הכיסוי $V - V' = \{u, v, s\}$ הוא כיסוי קודקודים ב-$\overline{G}$.

מכיוון שהדוקציה מבוצעת בזמן פולינומיאלי, הוכחנו כי CLIQUE $\leq_p$ VERTEX-COVER. מכאן נובע כי VERTEX-COVER היא NP-שלמה.

36.5 בעיות NP-שלמות

כפי שציינו לעיל, הידיעה כי בעיה מסוימת היא NP-שלמה מספקת מידע חשוב לגבי אפשרות מציאת אלגוריתם יעיל לפתרון הבעיה. אם בעיה היא NP-קשה, אזי אין סיכוי רב שקיים אלגוריתם יעיל לפתרונה. אם בעיה היא NP-שלמה, אזי ניתן להשתמש בדוקציה כדי לפתור כל בעיה אחרת ב-NP בזמן פולינומיאלי. אלגוריתמים פולינומיאליים הם שבירים במובן זה שהם נוטים להיות בעלי זמני ריצה פולינומיאליים נמוכים, כגון $\Theta(n)$, $\Theta(n^2)$ או $\Theta(n^3)$. לעתים קרובות, שינוי קטן בהגדרת הבעיה גורם לכך שאין אלגוריתם יעיל הפותר אותה. לעתים קרובות, בעיות NP-שלמות מעניינות מכיוון שלמרות שרק מעט מאוד בעיות ידועות כפתרונות בזמן פולינומיאלי, אף אחד לא הוכיח שאי-אפשר לפתור אותן בזמן פולינומיאלי. פתרון בזמן פולינומיאלי לבעיה NP-שלמה אחת יספק פתרונות בזמן פולינומיאלי לכל בעיה NP-שלמה אחרת, דבר שנראה לא סביר מאוד. בנוסף, בעיות NP-שלמות מעניינות מכיוון שחלקן דומות, על פני השטח, לבעיות הידועות כפתרונות בזמן פולינומיאלי. שינוי קטן בבעיה יכול לגרום לכך שיהיה קשה למצוא אלגוריתם פולינומיאלי. עם זאת, ידע בנוגע לאלגוריתמים פולינומיאליים לעתים קרובות יכול לשפוך אור על בעיה NP-שלמה ולמצוא במקרים מסוימים אלגוריתמים יעילים לפתרון חלקים חשובים מהבעיה.

36.5.3 בעיית סכום התת־קבוצה

הבעיה ה־NP־שלמה הבאה שנבחן היא בעיה אריתמטית. **בבעיית סכום התת־קבוצה** (the subset-sum problem), נתונים קבוצה סופית $S \subset \mathbf{N}$ **וערך מטרה** (target) $t \in \mathbf{N}$. אנו שואלים אם קיימת תת־קבוצה $S' \subseteq S$ שאיבריה מסתכמים לערך t. לדוגמה, אם $S = \{1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344\}$ ו־$t = 3754$, אזי תת־קבוצה $S' = \{1, 16, 64, 256, 1040, 1093, 1284\}$ היא פתרון.

כרגיל, נגדיר את הבעיה כשפה:

$$\text{SUBSET-SUM} = \{\langle S, t\rangle : t = \Sigma_{s \in S'} s \text{ כך שמתקיים } S' \subseteq S \text{ קיימת תת־קבוצה}\}$$

כמו בכל בעיה אריתמטית, חשוב לזכור שהקידוד הסטנדרטי מניח כי מספרי הקלט השלמים מיוצגים בכתיב בינרי. בהניחנו זאת, נוכל להראות שאין זה סביר שקיים אלגוריתם מהיר הפותר את בעיית סכום התת־קבוצה.

משפט 36.13

בעיית סכום התת־קבוצה היא NP־שלמה.

הוכחה כדי להראות כי SUBSET-SUM שייכת ל־NP, נשתמש בתת־קבוצה S' כאישור עבור מופע $\langle S, t\rangle$ של הבעיה. את הבדיקה אם $t = \Sigma_{s \in S'} s$ אפשר לבצע בזמן פולינומיאלי על־ידי אלגוריתם אימות.

עתה נראה כי VERTEX-COVER $\leq_p$ SUBSET-SUM. בהינתן מופע $\langle G, k\rangle$ של בעיית הכיסוי על־ידי קדקודים, אלגוריתם הרדוקציה בונה מופע $\langle S, t\rangle$ של בעיית סכום התת־קבוצה כך ש־G יש כיסוי על־ידי קדקודים שגודלו k אם ורק אם קיימת תת־קבוצה של S שסכום איבריה הוא בדיוק t.

בליבה של הרדוקציה עומד ייצוג של G על־ידי מטריצת קשירויות. יהי $G = (V, E)$ גרף בלתי־מכוון, ויהיו $V = \{v_0, v_1, \ldots, v_{|V|-1}\}$ ו־$E = \{e_0, e_1, \ldots, e_{|E|-1}\}$. **מטריצת הקשירויות** (incidence matrix) של G היא מטריצה $B = (b_{ij})$ שממדיה $|V| \times |E|$ כך שמתקיים:

$$b_{ij} = \begin{cases} 1 & \text{אם הקשת } e_j \text{ קשורה לקדקוד } v_i \\ 0 & \text{אחרת} \end{cases}$$

לדוגמה, באיור 36.14(ii) מופיעה מטריצת הקשירויות של הגרף הבלתי־מכוון מאיור 36.14(i). במטריצת הקשירויות המוצגת, הקשתות מופיעות בסדר עולה של אינדקסים מימין לשמאל, ולא משמאל לימין כמקובל, כדי לפשט את הנוסחאות עבור המספרים ב־S.

בהינתן גרף G ושלם k, אלגוריתם הרדוקציה מחשב קבוצה S של מספרים ומספר שלם t. כדי להבין כיצד פועל אלגוריתם הרדוקציה, הבה נייצג מספרים ב"בסיס־4־מתוקן". $|E|$ הספרות הפחות משמעותיות של המספר ייוצגו בבסיס 4, אולם ערכה של הספרה המשמעותית ביותר יכול להגיע עד k. קבוצת המספרים נבנית כך שלא ניתן לקדם נשא מספרות מסדר נמוך לספרות מסדר גבוה יותר.

הקבוצה S מורכבת משני סוגים של מספרים, המתאימים לקדקודים ולקשתות בהתאמה. עבור כל קדקוד $v_i \in V$, אנו יוצרים שלם חיובי x_i שייצוגו בבסיס־4־מתוקן מורכב מ־1 בספרה

המשמעותית ביותר ואחריו $|E|$ ספרות. ספרות אלה מתאימות לשורה של v_i במטריצת הקשירויות $B = (b_{ij})$ של G, כמתואר באיור 36.14(iii). באופן פורמלי, עבור $i = 0, 1, \ldots, |V| - 1$:

$$x_i = 4^{|E|} + \sum_{j=0}^{|E|-1} b_{ij} 4^j$$

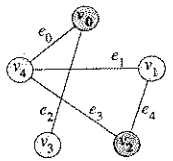
עבור כל קשת $e_j \in E$, אנו יוצרים שלם חיובי y_j שאינו אלא שורה ב"מטריצת היחידה" של מטריצות הקשירויות. (מטריצת היחידה של מטריצות הקשירויות היא המטריצה שממדיה $|E| \times |E|$ ובה מופיע הערך 1 רק באיברים שעל אלכסון המטריצה.) באופן פורמלי, עבור $j = 0, 1, \ldots, |E| - 1$:

$$y_j = 4^j$$

הספרה הראשונה של סכום המטרה t היא k, וכל $|E|$ הספרות מן הסדר הנמוך יותר הן 2. באופן פורמלי:

$$t = k4^{|E|} + \sum_{j=0}^{|E|-1} 2 \cdot 4^j$$

כל המספרים האלה הם בעלי גודל פולינומיאלי כאשר מייצגים אותם בכתיב בינרי. באמצעות פעולות על סיביות מטריצת הקשירויות, ניתן לבצע את הרדוקציה בזמן פולינומיאלי.



(i)

	e_4	e_3	e_2	e_1	e_0
v_0	0	0	1	0	1
v_1	1	0	0	1	0
v_2	1	1	0	0	0
v_3	0	0	1	0	0
v_4	0	1	0	1	1

(ii)

עשרוני — בסיס 4 מתוקן

	e_4	e_3	e_2	e_1	e_0	
x_0 =	1	0	0	1	0	1 = 1041
x_1 =	1	1	0	0	1	0 = 1284
x_2 =	1	1	1	0	0	0 = 1344
x_3 =	1	0	0	1	0	0 = 1040
x_4 =	1	0	1	0	1	1 = 1093
y_0 =	0	0	0	0	0	1 = 1
y_1 =	0	0	0	0	1	0 = 4
y_2 =	0	0	0	1	0	0 = 16
y_3 =	0	0	1	0	0	0 = 64
y_4 =	0	1	0	0	0	0 = 256
t =	3	2	2	2	2	2 = 3754

(iii)

איור 36.14 הרדוקציה של בעיית הכיסוי על־ידי קדקודים לבעיית סכום התת־קבוצה. **(i)** גרף בלתי־מכוון G. הקדקודים בהכרחים יוצרים כיסוי $\{v_1, v_3, v_4\}$ שגודלו 3. **(ii)** מטריצת הקשירויות המייצגת את הגרף. הצללת השורות מתאימה לכיסוי על־ידי קדקודים מחלק (i). עבור כל קשת e_j מופיע 1 בשורה בהירה אחת לפחות. **(iii)** המופע המתאים של בעיית סכום התת־קבוצה. החלק שבתוך הריבוע הוא מטריצת הקשירויות. כאן, הכיסוי $\{v_1, v_3, v_4\}$ שגודלו 3 מתאים לתת־קבוצה $\{1, 16, 64, 256, 1040, 1093, 1284\}$ שסכום איבריה 3754, והיא מיוצגת באיור על־ידי השורות הבהירות.

עתה עלינו להראות שלגרף G יש כיסוי על־ידי קדקודים שגודלו k אם ורק אם קיימת תת־קבוצה $S' \subseteq S$ שסכום איבריה t. תחילה, נניח שקיים עבור G כיסוי על־ידי קדקודים $V' \subseteq V$ שגודלו k. יהי $V' = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$, ונגדיר את S' על־ידי:

$$S' = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\} \cup \{y_j : V' \text{ קשורה בדיוק לקדקוד אחד ב־} e_j\}$$

כדי לראות כי $\Sigma_{s \in S'} s = t$, נשים לב שהספרה המשמעותית ביותר בכל אחד מ־k המספרים $x_{i_m} \in S'$ היא 1, ומסיכום הספרות האלה מתקבלת הספרה k, שהיא הספרה המשמעותית ביותר בייצוגו של t בבסיס־4־מתוקן. לקבלת הספרות הפחות משמעותיות של t, שכל אחת מהן היא 2, נתבונן במיקומים שלהן, שכל אחד מהם מתאים לקשת e_j, על פי הסדר. מכיוון ש־V' הוא כיסוי על־ידי קדקודים, e_j קשורה לפחות לקדקוד אחד ב־V'. לכן, עבור כל קשת e_j, קיים לפחות $x_{i_m} \in S'$ אחד שעבורו מופיע 1 במיקום j. אם e_j קשורה לשני קדקודים ב־V', אזי כל אחד מהם תורם 1 למיקום ה־j בסכום. הספרה ה־j של y_j אינה תורמת דבר, שכן e_j קשורה לשני קדקודים, ומכך נובע כי $y_j \notin S'$. לכן, במקרה זה, סכום איברי S' יוצר 2 במיקום ה־j של t. במקרה האחר — כאשר e_j קשורה לקדקוד אחד בדיוק ב־V' — מתקיים $x_j \in S'$ ו־y_j והקדקוד הקשור לקשת תורמים כל אחד 1 לסכום היוצר את הספרה ה־j של t, כך שגם במקרה זה מקבלים 2. לכן, S' היא פתרון למופע S של בעיית סכום התת־קבוצה.

עתה, נניח שקיימת תת־קבוצה $S' \subseteq S$ שסכום איבריה t.

תהי $\{ x_{i_1}, x_{i_2}, \ldots, x_{i_m} \} \cup \{ y_{j_1}, y_{j_2}, \ldots, y_{j_p} \}$ וכי $m = k$. אנו טוענים כי S'. $V' = \{ v_{i_1}, v_{i_2}, \ldots, v_{i_m} \}$ הוא כיסוי על־ידי קדקודים עבור G. כדי להוכיח טענה זו, נשים לב תחילה כי עבור כל קשת $e_j \in E$, הקבוצה S מכילה שלושה שלמים שבמיקום ה־e_j שלהם מופיע 1: אחד עבור כל אחד משני הקדקודים הקשורים לקשת e_j, ואחד מ־y_j. מכיוון שאנו עובדים בייצוג בבסיס 4־מתוקן, לא מועבר נשא מהמיקום ה־e_j למיקום ה־e_{j+1}. לכן, עבור כל אחד מ־$|E|$ המיקומים הפחות משמעותיים של t, בהכרח לפחות x_i אחד ולכל היותר שני x_i תורמים לסכום. מאחר שלפחות x_i אחד תורם לסכום עבור כל קשת, הרי ש־V' הוא כיסוי על־ידי קדקודים. כדי להיווכח כי $m = k$ ולכן V' הוא כיסוי על־ידי קדקודים שגודלו k, נשים לב כי הדרך היחידה שבה ניתן לקבל k בספרה המשמעותית ביותר של המטרה t, היא לכלול בסכום בדיוק k מבין ה־x_i. ■

באיור 36.14, הכיסוי על־ידי קדקודים $V' = \{ v_1, v_3, v_4 \}$ מתאים לתת־קבוצה $S' = \{ x_1, x_3, x_4, y_0, y_2, y_3, y_4 \}$. כל ה־$y_i$ נכללים ב־S', פרט ל־y_1, שמתאים לקשת הקשורה לשני קדקודים ב־V'.

36.5.4 בעיית המעגל ההמילטוני

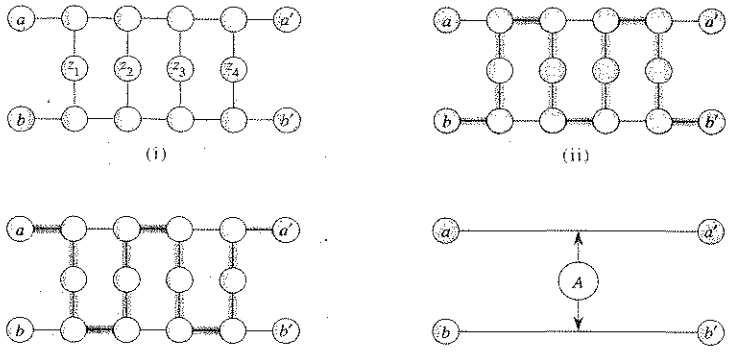
נחזור עתה לבעיית המעגל ההמילטוני שהוגדרה בסעיף 36.2.

משפט 36.14
בעיית המעגל ההמילטוני היא NP־שלמה.

הוכחה תחילה נראה כי HAM-CYCLE שייכת ל־NP. בהינתן גרף $G = (V, E)$, האישור שלנו הוא הסדרה בת $|V|$ הקדקודים היוצרים את המעגל ההמילטוני. אלגוריתם האימות בודק שאכן סדרה זו מכילה כל קדקוד מ־V בדיוק פעם אחת, וכן שהיא מסתיימת בקדקוד שבו התחילה, ויוצרת מעגל ב־G. אימות זה ניתן לביצוע בזמן פולינומיאלי.

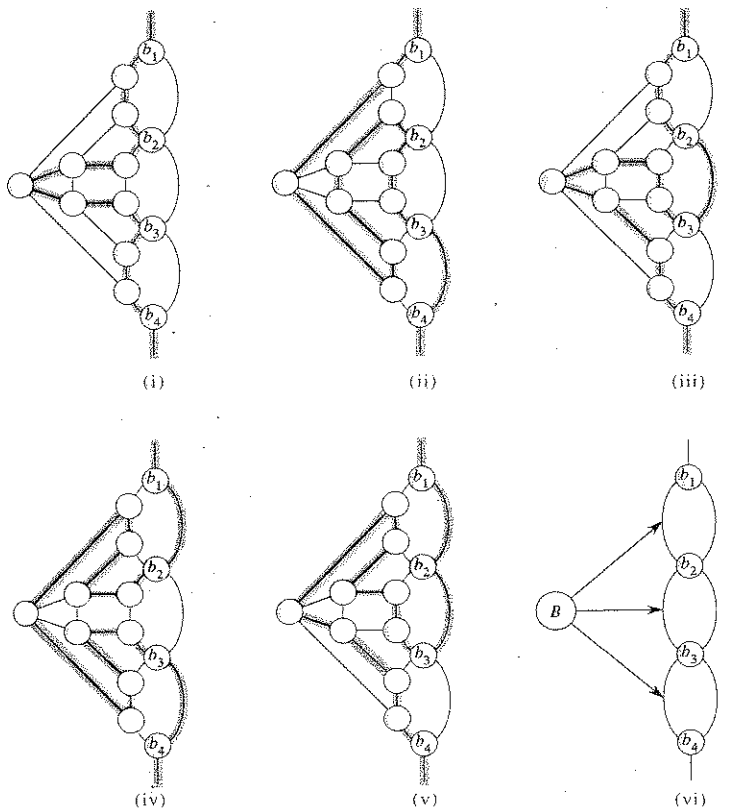
עתה נוכיח כי HAM-CYCLE היא NP־שלמה בכך שנראה כי 3-CNF-SAT $\leq_p$ HAM-CYCLE. בהינתן נוסחת 3-CNF בוליאנית ϕ מעל המשתנים $x_1, x_2, \ldots, x_n$ המורכבת מהפסוקיות $C_1, C_2, \ldots, C_k$ שכל אחת מהן מכילה בדיוק 3 ליטרלים שונים, אנו בונים בזמן פולינומיאלי גרף $G = (V, E)$ כך ש־G מכיל מעגל המילטוני אם ורק אם הנוסחה ϕ ספיקה. בנייתנו מבוססת על **רכיבי בנייה קטנים** (widgets) — חלקי גרף הכופים תכונות מסוימות.

הרכיב הראשון שלנו הוא התת־גרף A המתואר באיור 36.15(i). נניח כי A הוא תת־גרף של איזשהו גרף G, וכי A מחובר לשאר הגרף G בקדקודים a, a', b ו־b' בלבד. עוד נניח כי G מכיל מעגל המילטוני. מכיוון שכל מעגל המילטוני ב־G חייב לעבור דרך הקדקודים z_1, z_2, z_3 ו־z_4 באחת מן הדרכים המוצגות באיורים 36.15(ii)-(iii), נוכל להתייחס אל התת־גרף A כאילו היה פשוט זוג הקשתות (a, a') ו־(b, b'), עם ההגבלה שכל מעגל המילטוני ב־G חייב לכלול בדיוק אחת משתי קשתות אלה. נייצג את הרכיב A בצורה המתוארת באיור 36.15(iv).



איור 36.15 **(i)** רכיב הבנייה A המשמש ברדוקציה מ־3-CNF-SAT ל־HAM-CYCLE. **(ii)-(iii)** אם A הוא תת־גרף של איזשהו גרף G המכיל מעגל המילטוני ו־A מחובר לשאר הגרף G בקדקודים a, a', b ו־b' בלבד, אזי הקשתות המוצללות מייצגות את שתי הדרכים האפשריות היחידות שבהן יכול מעגל המילטוני לעבור דרך קשתות תת־גרף A. **(iv)** ייצוג קומפקטי של הרכיב A.

תת־גרף B שבאיור 36.16 הוא רכיב הבנייה השני שלנו. נניח כי B הוא תת־גרף של איזשהו גרף G וכי B מחובר לשאר הגרף G רק בקדקודים b_1, b_2, b_3 ו־b_4. מעגל המילטוני בגרף G אינו יכול לעבור דרך כל הקשתות (b_1, b_2), (b_2, b_3) ו־(b_3, b_4), משום שאז המעגל יעבור רק דרך הקדקודים b_1, b_2, b_3 ו־b_4 אבל לא דרך הקדקודים האחרים ברכיב. אבל, מעגל המילטוני ב־G יכול לעבור דרך כל תת־קבוצה ממש של קשתות אלה. באיורים 36.16(i)-(v) מוצגות חמש תת־קבוצות כאלה; את שתי התת־קבוצות הנותרות ניתן לקבל על־ידי החלפה בין החלק העליון והתחתון באיורים 36.16(ii) ו־36.16(v). אנו מייצגים רכיב זה בצורה המתוארת באיור 36.16(vi), כשהרעיון הוא שמעגל המילטוני חייב לעבור לפחות דרך אחד המסלולים שהחיצים מצביעים עליהם.



איור 36.16 רכיב הבנייה B המשמש ברדוקציה מ־3-CNF-SAT ל־HAM-CYCLE. לא ייתכן שמסלול מהקדקוד b_1 לקדקוד b_4 המכיל את כל הקדקודים ברכיב יעבור דרך כל שלוש הקשתות (b_1, b_2), (b_2, b_3) ו־(b_3, b_4). אולם, מסלול כזה יכול לעבור דרך כל תת־קבוצה ממש של קשתות אלה. **(i)-(v)** חמש תת־קבוצות כאלה. **(vi)** ייצוג של רכיב זה שבו מעגל המילטוני חייב לעבור לפחות דרך אחד המסלולים שהחיצים מצביעים עליהם.

הגרף G שנבנה מורכב ברובו מעותקים של שני רכיבי בנייה אלה. הבנייה מודגמת באיור 36.17. עבור כל אחת מ־k הפסוקיות ב־ϕ, אנו כוללים ב־G עותק של הרכיב B, ואנו מחברים את הרכיבים בטור באופן הבא: אם b_{ij} הוא העותק של הקדקוד b_j בעותק ה־i של הרכיב B, אנו מחברים את $b_{i,4}$ ל־$b_{i+1,1}$, עבור $i = 1, 2, \ldots, k-1$.

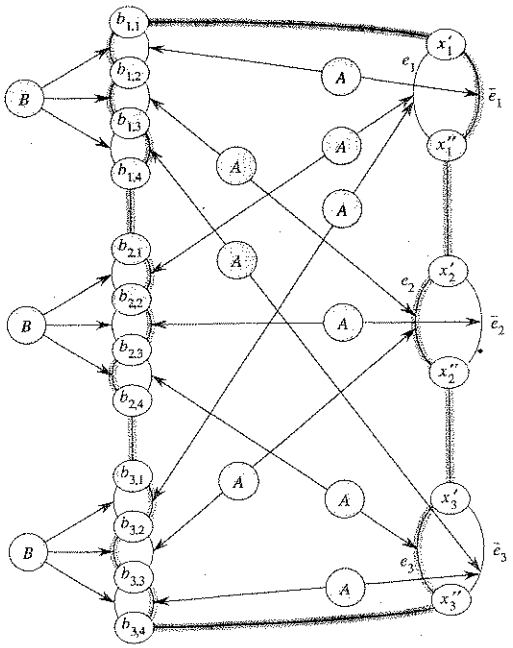
לאחר מכן, עבור כל משתנה x_m ב־ϕ, אנו כוללים ב־G שני קדקודים, x'_m ו־x''_m. אנו מחברים את שני הקדקודים האלה באמצעות שני עותקים של הקשת (x'_m, x''_m), שכדי להבחין ביניהם, אנו מסמנים אותם על־ידי e_m ו־$\bar{e}_m$. הרעיון הוא שאם המעגל ההמילטוני עובר דרך הקשת e_m, הדבר מתאים להשמה של הערך 1 במשתנה x_m. אם המעגל ההמילטוני עובר דרך הקשת $\bar{e}_m$, הדבר מתאים להשמה של הערך 0 במשתנה x_m. כל אחד מזוגות הקשתות הללו יוצר לולאה בת שתי קשתות; את הלולאות הקטנות האלה אנו מחברים בטור על־ידי הוספת הקשתות (x'_m, x''_{m+1}), עבור $m = 1, 2, \ldots, n-1$. אנו מחברים את צידו השמאלי של הגרף (צד הפסוקיות) לצידו

הימני (צד המשתנים) באמצעות שתי הקשתות $(b_{1,1}, x'_1)$ ו־$(b_{k,4}, x''_n)$, הקשת העליונה והקשת התחתונה באיור 36.17.

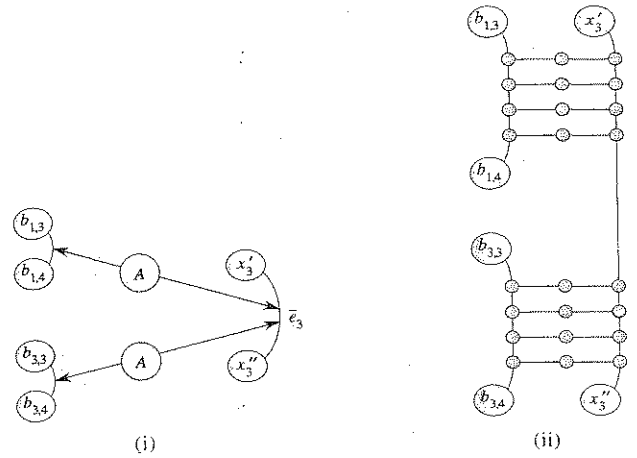
עדיין לא סיימנו את בנייתו של הגרף G, שכן עדיין עלינו לקשור בין המשתנים לבין הפסוקיות. אם הליטרל ה־j של הפסוקית C_i הוא x_m, אזי אנו משתמשים ברכיב A לחיבור הקשת $(b_{ij}, b_{i,j+1})$ לקשת e_m. אם הליטרל ה־j של הפסוקית C_i הוא $\neg x_m$, אזי אנו מחברים באמצעות רכיב A את הקשת $(b_{ij}, b_{i,j+1})$ לקשת $\bar{e}_m$. לדוגמה, באיור 36.17, מכיוון שהפסוקית C_2 היא $(x_3 \vee \neg x_2 \vee x_3)$, אנו כוללים בגרף שלושה רכיבי A באופן הבא:

- בין $(b_{2,1}, b_{2,2})$ ל־e_1,
- בין $(b_{2,2}, b_{2,3})$ ל־$\bar{e}_2$,
- בין $(b_{2,3}, b_{2,4})$ ל־e_3.

נשים לב כי חיבור שתי קשתות באמצעות רכיב A כרוך למעשה בהחלפת כל קשת בחמש הקשתות המופיעות בחלק העליון או התחתון של איור 36.15(i) וכן, כמובן, בהוספת החיבורים העוברים דרך הקדקודים z. ליטרל נתון l_m עשוי להופיע בכמה פסוקיות ($\neg x_3$ באיור 36.17, למשל), ולכן קשת e_m או $\bar{e}_m$ עשויה להיות מושפעת מכמה רכיבי A (הקשת $\bar{e}_3$, למשל). במקרה זה, אנו מחברים את רכיבי A בטור, כמודגם באיור 36.18, כך שלמעשה אנו מחליפים כל קשת e_m או $\bar{e}_m$ בסדרה של קשתות.



איור 36.17 הגרף G הנבנה מן הנוסחה $\phi = (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$. השמה מספקת s עבור המשתנים של ϕ היא $s(x_1) = 0$, $s(x_2) = 1$ ו־$s(x_3) = 1$, והיא מתאימה למעגל ההמילטוני המופיע באיור. יש לשים לב שאם $s(x_m) = 1$, אזי הקשת e_m נמצאת על המעגל ההמילטוני, ואם $s(x_m) = 0$ אזי הקשת $\bar{e}_m$ נמצאת על המעגל ההמילטוני.



איור 36.18 הבנייה המתבצעת בפועל כאשר קשת e_m או $\bar{e}_m$ מושפעת מכמה רכיבי A. **(i)** חלק מאיור 36.17. **(ii)** התת־גרף הנבנה בפועל.

אנו טוענים שהנוסחה ϕ ספיקה אם ורק אם הגרף G מכיל מעגל המילטוני. תחילה נניח כי G מכיל מעגל המילטוני h ונראה ש־ϕ ספיקה. המעגל h חייב להיות בעל צורה מסוימת:

- תחילה, הוא עובר מן החלק השמאלי העליון של הגרף לחלקו הימני העליון דרך הקשת $(b_{1,1}, x'_1)$.
- לאחר־מכן הוא עובר מלמעלה למטה דרך כל הקדקודים x'_m ו־x''_m, כאשר בכל פעם הוא בוחר באחת משתי הקשתות e_m או $\bar{e}_m$, אולם לא בשתיהן.
- אחר־כך הוא עובר דרך הקשת $(b_{k,4}, x''_n)$ כדי לחזור לצד שמאל.
- לבסוף, הוא עובר מלמטה למעלה בצידו השמאלי של הגרף דרך רכיבי B.

(למעשה, המעגל עובר גם דרך קשתות שבתוך רכיבי A, אולם אנו משתמשים בתת־גרפים אלה כדי לכפות את האופן "או/או" שבו בוחר המעגל בקשתות שהוא מחבר.)

בהינתן המעגל ההמילטוני h, אנו מגדירים השמת ערכי אמת עבור ϕ באופן הבא: אם קשת e_m שייכת ל־h, אזי אנו מציבים $x_m = 1$. אחרת, הקשת $\bar{e}_m$ שייכת ל־h, ואנו מציבים $x_m = 0$.

אנו טוענים כי השמה זו מספקת את ϕ. נתבונן בפסוקית C_i וברכיב B המתאים לה ב־G. כל קשת $(b_{i,j}, b_{i,j+1})$ מחוברת על־ידי רכיב A לאחת משתי הקשתות e_m (אם הליטרל ה־j בפסוקית הוא x_m) או $\bar{e}_m$ (אם הליטרל ה־j בפסוקית הוא $\neg x_m$). h עובר דרך הקשת $(b_{i,j}, b_{i,j+1})$ אם ורק אם ערכו של הליטרל המתאים הוא 0. מאחר שכל שלוש הקשתות $(b_{i,1}, b_{i,2})$, $(b_{i,2}, b_{i,3})$ ו־$(b_{i,3}, b_{i,4})$ בפסוקית C_i נמצאות ברכיב B אחד, המעגל ההמילטוני h אינו יכול לעבור דרך שלושתן. לכן, לאחת משלוש הקשתות הללו חייב להתאים ליטרל שהערך שהושם לו הוא 1, ולכן ההשמה מספקת את הפסוקית C_i. תכונה זו מתקיימת עבור כל פסוקית C_i, כאשר $i = 1, 2, \ldots, k$, ולכן ההשמה מספקת את הנוסחה ϕ.

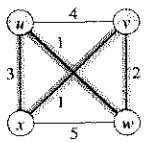
בכיוון ההפוך, נניח כי קיימת השמת ערכי אמת s המספקת את הנוסחה ϕ. על פי הכללים שלעיל, נוכל לבנות ב־G מעגל המילטוני: אם $x_m = 1$ נעבור דרך הקשת e_m, אם $x_m = 0$ נעבור דרך הקשת $\bar{e}_m$, ונעבור דרך הקשת $(b_{i,j}, b_{i,j+1})$ אם ורק אם בליטרל ה־j של הפסוקית C_i הציבה ההשמה את הערך 0. בנייתו של מעגל על פי כללים אלה אכן אפשרית מכיוון שאנו מניחים כי s היא השמה מספקת עבור הנוסחה ϕ.

לבסוף, יש לציין כי ניתן לבנות את הגרף G בזמן פולינומיאלי. הגרף מכיל רכיב B אחד עבור כל אחת מ־k הפסוקיות ב־ϕ. ישנו רכיב A אחד עבור כל מופע של כל ליטרל ב־ϕ, ולכן ישנם $3k$ רכיבי A. מאחר שרכיבי A ו־B הם בעלי גודל קבוע, הגרף G מורכב מ־$O(k)$ קדקודים וקשתות ולכן אפשר לבנותו בקלות בזמן פולינומיאלי. מצאנו אפוא רדוקציית זמן־פולינומיאלי מ־3-CNF-SAT ל־HAM-CYCLE. ■

36.5.5 בעיית הסוכן הנוסע

בעיית הסוכן הנוסע (the traveling salesman problem), הקשורה קשר הדוק לבעיית המעגל ההמילטוני, עוסקת בסוכן נוסע שמוטל עליו לבקר ב־n ערים. אם נייצג את הבעיה על־ידי גרף שלם בעל n קדקודים, נוכל לומר שהסוכן הנוסע מעוניין להרכיב לעצמו **מסלול סיור** או **נתיב** (tour), או מעגל המילטוני, שיעבור דרך כל עיר בדיוק פעם אחת ויסתיים בעיר שבה החל. הנסיעה מעיר i לעיר j כרוכה בעלות של $c(i, j)$ שהיא מספר שלם, והסוכן שלנו מעוניין להרכיב נתיב שעלותו הכוללת מינימלית, כאשר העלות הכוללת היא סכום העלויות הנפרדות של קשתות הנתיב. לדוגמה, בגרף שבאיור 36.19, נתיב בעל עלות מינימלית הוא $\langle u, w, v, x, u \rangle$, שעלותו 7. השפה הפורמלית עבור בעיית ההכרעה המתאימה לבעיית הסוכן הנוסע היא:

$$\text{TSP} = \{\langle G, c, k \rangle : G = (V, E) \text{ הוא גרף שלם},$$
$$c \text{ היא פונקציה מ־} V \times V \text{ ל־} \mathbf{Z},$$
$$k \in \mathbf{Z} \text{, וכן}$$
$$G \text{ מכיל נתיב עבור הסוכן הנוסע שעלותו לכל היותר } k\}$$



איור 36.19 מופע של בעיית הסוכן הנוסע. הקשתות המוצללות יוצרות נתיב בעל עלות מינימלית שעלותו 7.

המשפט הבא מראה כי אין זה סביר שקיים אלגוריתם מהיר לפתרון בעיית הסוכן הנוסע.

משפט 36.15
בעיית הסוכן הנוסע היא NP־שלמה.

הוכחה תחילה נראה כי TSP שייכת ל־NP. בהינתן מופע של הבעיה, האישור שנשתמש בו הוא סדרת n קדקודי הנתיב. אלגוריתם האימות בודק אם סדרה זו אכן מכילה כל קדקוד בדיוק פעם אחת, מסכם את עלויות הקשתות, ובודק אם הסכום אינו עולה על k. תהליך זה בוודאי ניתן לביצוע בזמן פולינומיאלי.

כדי להוכיח כי TSP היא NP־קשה, נראה כי HAM-CYCLE $\leq_P$ TSP. יהי $G = (V, E)$ מופע של HAM-CYCLE. נבנה מופע של TSP כדלקמן. ניצור את הגרף השלם $G' = (V, E')$, שבו $E' = \{(i, j) : i, j \in V\}$, ונגדיר את פונקציית העלות c על־ידי:

$$c(i, j) = \begin{cases} 0 & \text{אם } (i, j) \in E \\ 1 & \text{אם } (i, j) \notin E \end{cases}$$

המופע של TSP הוא אפוא $(G', c, 0)$, וקל לבנותו בזמן פולינומיאלי.

עתה נראה שהגרף G מכיל מעגל המילטוני אם ורק אם הגרף G' מכיל נתיב שעלותו לכל היותר 0. נניח שהגרף G מכיל מעגל המילטוני h. כל קשת ב־h שייכת ל־E ולכן עלותה ב־G' היא 0. לכן, h הוא נתיב ב־G' שעלותו 0. בכיוון ההפוך, נניח שהגרף G' מכיל נתיב h' שעלותו לכל היותר 0. מכיוון שעלויותיהן של הקשתות ב־E' הן 0 או 1, עלותו של הנתיב h' היא בדיוק 0. לכן, h' מכיל אך ורק קשתות השייכות ל־E. המסקנה היא ש־h' הוא מעגל המילטוני בגרף G. ■

תרגילים

36.5-1

בעיית התת־גרף האיזומורפי (the subgraph-isomorphism problem) מקבלת כקלט שני גרפים G_1 ו־G_2, ושואלת אם G_1 איזומורפי לתת־גרף של G_2. הראה שבעיית התת־גרף האיזומורפי היא NP־שלמה.

36.5-2

בהינתן מטריצת שלמים A שממדיה $m \times n$ ווקטור שלמים b שגודלו m, **גרסת 1-0 של בעיית התכנון בשלמים** (the 0-1 integer-programming problem) שואלת אם קיים וקטור x שגודלו n ואיבריו הם שלמים השייכים לקבוצה $\{0, 1\}$, כך ש־$Ax \leq b$. הוכח שבעיה זו היא NP־שלמה. (**רמז**: בצע רדוקציה מ־3-CNF-SAT.)

36.5-3

הראה שבעיית סכום התת־קבוצה פתירה בזמן פולינומיאלי אם ערך המטרה t מיוצג בכתיב אונרי.

36.5-4

בעיית החלוקה של קבוצה (the set-partitioning problem) מקבלת כקלט קבוצה S של מספרים, ושואלת אם ניתן לחלק את המספרים לשתי קבוצות, A ו־$\overline{A} = S - A$, כך שיתקיים $\sum_{x \in A} x = \sum_{x \in \overline{A}} x$. הראה שבעיית החלוקה של קבוצה היא NP־שלמה.

36.5-5

הראה שבעיית הנתיב ההמילטוני היא NP־שלמה.

36.5-6

בעיית המעגל הפשוט הארוך ביותר (the longest-simple-cycle problem) היא בעיית מציאתו של מעגל פשוט (דהיינו, מעגל שאינו עובר דרך קדקוד יותר מפעם אחת) שאורכו מקסימלי בגרף. הראה שבעיה זו היא NP־שלמה.

36.5-7

פרופסור מארקוני מכריז שהתת־גרף המשמש כרכיב הבנייה A בהוכחת משפט 36.14 מסובך מהנדרש: אין צורך בקדקודים z_3 ו־z_4 מאיור 36.15(i) ולא בקדקודים שמעליהם ומתחתם. האם הפרופסור צודק? כלומר, האם הרדוקציה פועלת כהלכה גם כאשר משתמשים ברכיב קטן יותר זה, או שתכונת ה"אי־אור" של הרכיב נעלמת?

בעיות

36-1 קבוצה בלתי־תלויה

קבוצה בלתי־תלויה (independent set) בגרף $G = (V, E)$ היא תת־קבוצה $V' \subseteq V$ של קדקודים כך שכל קשת ב־E קשורה לכל היותר לקדקוד אחד ב־V'. **בעיית הקבוצה הבלתי־תלויה** (the independent-set problem) היא למצוא ב־G קבוצה בלתי־תלויה שגודלה מקסימלי.

א. נסח את בעיית ההכרעה המתאימה לבעיית הקבוצה הבלתי־תלויה והוכח שהיא NP־שלמה. (**רמז**: בצע רדוקציה מבעיית הקליקה.)

ב. נניח שנתונה לך שגרה הפותרת את בעיית ההכרעה שניסחת בחלק (א). כתוב אלגוריתם המוצא קבוצה בלתי־תלויה שגודלה מקסימלי. זמן הריצה של האלגוריתם שלך צריך להיות פולינומיאלי ב־$|V|$ וב־$|E|$, כאשר כל שאילתה לקופסה השחורה נחשבת לצעד יחיד.

אף כי בעיית ההכרעה של הקבוצה הבלתי־תלויה היא NP־שלמה, קיימים מקרים מיוחדים מסוימים שניתן לפותרם בזמן פולינומיאלי.

ג. תאר אלגוריתם יעיל הפותר את בעיית הקבוצה הבלתי־תלויה כאשר דרגתו של כל קדקוד ב־G היא 2. נתח את זמן הריצה והוכח את נכונותו של האלגוריתם שלך.

ד. תאר אלגוריתם יעיל הפותר את בעיית הקבוצה הבלתי־תלויה כאשר G הוא דו־צדדי. נתח את זמן הריצה והוכח את נכונותו של האלגוריתם שלך. (**רמז**: השתמש בתוצאות מסעיף 27.3.)

36-2 צביעת גרף

צביעת־k (k-coloring) של גרף בלתי־מכוון $G = (V, E)$ היא פונקציה $c : V \to \{1, 2, \ldots, k\}$ המקיימת $c(u) \neq c(v)$ עבור כל קשת $(u, v) \in E$. במילים אחרות, המספרים $1, 2, \ldots, k$ מייצגים k צבעים, וקדקודים סמוכים חייבים להיות בעלי צבעים שונים. **בעיית צביעת הגרף** (the graph-coloring problem) שואלת מהו המספר המינימלי של צבעים הדרוש לצביעת גרף נתון.

א. תאר אלגוריתם יעיל המוצא צביעת־2 של גרף, אם קיימת צביעה כזאת.

ב. נסח את בעיית צביעת הגרף כבעיית הכרעה. הראה שבעיית ההכרעה שניסחת פתירה בזמן פולינומיאלי אם ורק אם בעיית צביעת הגרף המקורית פתירה בזמן פולינומיאלי.

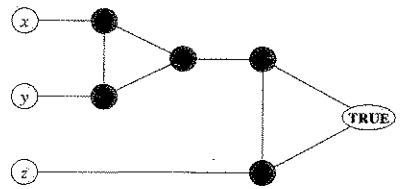
ג. תהי השפה 3-COLOR קבוצת הגרפים שניתן לצובעם בצביעת־3. הראה שאם 3-COLOR היא NP־שלמה, אזי בעיית ההכרעה שניסחת בחלק (ב) גם היא NP־שלמה.

כדי להראות כי 3-COLOR היא NP־שלמה, נשתמש ברדוקציה מ־3-CNF-SAT. בהינתן נוסחה ϕ בת m פסוקיות ו־n משתנים, $x_1, x_2, \ldots, x_n$, נבנה גרף $G = (V, E)$ באופן הבא. הקבוצה V מורכבת מקדקוד אחד עבור כל משתנה, קדקוד אחד עבור שלילתו של כל משתנה, 5 קדקודים

עבור כל פסוקית, ו־3 קדקודים מיוחדים: TRUE, FALSE ו־RED. קשתות הגרף הן משני סוגים: קשתות "ליטרל" שאינן תלויות בפסוקיות, וקשתות "פסוקית" התלויות בפסוקיות. קשתות הליטרל מחברות למשולש את הקדקודים המיוחדים, וכן מחברות למשולש את הקדקודים x_i ו־$\neg x_i$, עבור $i = 1, 2, \ldots, n$.

ד. הראה כי בכל צביעת־3 c של גרף המכיל את קשתות הליטרל, בדיוק אחד מבין משתנה ושלילתו נצבע בצבע $c(\text{TRUE})$ והאחר נצבע בצבע $c(\text{FALSE})$. הראה כי עבור כל השמת ערכי אמת ב־ϕ, קיימת צביעת־3 של הגרף המכילה את קשתות הליטרל בלבד.

רכיב הבנייה המוצג באיור 36.20 משמש כדי לכפות את התנאי המתאים לפסוקית $(x \vee y \vee z)$. לכל פסוקית דרוש עותק ייחודי של חמשת הקדקודים הצבועים שחור באיור; הם מחוברים כמתואר באיור לליטרלים של הפסוקית ולקדקוד המיוחד TRUE.



איור 36.20 רכיב הבנייה המתאים לפסוקית $(x \vee y \vee z)$, עבור בעיה 36-2.

ה. הראה שאם כל אחד מן הקדקודים x, y ו־z נצבע ב־$c(\text{TRUE})$ או ב־$c(\text{FALSE})$, אזי הרכיב ניתן לצביעת־3 אם ורק אם לפחות אחד מן הקדקודים x, y ו־z נצבע בצבע $c(\text{TRUE})$.

ו. השלם את ההוכחה כי 3-COLOR היא NP-שלמה.

הערות ביבליוגרפיות לפרק 36

Garey & Johnson [79] מספקים מדריך נפלא לשלמות ב־NP, הכולל דיון מפורט בתיאוריה ורשימה של בעיות רבות שהיו ידועות כ־NP-שלמות בשנת 1979. (רשימת התחומים שבהם קיימות בעיות NP-שלמות, המופיעה בתחילת סעיף 36.5, לקוחה מתוכן העניינים שלהם.) אצל Hopcroft & Ullman [104] וכן אצל Lewis & Papadimitriou [139] מצוי דיון טוב בשלמות ב־NP בהקשר של תורת הסיבוכיות. Aho, Hopcroft & Ullman [4] סוקרים גם הם שלמות ב־NP ומציגים כמה רדוקציות, ובהן רדוקציה של בעיית הכיסוי על־ידי קדקודים מבעיית המעגל ההמילטוני.

המחלקה P הוגדרה בשנת 1964 על־ידי Cobham [44] ובאופן בלתי־תלוי בשנת 1965 על־ידי Edmonds [61] שהוגדר גם את המחלקה NP והעלה את ההשערה כי $P \neq NP$. המושג "שלמות ב־NP" הוצע בשנת 1971 על־ידי Cook [49] שהציג את ההוכחות הראשונות לשלמות ב־NP של ספיקות נוסחאות ושל ספיקות נוסחאות 3-CNF. Levin [138] גילה את המושג באופן בלתי־תלוי, והציג הוכחה של השלמות ב־NP של בעיית ריצוף. Karp [116] הציג בשנת 1972 את שיטות הרדוקציות והדגים את המגוון העשיר של בעיות NP-שלמות. מאמרו של Karp כלל את ההוכחות המקוריות של השלמות ב־NP של בעיית הקליקה, בעיית הכיסוי על־ידי קדקודים ובעיית המעגל ההמילטוני. מאז הוכיחו חוקרים רבים את השלמות ב־NP של מאות בעיות.

37 אלגוריתמי קירוב

בעיות רבות בעלות חשיבות מעשית הן NP-שלמות אבל הן חשובות מכדי שנזנח אותן רק משום שמציאת פתרון אופטימלי היא בלתי מעשית. אם בעיה היא NP-שלמה, אין סיכוי רב שנמצא אלגוריתם זמן־פולינומיאלי שפותר אותה במדויק, אולם אין פירוש הדבר שעלינו לאבד כל תקווה. קיימות שתי גישות לעקיפת בעיית השלמות ב־NP. ראשית, אם בפועל הקלטים קטנים, אזי גם ביצועיו של אלגוריתם אקספוננציאלי עשויים להיות מספקים. בהחלט. שנית, עדיין ייתכן שניתן למצוא בזמן פולינומיאלי פתרונות *כמעט־אופטימליים* (בין אם במקרה הגרוע או במקרה הממוצע). באופן מעשי, פתרונות כמעט־אופטימליים הם לעתים קרובות טובים דיים. אלגוריתם המחזיר פתרונות כמעט־אופטימליים נקרא **אלגוריתם קירוב** (approximation algorithm). בפרק זה נציג אלגוריתמי קירוב בעלי זמן ריצה פולינומיאלי עבור כמה בעיות NP-שלמות.

חסמים על ביצועיהם של אלגוריתמי קירוב

נניח שברצוננו למצוא פתרון כמעט־אופטימלי לבעיית אופטימיזציה שבה לכל פתרון אפשרי יש עלות חיובית. על פי הבעיה, ניתן להגדיר פתרון אופטימלי כפתרון שעלותו מקסימלית או כפתרון שעלותו מינימלית; הבעיה עשויה אפוא להיות בעיית מקסימום או מינימום.

אנו אומרים שאלגוריתם קירוב עבור הבעיה הוא בעל חסם יחס (ratio bound) של $\rho(n)$ אם עבור כל קלט בגודל n, היחס בין העלות C של הפתרון שמפיק אלגוריתם הקירוב לבין העלות C^* של פתרון אופטימלי אינו עולה על $\rho(n)$:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n) \qquad (37.1)$$

הגדרה זו ישימה הן לבעיות מקסימום והן לבעיות מינימום. עבור בעיית מקסימום, $0 < C \leq C^*$, והיחס C^*/C הוא הגורם המציין פי כמה גדולה העלות של פתרון אופטימלי מעלותו של הפתרון המקורב. באופן דומה, עבור בעיית מינימום, $0 < C^* \leq C$, והיחס C/C^* הוא הגורם המציין פי כמה גדולה עלותו של הפתרון המקורב מעלותו של פתרון אופטימלי. מאחר שההנחה היא כי כל הפתרונות הם בעלי עלות חיובית, יחסים אלה תמיד מוגדרים היטב. חסם היחס של אלגוריתם קירוב לעולם אינו קטן מ־1, שכן $C/C^* < 1$ גורר $C^*/C > 1$. אלגוריתם אופטימלי הוא בעל חסם יחס של 1, ואלגוריתם קירוב בעל חסם יחס גדול עלול להחזיר פתרון שהוא גרוע בהרבה מפתרון אופטימלי.

לעתים נוח יותר להשתמש במדד של שגיאה יחסית. עבור קלט כלשהו, **השגיאה היחסית** (relative error) של אלגוריתם הקירוב מוגדרת על־ידי:

$$\frac{|C - C^*|}{C^*}$$

כאשר כמו קודם, C^* היא העלות של פתרון אופטימלי ו־C היא העלות של הפתרון שמפיק אלגוריתם הקירוב. השגיאה היחסית היא תמיד אי־שלילית. אלגוריתם קירוב הוא בעל **חסם שגיאה יחסית** (relative error bound) של $\epsilon(n)$ אם:

$$\frac{|C - C^*|}{C^*} \leq \epsilon(n) \tag{37.2}$$

מההגדרות שלעיל נובע הקשר שלהלן בין חסם השגיאה היחסית לבין חסם היחס:

$$\epsilon(n) \leq \rho(n) - 1 \tag{37.3}$$

(עבור בעיית מינימום זהו שוויון, ואילו עבור בעיית מקסימום, מקבלים $\epsilon(n) = (\rho(n) - 1)/\rho(n)$, המספק את אי־שוויון (37.3), שכן $\rho(n) \geq 1$.)

עבור בעיות רבות פותחו אלגוריתמי קירוב בעלי חסם יחס קבוע, שאינו תלוי ב־n. עבור בעיות כאלה נשתמש פשוט בסימונים ρ ו־ϵ, כדי לציין את האי־תלות ב־n.

עבור בעיות אחרות לא הצליחו מדעני מחשב למצוא אלגוריתמי קירוב שרצים בזמן פולינומיאלי והם בעלי חסם יחס קבוע. עבור בעיות כאלה, כל שניתן לעשות הוא להרשות לחסם היחס לגדול כפונקציה של גודל הקלט n. דוגמה לבעיה כזאת היא בעיית כיסוי הקבוצה שתוצג בסעיף 37.3.

ישנן בעיות NP־שלמות שעבורן אפשר למצוא אלגוריתמי קירוב שיכולים להשיג חסמי יחס (או, באופן שקול, חסמי שגיאה יחסיות) שהולכים וקטנים ככל שמשתמשים ביותר ויותר זמן חישוב. כלומר, מתקיימים יחסי תן־וקח בין זמן החישוב לבין איכות הקירוב. דוגמה לכך היא בעיית סכום התת־קבוצה שנחקור בסעיף 37.4. מצב זה חשוב דיו כדי שיהיה ראוי לשם משלו.

סכמת קירוב (approximation scheme) עבור בעיית אופטימיזציה היא אלגוריתם קירוב המקבל כקלט לא רק מופע של הבעיה אלא גם ערך $\epsilon > 0$ כך שעבור כל ϵ קבוע, הסכמה היא אלגוריתם קירוב בעל חסם שגיאה יחסית ϵ. אנו אומרים כי סכמת קירוב היא **סכמת קירוב פולינומיאלית** (polynomial-time approximation scheme) אם עבור כל $\epsilon > 0$ קבוע, זמן הריצה של הסכמה הוא פולינומיאלי ב־n, גודלו של מופע הקלט.

זמן הריצה של סכמת קירוב פולינומיאלית אינו אמור לגדול מהר מדי כאשר ϵ קטן. באופן אידיאלי, אם ϵ קטן פי גורם קבוע, זמן הריצה לחישוב הקירוב המבוקש אינו צריך לגדול פי יותר מאשר גורם קבוע. במילים אחרות, היינו רוצים שזמן הריצה יהיה פולינומיאלי לא רק ב־n אלא גם ב־$1/\epsilon$.

סכמת קירוב נקראת **סכמת קירוב פולינומיאלית לחלוטין** (fully polynomial-time approximation scheme) אם זמן הריצה שלה פולינומיאלי ב־$1/\epsilon$ וגם בגודל n של מופע הקלט, כאשר ϵ הוא חסם השגיאה היחסית עבור סכמה זו. לדוגמה, הסכמה עשויה להיות בעלת זמן ריצה של $(1/\epsilon)^2 n^3$. בסכמה כזאת, אפשר לקבל כל הקטנה של ϵ פי גורם קבוע על־ידי הגדלה מתאימה של זמן הריצה פי גורם קבוע.

תמצית הפרק

בשלושת הסעיפים הראשונים של פרק זה נציג כמה דוגמאות של אלגוריתמי קירוב בעלי זמן פולינומיאלי עבור בעיות NP־שלמות, ובסעיף האחרון נציג סכמת קירוב פולינומיאלית לחלוטין.

סעיף 37.1 מתחיל בחקירת בעיית הכיסוי על־ידי קדקודים, בעיית מזעור (כלומר, בעיית מציאת מינימום) NP־שלמה שקיים עבורה אלגוריתם קירוב בעל חסם יחס של 2. סעיף 37.2 מציג אלגוריתם קירוב בעל חסם יחס של 2 עבור המקרה של בעיית הסוכן הנוסע שבו פונקציית העלות מקיימת את אי־שוויון המשולש. שם נראה גם שללא אי־שוויון המשולש לא ייתכן קיומו של אלגוריתם קירוב בעל חסם יחס ϵ אלא אם כן $P = NP$. בסעיף 37.3 נראה כיצד ניתן להשתמש בשיטה החמדנית כאלגוריתם קירוב יעיל עבור בעיית כיסוי הקבוצה. באמצעות שיטה זו נקבל כיסוי שעלותו גדולה פי גורם לוגריתמי לכל היותר מן העלות האופטימלית. לבסוף, בסעיף 37.4, נציג סכמת קירוב פולינומיאלית לחלוטין עבור בעיית סכום התת־קבוצה.

37.1 בעיית הכיסוי על־ידי קדקודים

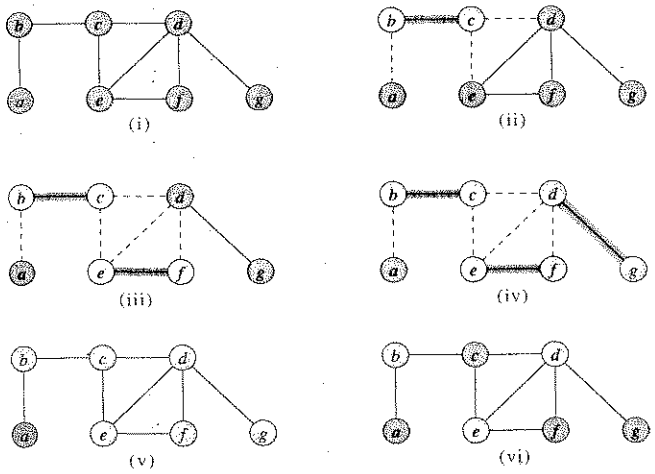
בסעיף 36.5.2 הגדרנו את בעיית הכיסוי על־ידי קדקודים והוכחנו את שלמותה ב־NP. **כיסוי על־ידי קדקודים** (vertex cover) של גרף בלתי־מכוון $G = (V, E)$ הוא תת־קבוצה $V' \subseteq V$ כך שאם (u, v) היא קשת ב־G, אזי $u \in V'$ או $v \in V'$ (או שניהם). גודלו של כיסוי על־ידי קדקודים הוא מספר הקדקודים בכיסוי.

בעיית הכיסוי על־ידי קדקודים (the vertex-cover problem) היא הבעיה של מציאת כיסוי על־ידי קדקודים שגודלו מינימלי עבור גרף בלתי־מכוון נתון. כיסוי כזה נקרא **כיסוי קדקודים אופטימלי** (optimal vertex cover). זוהי בעיה NP־קשה, שכן על־פי משפט 36.12, בעיית ההכרעה המתאימה לה היא NP־שלמה.

אף כי קשה אולי למצוא כיסוי קדקודים אופטימלי עבור גרף G, אין זה קשה מדי למצוא כיסוי כמעט אופטימלי. אלגוריתם הקירוב שלהלן מקבל כקלט גרף בלתי־מכוון G ומחזיר כיסוי על־ידי קדקודים אשר מובטח לנו כי גודלו אינו עולה על כפליים גודלו של כיסוי קדקודים אופטימלי.

```
APPROX-VERTEX-COVER(G)
1  C ← φ
2  E' ← E[G]
3  while E' ≠ φ
4      do let (u, v) be an arbitrary edge of E'
5          C ← C ∪ {u, v}
6          remove from E' every edge incident on either u or v
7  return C
```

איור 37.1 מדגים את פעולתה של APPROX-VERTEX-COVER. המשתנה C מכיל את הכיסוי ההולך ונבנה. שורה 1 מאתחלת את C לקבוצה ריקה. שורה 2 יוצרת עותק E' של קבוצת הקשתות $E[G]$ של הגרף. הלולאה בשורות 3-6 בוחרת בכל איטרציה קשת (u, v) מ־E', מוסיפה ל־C את קדקודי הקצה שלה u ו־v, ומוחקת מ־E' את כל הקשתות המכוסות על־ידי u או v. כאשר E' מיוצגת על־ידי מבנה נתונים מתאים, זמן הריצה של אלגוריתם זה הוא $O(E)$.



איור 37.1 פעולתה של APPROX-VERTEX-COVER. **(i)** גרף הקלט G מורכב מ־7 קדקודים ו־8 קשתות. **(ii)** הקשת (b, c), המודגשת באיור, היא הקשת הראשונה שבוחרת APPROX-VERTEX-COVER. הקדקודים b ו־c, הקדקודים הבהירים באיור, מצורפים לקבוצה C המכילה את כיסוי הקדקודים ההולך ונבנה. הקשתות (a, b), (c, e) ו־(c, d), הקשתות המקוקוות באיור, מוסרות, משום שכל אחת מהן מכוסה עתה על־ידי איזשהו קדקוד ב־C. **(iii)** הקשת (e, f) מצורפת ל־C. **(iv)** הקשת (d, g) מצורפת ל־C. **(v)** הקבוצה C, שהיא כיסוי הקדקודים שיצרה APPROX-VERTEX-COVER, מכילה את ששת הקדקודים b, c, d, e, f ו־g. **(vi)** כיסוי הקדקודים האופטימלי עבור גרף זה מכיל שלושה קדקודים בלבד: b, d ו־e.

משפט 37.1

האלגוריתם APPROX-VERTEX-COVER הוא בעל חסם יחס של 2.

הוכחה קבוצת הקדקודים C המוחזרת על־ידי APPROX-VERTEX-COVER היא כיסוי על־ידי קדקודים, שכן האלגוריתם חוזר ומבצע את הלולאה עד שכל קשת ב־$E[G]$ מכוסה על־ידי איזשהו קדקוד ב־C.

כדי להיווכח ש־APPROX-VERTEX-COVER מחזיר כיסוי שגודלו כפליים לכל היותר מגודלו של כיסוי אופטימלי, נסמן ב־A את קבוצת הקשתות שנבחרו בשורה 4 של APPROX-VERTEX-COVER. A אינה מכילה אף זוג קשתות שיש להן קדקודים משותפים, שכן ברגע שנבחרת קשת בשורה 4, כל הקשתות האחרות השכורות לקדקודי הקצה של קשת זו נמחקות מ־E' בשורה 6. לכן, כל ביצוע של שורה 5 מוסיף ל־C שני קדקודים חדשים, ו־$|C| = 2|A|$. אולם, כדי לכסות את הקשתות ב־A, כל כיסוי על ידי קדקודים — ובפרט כיסוי אופטימלי C^* — חייב לכלול לפחות קדקוד קצה אחד של כל קשת ב־A. מאחר ש־A אינה מכילה אף זוג קשתות שיש להן קדקודי קצה משותפים, אף קדקוד בכיסוי אינו קשור ליותר מקשת אחת ב־A. לכן, $|C^*| \geq |A|$ וגם $|C| \leq 2|C^*|$, ובכך נשלמה הוכחת המשפט. ■

תרגילים

37.1-1

מצא דוגמה של גרף שעבורו APPROX-VERTEX-COVER תמיד מחזיר פתרון תת־אופטימלי.

37.1-2

פרופסור ניקסון מציע את ההיוריסטיקה שלהלן לפתרון בעיית הכיסוי על־ידי קדקודים: בוחרים בכל פעם קדקוד מן הדרגה הגבוהה ביותר, ומסירים את כל הקשתות הקשורות אליו. מצא דוגמה המראה שההיוריסטיקה שמציע הפרופסור אינה בעלת חסם יחס של 2.

37.1-3

כתוב אלגוריתם חמדן יעיל המוצא בזמן ליניארי כיסוי קדקודים אופטימלי עבור עץ.

37.1-4

על פי ההוכחה של משפט 36.12, אנו יודעים שבעיית הכיסוי על־ידי קדקודים ובעיית הקליקה ה־NP־שלמה הן בעיות משלימות, במובן זה שכיסוי קדקודים אופטימלי הוא המשלים של קליקה מקסימלית בגרף המשלים. האם קשר זה גורר קיומו של אלגוריתם קירוב בעל חסם יחס קבוע עבור בעיית הקליקה? נמק את תשובתך.

37.2 בעיית הסוכן הנוסע

בבעיית הסוכן הנוסע שהוצגה בסעיף 36.5.5, נתון לנו גרף שלם בלתי־מכוון $G = (V, E)$ שבו לכל קשת $(u, v) \in E$ מצורפת עלות $c(u, v)$ שהיא מספר שלם אי־שלילי. עלינו למצוא ב־G מעגל המילטוני (מסלול סגור) שעלותו מינימלית. לסימונים שלנו נוסיף את הסימון $c(A)$ לציון העלות הכוללת של הקשתות בתת־קבוצה $A \subseteq E$.

$$c(A) = \sum_{(u,v) \in A} c(u, v)$$

במצבים מעשיים רבים, הדרך הישירה ממקום u למקום w היא תמיד הזולה ביותר; מעבר דרך תחנת ביניים כלשהי v אינו יכול להיות זול יותר. במילים אחרות, ביטול תחנת ביניים לעולם אינו מגדיל את העלות. בניסוח פורמלי, נאמר שפונקציית העלות c מקיימת את **אי־שוויון המשולש** (triangle inequality) אם עבור כל הקדקודים $u, v, w \in V$ מתקיים:

$$c(u, w) \leq c(u, v) + c(v, w)$$

אי־שוויון המשולש הוא תכונה טבעית המתקיימת אוטומטית ביישומים רבים. לדוגמה, אם קדקודי הגרף הם נקודות במישור ועלות הדרך בין שני קדקודים היא המרחק האויקלידי הרגיל ביניהם, אזי מתקיים אי־שוויון המשולש.

כפי שמראה תרגיל 37.2-1, הדרישה שפונקציית העלות תקיים את אי־שוויון המשולש אינה משנה את השלמות ב־NP של בעיית הסוכן הנוסע, לכן אין זה סביר שיימצא אלגוריתם זמן־פולינומיאלי לפתרון מדויק של הבעיה. לכן נחפש תחת זאת אלגוריתמי קירוב טובים עבור בעיה זו.

בסעיף 37.2.1 נבחן אלגוריתם קירוב עבור בעיית הסוכן הנוסע המקיימת את אי־שוויון המשולש, שהוא בעל חסם יחס של 2. בסעיף 37.2.2 נראה שללא אי־שוויון המשולש, לא קיים עבור הבעיה אלגוריתם קירוב בעל חסם יחס קבוע, אלא אם כן $P = NP$.

37.2.1 בעיית הסוכן הנוסע עם אי־שוויון המשולש

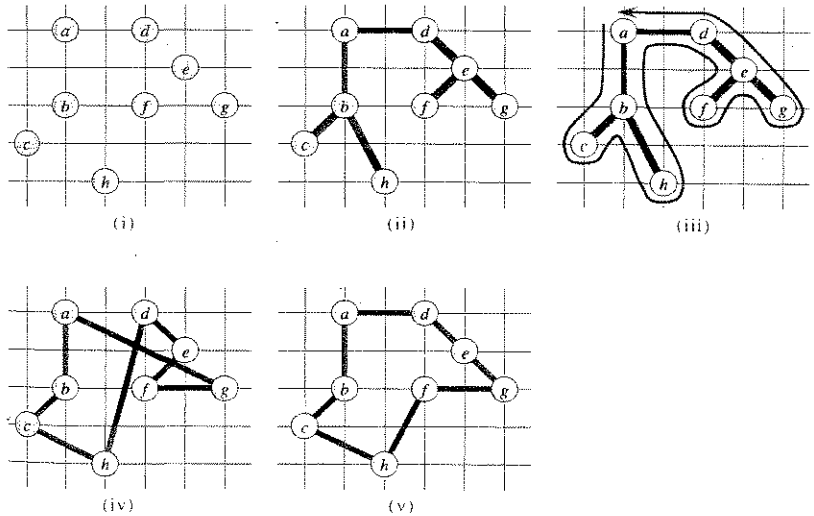
האלגוריתם שלהלן מחשב מסלול סגור כמעט אופטימלי בגרף בלתי־מכוון G תוך שימוש באלגוריתם העץ־פורש־המינימלי MST-PRIM מסעיף 24.2. אנו נראה שכאשר פונקציית העלות

מקיימת את אי־שוויון המשולש, אורכו של המסלול הסגור שמחזיר אלגוריתם זה אינו עולה על כפליים אורכו של מסלול סגור אופטימלי.

APPROX-TSP-TOUR(G, c)
1 select a vertex $r \in V[G]$ to be a "root" vertex
2 grow a minimum spanning tree T for G from root r using MST-PRIM(G, c, r)
3 let L be the list of vertices visited in a preorder tree walk of T
4 **return** the hamiltonian cycle H that visits the vertices in the order L

כזכור מסעיף 13.1, סריקה תחילית של עץ מבקרת באופן רקורסיבי בכל קדקוד בעץ, ומוסיפה לרשימה כל קדקוד בעת הביקור הראשון בו, לפני שנערך ביקור בצאצא כלשהו שלו.

איור 37.2 מדגים את פעולתה של APPROX-TSP-TOUR. בחלק (i) של האיור מופיעה קבוצת הקדקודים הנתונה, וחלק (ii) מראה את העץ הפורש המינימלי T שמצמיחה MST-PRIM החל מקדקוד השורש a. חלק (iii) מראה את סדר הביקור בקדקודי T בסריקה תחילית, וחלק (iv) מציג את המסלול הסגור המתאים, שהוא המסלול שמחזירה APPROX-TSP-TOUR. בחלק (v) מוצג מסלול סגור אופטימלי, הקצר ב־23% בערך מן המסלול המוחזר.



איור 37.2 פעולתה של APPROX-TSP-TOUR. **(i)** קבוצת הנקודות הנתונה. הנקודות מונחות על הקדקודים של סריג שלמים. לדוגמה, f נמצאת במרחק של יחידה אחת מימין ל־h, ובמרחק של שתי יחידות מעל h. המרחק האוקלידי הרגיל בין שתי נקודות משמש כפונקציית העלות. **(ii)** עץ פורש מינימלי T שמחשבת MST-PRIM עבור נקודות אלה. הקדקוד a הוא קדקוד השורש. סימוני הקדקודים נקבעו כך שהקדקודים מצורפים לעץ הראשי על־ידי MST-PRIM בסדר אלפביתי. **(iii)** סריקה של T המתחילה ב־a. סריקה מלאה של העץ מבקרת בקדקודים בסדר הבא (משמאל לימין): $a, b, c, b, h, b, a, d, e, f, e, g, e, d, a$. סריקה תחילית של T מוסיפה לרשימה כל קדקוד כאשר היא מבקרת בו לראשונה, וכך מתקבל הסדר a, b, c, h, d, e, f, g. **(iv)** מסלול סגור דרך הקדקודים המתקבל כאשר מבקרים בקדקודים בסדר הנתון על־ידי הסריקה התחילית. זהו המסלול H שמחזירה APPROX-TSP-TOUR. עלותו הכוללת היא 19.074 בערך. **(v)** מסלול אופטימלי H^* עבור קבוצת הקדקודים הנתונה. עלותו הכוללת היא 14.715 בערך.

זמן הריצה של APPROX-TSP-TOUR הוא $\Theta(E) = \Theta(V^2)$, שכן הקלט הוא גרף שלם (ראה תרגיל 24.2-2). עתה נראה שאם פונקציית העלות עבור מופע של בעיית הסוכן הנוסע מקיימת את אי־שוויון המשולש, אזי APPROX-TSP-TOUR מחזירה מסלול שעלותו כפליים עלותו של מסלול אופטימלי.

משפט 37.2

APPROX-TSP-TOUR הוא אלגוריתם קירוב בעל חסם יחס של 2 עבור בעיית הסוכן הנוסע עם אי־שוויון המשולש.

הוכחה יהי H^* מסלול אופטימלי עבור קבוצת הקדקודים הנתונה. טענה שקולה של המשפט היא כי $c(H) \le 2c(H^*)$, כאשר H הוא המסלול שמחזירה APPROX-TSP-TOUR. מכיוון שמחיקתה של קשת כלשהי ממסלול המילטוני נותנת עץ פורש, הרי שאם T הוא עץ פורש מינימלי עבור קבוצת הקדקודים הנתונה, אזי:

$$c(T) \le c(H^*) \tag{37.4}$$

סריקה מלאה (full walk) של T מוסיפה לרשימה את הקדקודים כאשר היא מבקרת בהם לראשונה וגם בכל פעם שהיא חוזרת אליהם לאחר ביקור בתת־עץ. הבה נקרא לסריקה זו W. בדוגמה שלנו, הסריקה המלאה מניבה את הסדר:

$$a, b, c, b, h, b, a, d, e, f, e, g, e, d, a$$

מכיוון שסריקה מלאה עוברת על כל קשת ב־T בדיוק פעמיים, אנו מקבלים:

$$c(W) = 2c(T) \tag{37.5}$$

משוואות (37.4) ו־(37.5) גוררות כי:

$$c(W) \le 2c(H^*) \tag{37.6}$$

כלומר, עלותה של W אינה עולה על כפליים עלותו של מסלול אופטימלי.

לרוע המזל, הסריקה W אינה מהווה בדרך כלל מסלול המילטוני, שכן היא מבקרת בחלק מהקדקודים יותר מפעם אחת. אבל, על פי אי־שוויון המשולש, ניתן למחוק מ־W ביקור בקדקוד כלשהו בלי להגדיל את העלות. (אם מוחקים מ־W קדקוד v הנמצא בין u ל־w, בסדר המתקבל יש לעבור ישירות מ־u ל־w.) על־ידי ביצוע חוזר של פעולה זו, נוכל למחוק מ־W את כל הביקורים פרט לאלה שהם ביקור ראשון בקדקוד. בדוגמה שלנו מתקבלת באופן זה רשימת הקדקודים הבאה:

$$a, b, c, h, d, e, f, g$$

רשימה זו זהה לרשימה המתקבלת מסריקה תחילית של העץ T. יהי H המעגל המתאים לסריקה תחילית זו. זהו מעגל המילטוני, שכן הוא מבקר בכל קדקוד בדיוק פעם אחת. למעשה, זהו המעגל שמחשבת APPROX-TSP-TOUR. מכיוון ש־H מתקבל על־ידי מחיקת קדקודים מן הסריקה המלאה W, אנו מקבלים:

$$c(H) \le c(W) \tag{37.7}$$

צירוף האי־שוויונים (37.6) ו־(37.7) משלים את ההוכחה. ■

למרות חסם היחס הנאה שהוכח במשפט 37.2, APPROX-TSP-TOUR איננה בדרך כלל הבחירה המעשית הטובה ביותר לפתרון בעיה זו. קיימים אלגוריתמי קירוב אחרים שביצועיהם בדרך כלל טובים בהרבה ביישומים מעשיים (ראה ההערות הביבליוגרפיות בסופו של פרק זה).

37.2.2 בעיית הסוכן הנוסע הכללית

אם מוותרים על ההנחה שפונקציית העלות c מקיימת את אי־שוויון המשולש, לא ניתן למצוא בזמן פולינומיאלי מסלולים מקורבים טובים, אלא אם כן $P = NP$.

משפט 37.3
אם $P \neq NP$ ו־$\rho \geq 1$, אזי לא קיים עבור בעיית הסוכן הנוסע הכללית אלגוריתם קירוב בעל זמן ריצה פולינומיאלי וחסם יחס ρ.

הוכחה ההוכחה היא בדרך השלילה. נניח בשלילה שעבור איזשהו מספר $\rho \geq 1$ קיים אלגוריתם קירוב A שרץ בזמן פולינומיאלי והוא בעל חסם יחס ρ. בלי הגבלת הכלליות, נניח כי ρ הוא מספר שלם (אחרת, נעגל אותו). עתה נראה כיצד ניתן להשתמש ב־A כדי לפתור בזמן פולינומיאלי מופעים של בעיית המעגל ההמילטוני (שהוגדרה בסעיף 36.5.5). על פי משפט 36.14, בעיית המעגל ההמילטוני היא NP־שלמה, ולכן, על פי משפט 36.4, פתרונה בזמן פולינומיאלי נותר כי $P = NP$.

יהי $G = (V, E)$ מופע של בעיית המעגל ההמילטוני. ברצוננו לקבוע באופן יעיל אם G מכיל מעגל המילטוני על־ידי שימוש באלגוריתם הקירוב המשוער A. נהפוך את G למופע של בעיית הסוכן הנוסע באופן הבא: יהי $G' = (V, E')$ הגרף השלם עבור קבוצת הקדקודים V, כלומר:

$$E' = \{(u, v) : u, v \in V \text{ וגם } u \neq v\}$$

לכל קשת ב־E' נתאים עלות שהיא מספר שלם, באופן הבא:

$$c(u, v) = \begin{cases} 1 & \text{אם } (u, v) \in E \\ \rho|V| + 1 & \text{אחרת} \end{cases}$$

בהינתן ייצוג של G, ניתן ליצור ייצוגים של G' ושל c בזמן פולינומיאלי ב־$|V|$ וב־$|E|$.

כעת, נתבונן בבעיית הסוכן הנוסע (G', c). אם הגרף המקורי G מכיל מעגל המילטוני H, אזי פונקציית העלות c מתאימה לכל קשת ב־H עלות של 1, ולכן (G', c) מכיל מסלול המילטוני שעלותו $|V|$. מצד שני, אם G אינו מכיל מעגל המילטוני, אזי כל מסלול המילטוני ב־G' כולל בהכרח איזושהי קשת שאינה שייכת ל־E. אבל כל מסלול הכולל קשת שאינה שייכת ל־E עלותו היא לפחות:

$$(\rho|V| + 1) + (|V| - 1) > \rho|V|$$

מכיוון שעלותן של קשתות שאינן שייכות ל־G גבוהה כל כך, קיים פער גדול בין עלותו של מסלול המהווה מעגל המילטוני ב־G (עלות של $|V|$) לבין עלותו של כל מסלול אחר (עלות הגדולה מ־$\rho|V|$).

מה קורה אם מריצים את אלגוריתם הקירוב A על המופע (G', c) של בעיית הסוכן הנוסע? מאחר שמובטח לנו כי A יחזיר מסלול שעלותו אינה עולה על ρ פעמים עלותו של מסלול אופטימלי, הרי שאם G מכיל מעגל המילטוני, אזי A מחזיר בהכרח מעגל זה. אם G אינו מכיל מעגל המילטוני, אזי A מחזיר מסלול שעלותו גדולה מ־$\rho|V|$. לכן, ניתן להשתמש ב־A כדי לפתור את בעיית המעגל ההמילטוני בזמן פולינומיאלי. ■

תרגילים

37.2-1
הראה כיצד ניתן להמיר בזמן פולינומיאלי מופע אחד של בעיית הסוכן הנוסע למופע אחר שפונקציית העלות שלו מקיימת את אי־שוויון המשולש. לשני המופעים חייבת להיות אותה קבוצה של מסלולים אופטימליים. הסבר מדוע המרה כזאת בזמן פולינומיאלי אינה סותרת את משפט 37.3, בכפוף להנחה כי $P \neq NP$.

37.2-2
נתבונן ב**היוריסטיקת הנקודה הקרובה ביותר** (the closest-point heuristic) לבנייתו של מסלול מקורב עבור בעיית הסוכן הנוסע. מתחילים עם מעגל טריוויאלי המורכב מקדקוד יחיד שנבחר באופן שרירותי. בכל צעד, מוצאים את הקדקוד u שאינו שייך למעגל אבל מרחקו מקדקוד כלשהו על המעגל הוא מינימלי. נניח שמבין קדקודי המעגל, הקדקוד הקרוב ביותר ל־u הוא v. מרחיבים את המעגל כך שיכלול גם את u על־ידי הכנסת u מיד אחרי v. חזרו על צעדים אלה עד שכל הקדקודים נכללים במעגל. הוכח שהיוריסטיקה זו מחזירה מסלול שעלותו הכוללת אינה עולה על כפליים עלותו של מסלול אופטימלי.

37.2-3
בבעיית הסוכן הנוסע עם צוואר־בקבוק (the bottleneck traveling salesman problem), ברצוננו למצוא את המעגל ההמילטוני שבו אורכה של הקשת הארוכה ביותר במעגל הוא מינימלי. בהנחה שפונקציית העלות מקיימת את אי־שוויון המשולש, הראה שקיים עבור בעיה זו אלגוריתם קירוב שרץ בזמן פולינומיאלי והוא בעל חסם יחס של 3. (**רמז:** הראה באופן רקורסיבי שניתן לבקר בכל הצמתים של עץ צוואר־בקבוק פורש בדיוק פעם אחת על־ידי ביצוע סריקה מלאה של העץ תוך דילוג על צמתים, אבל בלי לדלג על יותר משני צומתי ביניים עוקבים.)

37.2-4
נניח שהקדקודים במופע של בעיית הסוכן הנוסע הם נקודות במישור ושהעלות $c(u, v)$ היא המרחק האוקלידי בין הנקודות u ו־v. הראה שמסלול אופטימלי לעולם אינו חוצה את עצמו.

37.3 בעיית כיסוי הקבוצה

בעיית כיסוי הקבוצה היא בעיית אופטימיזציה המייצגת בעיות רבות של בחירת משאבים. זוהי הכללה של הבעיה ה־NP שלמה של כיסוי על־ידי קדקודים, ולכן גם היא NP־קשה. אולם אלגוריתם הקירוב שפותח עבור בעיית הכיסוי על־ידי קדקודים אינו מתאים כאן, ולכן עלינו לנסות גישות אחרות. אנו נבחן היוריסטיקה חמדנית פשוטה בעלת חסם יחס לוגריתמי. כלומר, ככל שגודלו של המופע גדל, גודלו של הפתרון המקורב עשוי לגדול, יחסית לגודלו של פתרון אופטימלי. אבל, מאחר שפונקציית הלוגריתם גדלה בצורה איטית למדי, אלגוריתם קירוב זה עשוי בכל זאת להניב תוצאות מועילות.

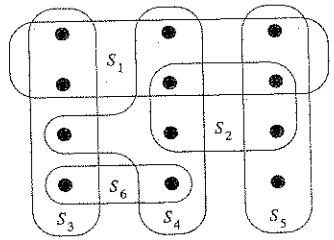
מופע $(X, \mathcal{F})$ של **בעיית כיסוי הקבוצה** (the set-covering problem) מורכב מקבוצה סופית X וממשפחה $\mathcal{F}$ של תת־קבוצות של X, כך שכל איבר ב־X שייך לפחות לתת־קבוצה אחת ב־$\mathcal{F}$:

$$X = \bigcup_{S \in \mathcal{F}} S$$

אנו אומרים שתת־קבוצה $S \in \mathcal{F}$ **מכסה** (covers) את איבריה. הבעיה היא למצוא תת־קבוצה $\mathcal{C} \subseteq \mathcal{F}$ שגודלה מינימלי אשר איבריה מכסים את X כולה:

$$X = \bigcup_{S \in \mathcal{C}} S \tag{37.8}$$

על כל $\mathcal{C}$ המקיימת את משוואה (37.8) אנו אומרים שהיא **מכסה** (covers) את X. איור 37.3 מדגים את הבעיה.



איור 37.3 מופע $(X, \mathcal{F})$ של בעיית כיסוי הקבוצה, כאשר X מורכבת מ־12 הנקודות השחורות ו־$\mathcal{F} = \{S_1, S_2, S_3, S_4, S_5, S_6\}$. $\mathcal{C} = \{S_3, S_4, S_5\}$ הוא כיסוי של הקבוצה שגודלו מינימלי. האלגוריתם החמדן מחזיר כיסוי בגודל 4 על־ידי בחירת הקבוצות S_1, S_4, S_5 ו־S_3, בסדר זה.

בעיית כיסוי הקבוצה היא הפשטה של בעיות קומבינטוריות נפוצות רבות. כדוגמה פשוטה, נניח כי X מייצגת קבוצה של מיומנויות הנחוצות כדי לפתור בעיה וכי נתונה לנו קבוצת בני אדם הזמינים לעבוד על הבעיה. ברצוננו למנות ועדה, הכוללת מספר קטן ככל האפשר של חברים, כך שעבור כל מיומנות נדרשת ב־X, יהיה בוועדה חבר בעל מיומנות זו. בגרסת ההכרעה של בעיית כיסוי הקבוצה, אנו שואלים אם קיים או לא קיים כיסוי שגודלו לכל היותר k, כאשר k הוא פרמטר נוסף הנתון במופע של הבעיה. כפי שיתבקש הקורא להראות בתרגיל 37.3-2, גרסת ההכרעה של בעיה זו היא NP־שלמה.

אלגוריתם קירוב חמדן

השיטה החמדנית פועלת כך: בכל שלב בוחרים את הקבוצה S המכסה את המספר הגדול ביותר של איברים שטרם כוסו.

```
GREEDY-SET-COVER(X, F)
1  U ← X
2  C ← φ
3  while U ≠ φ
4      do select an S ∈ F that maximizes |S ∩ U|
5         U ← U − S
6         C ← C ∪ {S}
7  return C
```

בדוגמה שבאיור 37.3, GREEDY-SET-COVER מצרפת ל־$\mathcal{C}$ הקבוצות S_1, S_4, S_5 ו־S_3, בסדר זה.

האלגוריתם פועל כדלקמן. הקבוצה U מכילה, בכל שלב, את קבוצת האיברים הנותרים שטרם כוסו. הקבוצה $\mathcal{C}$ מכילה את הכיסוי ההולך ונבנה. בשורה 4 מתבצעת הבחירה החמדנית. בשורה

זו נבחרת תת־קבוצה S המכסה מספר גדול ככל האפשר של איברים בלתי מכוסים (במקרה של שוויון, הבחירה שרירותית). לאחר ש־S נבחרה היא מצורפת ל־$\mathcal{C}$ ואיבריה נמחקים מ־U. כאשר האלגוריתם עוצר, הקבוצה $\mathcal{C}$ מכילה תת־משפחה של $\mathcal{F}$ המכסה את X.

את האלגוריתם GREEDY-SET-COVER ניתן לממש בנקל כך שירוץ בזמן פולינומיאלי ב־$|X|$ וב־$|\mathcal{F}|$. מכיוון שמספר האיטרציות שמבצעת הלולאה שבשורות 3-6 הוא לכל היותר $\min(|X|, |\mathcal{F}|)$, ואת גוף הלולאה ניתן לממש כך שירוץ בזמן $O(|X||\mathcal{F}|)$, קיים מימוש שרץ בזמן $O(|X||\mathcal{F}| \min(|X|, |\mathcal{F}|))$. בתרגיל 37.3-3 יתבקש הקורא לכתוב אלגוריתם לינארי לבעיה זו.

ניתוח

עתה נראה שהאלגוריתם החמדן מחזיר כיסוי קבוצה שאינו גדול בהרבה מכיסוי קבוצה אופטימלי. מטעמי נוחות, נסמן בפרק זה ב־$H(d)$ את המספר ההרמוני ה־d. כלומר, נגדיר $H(d) = H_d = \sum_{i=1}^{d} 1/i$. (ראה סעיף 3.1.)

משפט 37.4
האלגוריתם GREEDY-SET-COVER הוא בעל חסם יחס של:

$$H(\max\{|S| : S \in \mathcal{F}\})$$

הוכחה ההוכחה נעשית על־ידי קביעת עלות לכל קבוצה הנבחרת על־ידי האלגוריתם, חלוקת עלות זו בין כל האיברים המכוסים בפעם הראשונה, ואז שימוש בעלויות אלה לקבלת הקשר המבוקש בין גודלו של כיסוי קבוצה אופטימלי $\mathcal{C}^*$ לבין גודלו של כיסוי הקבוצה $\mathcal{C}$ שמחזיר האלגוריתם. נסמן ב־S_i את תת־קבוצה ה־i הנבחרת על־ידי GREEDY-SET-COVER; האלגוריתם מחייב את עצמו בעלות של 1 כאשר הוא מוסיף את S_i ל־$\mathcal{C}$. את העלות הזאת של בחירת S_i אנו מחלקים באופן שווה בין האיברים המכוסים לראשונה על־ידי S_i. נסמן ב־c_x את העלות המיוחסת לאיבר x, עבור כל $x \in X$. לכל איבר מיוחסת עלות פעם אחת בלבד, כאשר הוא מכוסה בפעם הראשונה. אם x מכוסה לראשונה על־ידי S_i, אזי:

$$c_x = \frac{1}{|S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|}$$

האלגוריתם מוצא פתרון $\mathcal{C}$ שעלותו הכוללת $|\mathcal{C}|$, ועלות זו התחלקה בין כל איברי X. לכן, מאחר שהכיסוי האופטימלי $\mathcal{C}^*$ מכסה גם הוא את X, אנו מקבלים:

$$\begin{aligned} |\mathcal{C}| &= \sum_{x \in X} c_x \\ &\le \sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x \end{aligned} \tag{37.9}$$

שארית ההוכחה מתבססת על האי־שוויון העיקרי הבא, שאותו נוכיח בהמשך. לכל קבוצה S והשייכת למשפחה $\mathcal{F}$, מתקיים:

$$\sum_{x \in S} c_x \le H(|S|) \tag{37.10}$$

מהאי־שוויונים (37.9) ו־(37.10) נובע כי:

$$
\begin{aligned}
|C| &\le \sum_{S \in C^*} H(|S|) \\
&\le |C^*| \cdot H(\max\{|S| : S \in \mathcal{F}\})
\end{aligned}
$$

ובכך הוכח המשפט. לא נותר אפוא אלא להוכיח את אי־שוויון (37.10). עבור כל קבוצה $S \in \mathcal{F}$ וכל $i = 1, 2, \ldots, |C|$, יהי:

$$u_i = |S - (S_1 \cup S_2 \cup \cdots \cup S_i)|$$

מספר איברי S שנותרו בלתי מכוסים לאחר שהאלגוריתם בחר את $S_1, S_2, \ldots, S_i$. נגדיר את $u_0 = |S|$ כמספר האיברים ב־S, שכן במצב ההתחלתי כל האיברים אינם מכוסים. יהי k האינדקס הקטן ביותר כך ש־$u_k = 0$, כלומר, כל האיברים ב־S מכוסים לפחות על־ידי אחת מן הקבוצות $S_1, S_2, \ldots, S_k$. אזי $u_i \ge u_{i-1}$, ו־$u_{i-1} - u_i$ מאיברי S מכוסים לראשונה על־ידי S_i עבור $i = 1, 2, \ldots, k$. לכן:

$$\sum_{x \in S} c_x = \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot \frac{1}{|S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|}$$

נשים לב כי:

$$
\begin{aligned}
|S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})| &\ge |S - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})| \\
&= u_{i-1}
\end{aligned}
$$

מכיוון שהבחירה החמדנית של S_i מבטיחה ש־S אינה יכולה לכסות יותר איברים חדשים מאשר S_i (אחרת, היתה נבחרת S במקום S_i). מכאן אנו מקבלים:

$$\sum_{x \in S} c_x \le \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot \frac{1}{u_{i-1}}$$

עבור שלמים a ו־b כך ש־$a < b$, מתקיים:

$$
\begin{aligned}
H(b) - H(a) &= \sum_{i=a+1}^{b} \frac{1}{i} \\
&\ge (b-a)\frac{1}{b}
\end{aligned}
$$

תוך שימוש באי־שוויון זה אנו מקבלים את הסכום הטלסקופי:

$$
\begin{aligned}
\sum_{x \in S} c_x &\le \sum_{i=1}^{k} (H(u_{i-1}) - H(u_i)) \\
&= H(u_0) - H(u_k) \\
&= H(u_0) - H(0) \\
&= H(u_0) \\
&= H(|S|)
\end{aligned}
$$

שכן $H(0) = 0$. בכך נשלמה הוכחת אי־שוויון (37.10). ■

מסקנה 37.5

GREEDY-SET-COVER הוא בעל חסם יחס של $(\ln|X| + 1)$.

הוכחה באמצעות אי־שוויון (3.12) ומשפט 37.4. ■

קיימים יישומים שבהם הערך $\max\{|S| : S \in \mathcal{F}\}$ הוא קבוע קטן, ולכן הפתרון שמחזירה GREEDY-SET-COVER גדול לכל היותר פי גורם קבוע מפתרון אופטימלי. זהו, למשל, המצב כאשר משתמשים בהיוריסטיקה זו לבניית כיסוי קדקודים מקורב עבור גרף שקדקודיו בעלי דרגה שאינה עולה על 3. במקרה זה, גודלו של הפתרון שמוצאת GREEDY-SET-COVER אינו עולה על $H(3) = 11/6$ פעמים גודלו של פתרון אופטימלי, ופירוש הדבר שביצועיה טובים במקצת מאלה של APPROX-VERTEX-COVER.

תרגילים

37.3-1

התייחס לכל אחת מהמילים הבאות כאל קבוצת אותיות:

{arid, dash, drain, heard, lost, nose, shun, slate, snare, thread}

הראה איזה כיסוי קבוצה מחזירה GREEDY-SET-COVER, כאשר במקרה של שוויון, נבחרת המילה המופיעה ראשונה במילון.

37.3-2

הראה שגרסת ההכרעה של בעיית כיסוי הקבוצה היא NP־שלמה, באמצעות רדוקציה מבעיית הכיסוי על־ידי קדקודים.

37.3-3

הראה כיצד ניתן לממש את GREEDY-SET-COVER כך שתרוץ בזמן $O\left(\sum_{S \in \mathcal{F}} |S|\right)$.

37.3-4

הראה שהצורה החלשה הבאה של משפט 37.4 נכונה באופן טריוויאלי:

$$|C| \le |C^*| \max\{|S| : S \in \mathcal{F}\}$$

37.3-5

צור משפחה של מופעי בעיית כיסוי הקבוצה המוכיחה כי מספר הפתרונות השונים שמחזירה GREEDY-SET-COVER יכול להיות אקספוננציאלי בגודל המופע. (פתרונות שונים נוצרים על־ידי כך שבמקרי שוויון, S נבחרת בכל פעם בצורה שונה בשורה 4.)

37.4 בעיית סכום התת־קבוצה

מקרה של בעיית סכום התת־קבוצה הוא זוג (S, t), כאשר S היא קבוצה $\{x_1, x_2, \ldots, x_n\}$ של שלמים חיוביים ו־t הוא שלם חיובי. בעיית הכרעה זו שואלת אם קיימת תת־קבוצה של S אשר סכום איבריה שווה בדיוק לערך המטרה t. בעיה זו היא NP־שלמה (ראה סעיף 36.5.3).

בעיית האופטימיזציה הנלווית לבעיית הכרעה זו מתעוררת ביישומים מעשיים. בבעיית האופטימיזציה, ברצוננו למצוא תת־קבוצה של $\{x_1, x_2, \ldots, x_n\}$ אשר סכום איבריה גדול ככל האפשר אולם אינו גדול מ־t. לדוגמה, ייתכן שיש לנו משאית שיכולה לשאת משא שמשקלו t לכל

היותר, ועלינו לשלוח n קופסאות שונות, כאשר הקופסה ה־i משקלה x_i ק"ג. ברצוננו למלא את המשאית ככל האפשר, בלי לחרוג מהמשקל המקסימלי המותר.

בסעיף זה נציג אלגוריתם זמן־אקספוננציאלי לפתרון בעיה זו ונראה כיצד ניתן לשנות את האלגוריתם כך שנקבל סכמת קירוב פולינומיאלית לחלוטין. (כזכור, סכמת קירוב פולינומיאלית לחלוטין רצה בזמן שהוא פולינומיאלי גם ב־$1/\epsilon$ ולא רק ב־n.)

אלגוריתם זמן־אקספוננציאלי

אם L היא רשימה של שלמים חיוביים ו־x הוא שלם חיובי נוסף, אזי אנו מסמנים ב־$L + x$ את רשימת השלמים המתקבלת מ־L על־ידי הוספת x לכל אחד מאיברי L. לדוגמה, אם $L = \langle 1, 2, 3, 5, 9 \rangle$, אזי $L + 2 = \langle 3, 4, 5, 7, 11 \rangle$. סימון זה משמש אותנו גם עבור קבוצות:

$$S + x = \{s + x : s \in S\}$$

נשתמש בשגרת עזר MERGE-LISTS(L, L') המחזירה את הרשימה הממוינת המתקבלת ממיזוג של שתי רשימות הקלט הממוינות L ו־L'. כמו השגרה MERGE שהשתמשנו בה במיון־מיזוג (סעיף 1.3.1), השגרה MERGE-LISTS רצה בזמן $O(|L| + |L'|)$. (לא נביא כאן את הפסאודו־קוד עבור MERGE-LISTS.) השגרה EXACT-SUBSET-SUM מקבלת כקלט קבוצה $S = \{x_1, x_2, \ldots, x_n\}$ וערך מטרה t.

```
EXACT-SUBSET-SUM(S, t)
1  n ← |S|
2  L_0 ← ⟨0⟩
3  for i ← 1 to n
4      do L_i ← MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
5         remove from L_i every element that is greater than t
6  return the largest element in L_n
```

נסמן ב־P_i את קבוצת כל הערכים שיכולים להתקבל מבחירת תת־קבוצה (אולי ריקה) של $\{x_1, x_2, \ldots, x_i\}$ וחישוב סכום איבריה. לדוגמה, אם $S = \{1, 4, 5\}$, אזי:

$$P_1 = \{0, 1\}$$
$$P_2 = \{0, 1, 4, 5\}$$
$$P_3 = \{0, 1, 4, 5, 6, 9, 10\}$$

בהינתן הזהות:

$$P_i = P_{i-1} \cup (P_{i-1} + x_i) \tag{37.11}$$

ניתן להוכיח באינדוקציה על i (ראה תרגיל 37.4-1) שהרשימה L_i היא רשימה ממוינת המכילה כל איבר של P_i שערכו אינו עולה על t. מכיוון שאורכה של L_i יכול להגיע עד 2^i, האלגוריתם EXACT-SUBSET-SUM הוא, בכלל, אלגוריתם זמן־אקספוננציאלי, אף כי הוא אלגוריתם זמן־פולינומיאלי במקרים המיוחדים שבהם t הוא פולינום ב־$|S|$ או שכל המספרים ב־S חסומים על־ידי פולינום ב־$|S|$.

סכמת קירוב פולינומיאלית לחלוטין

ניתן לקבל סכמת קירוב פולינומיאלית לחלוטין עבור בעיית סכום התת־קבוצה על־ידי "גיזום" של כל רשימה L_i לאחר שנוצרה. נשתמש בפרמטר גיזום δ המקיים $0 < \delta < 1$. **לגזום** (trim) רשימה

L ב־δ פירושו להסיר מ־L איברים רבים ככל האפשר, באופן כזה שאם L' היא הרשימה המתקבלת מגיזום L, אזי עבור כל איבר y שהוסר מ־L, עדיין קיים ב־L' איבר z שעבורו:

$$\frac{y - z}{y} \leq \delta$$

או, באופן שקול:

$$(1 - \delta)y \leq z \leq y$$

ניתן לחשוב על z כזה כאילו הוא "מייצג" את y ברשימה החדשה L'. כל y מיוצג על־ידי z כך שהשגיאה היחסית של z ביחס ל־y היא לכל היותר δ. לדוגמה, אם $\delta = 0.1$ והרשימה היא:

$$L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle$$

נוכל לגזום את L ולקבל את הרשימה:

$$L' = \langle 10, 12, 15, 20, 23, 29 \rangle$$

כאשר הערך 11 שנמחק מיוצג על־ידי 10, הערכים 21 ו־22 שנמחקו מיוצגים על־ידי 20, והערך 24 שנמחק מיוצג על־ידי 23. חשוב לזכור שכל איבר בגרסה הגזומה של הרשימה הוא גם איבר בגרסה המקורית של הרשימה. גיזום רשימה יכול להקטין באופן דרמטי את מספר האיברים ברשימה ועם זאת לשמור עבור כל איבר שנמחק מהרשימה ערך מייצג קרוב (וקטן במקצת).

השגרה שלהלן גוזמת רשימת קלט $L = \langle y_1, y_2, \ldots, y_m \rangle$ בזמן $\Theta(m)$, בהנחה ש־L ממוינת בסדר לא יורד. השגרה מחזירה רשימה גזומה ממוינת.

```
TRIM(L, δ)
1  m ← |L|
2  L' ← ⟨y_1⟩
3  last ← y_1
4  for i ← 2 to m
5      do if last < (1 − δ)y_i
6           then append y_i onto the end of L'
7                last ← y_i
8  return L'
```

איברי L נסרקים בסדר עולה, ומספר מצורף לרשימה המוחזרת L' רק אם זהו האיבר הראשון ב־L או אם אינו יכול להיות מיוצג על־ידי המספר האחרון שצורף ל־L'.

בהינתן השגרה TRIM, נוכל לבנות את סכמת הקירוב שלנו באופן הבא. השגרה תקבל כקלט קבוצה $S = \{x_1, x_2, \ldots, x_n\}$ של n שלמים (בסדר שרירותי), ערך מטרה t, ו"פרמטר קירוב" ϵ המקיים $0 < \epsilon < 1$.

```
APPROX-SUBSET-SUM(S, t, ε)
1  n ← |S|
2  L_0 ← ⟨0⟩
3  for i ← 1 to n
4      do L_i ← MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
5         L_i ← TRIM(L_i, ε/n)
6         remove from L_i every element that is greater than t
7  let z be the largest value in L_n
8  return z
```

שורה 2 מאתחלת את הרשימה L_0 כך שתכיל רק את האיבר 0. בלולאה שבשורות 3-6 מחושבת L_i כרשימה ממוינת המכילה גרסה גזומה מתאימה של הקבוצה P_i, שהוסרו ממנה כל האיברים הגדולים מ־t. מאחר ש־L_i נוצרת מ־L_{i-1}, עלינו להבטיח שהגיזום החוזר לא יגרום לחוסר דיוק רב מדי. מיד נראה כי APPROX-SUBSET-SUM מחזירה קירוב נכון אם קיים קירוב כזה.

לדוגמה, נניח שנתון לנו המופע:

$$L = \langle 104, 102, 201, 101 \rangle$$

עם $t = 308$ ו־$\epsilon = 0.20$. פרמטר הגיזום δ הוא $\epsilon/4 = 0.05$. APPROX-SUBSET-SUM מחשבת את הערכים הבאים בשורות המתאימות:

שורה 2:	$L_0 = \langle 0 \rangle$
שורה 4:	$L_1 = \langle 0, 104 \rangle$
שורה 5:	$L_1 = \langle 0, 104 \rangle$
שורה 6:	$L_1 = \langle 0, 104 \rangle$
שורה 4:	$L_2 = \langle 0, 102, 104, 206 \rangle$
שורה 5:	$L_2 = \langle 0, 102, 206 \rangle$
שורה 6:	$L_2 = \langle 0, 102, 206 \rangle$,
שורה 4:	$L_3 = \langle 0, 102, 201, 206, 303, 407 \rangle$
שורה 5:	$L_3 = \langle 0, 102, 201, 303, 407 \rangle$
שורה 6:	$L_3 = \langle 0, 102, 201, 303 \rangle$
שורה 4:	$L_4 = \langle 0, 101, 102, 201\ 203, 302, 303, 404 \rangle$
שורה 5:	$L_4 = \langle 0, 101, 201, 302, 404 \rangle$
שורה 6:	$L_4 = \langle 0, 101, 201, 302 \rangle$

האלגוריתם מחזיר כתשובה $z = 302$, תשובה שסטייתה מן התשובה האופטימלית $101 + 102 + 104 = 307$ בהחלט אינה עולה על $\epsilon = 20\%$, ולמעשה היא קטנה מ־2%.

משפט 37.6

APPROX-SUBSET-SUM היא סכמת קירוב פולינומיאלית לחלוטין עבור בעיית סכום התת־קבוצה.

הוכחה הפעולות של גיזום L_i בשורה 5 והסרת כל האיברים הגדולים מ־t מן הרשימה L_i משמרות את התכונה שכל איבר של L_i שייך גם ל־P_i. לכן, הערך z המוחזר בשורה 8 הוא אכן סכום של איזושהי תת־קבוצה של S. נותר להראות שערך זה אינו קטן מ־$(1 - \epsilon)$ פעמים ערכו של פתרון אופטימלי. (נשים לב כי מאחר שבעיית סכום התת־קבוצה היא בעיית מקסימום, אי־שוויון (37.2) שקול לאי־שוויון $C^*(1 - \epsilon) \le C$.) עלינו גם להראות שהאלגוריתם רץ בזמן פולינומיאלי.

כדי להראות שהשגיאה היחסית של התשובה המוחזרת היא קטנה, נשים לב כי כאשר גוזמים רשימה L_i, מכניסים שגיאה יחסית שאינה עולה על ϵ/n בין הערכים המייצגים הנותרים ברשימה לבין הערכים לפני הגיזום. באינדוקציה על i ניתן להראות שעבור כל איבר y השייך ל־P_i וערכו אינו גדול מ־t, קיים $z \in L_i$ המקיים:

$$(1 - \epsilon/n)^i y \le z \le y \tag{37.12}$$

אם $y^* \in P_n$ הוא פתרון אופטימלי לבעיית סכום התת־קבוצה, אזי קיים $z \in L_n$ המקיים:

$$(1 - \epsilon/n)^n y^* \le z \le y^* \tag{37.13}$$

ה־z הגדול ביותר מבין אלה הוא הערך שמחזירה APPROX-SUBSET-SUM. מאחר שניתן להראות כי:

$$\frac{d}{dn}\left(1 - \frac{\epsilon}{n}\right)^n > 0$$

הרי שהפונקציה $(1 - \epsilon/n)^n$ גדלה עם n, ולכן $n > 1$ גורר:

$$1 - \epsilon < (1 - \epsilon/n)^n$$

ומכאן:

$$(1 - \epsilon)y^* \le z$$

לכן, הערך z שמחזירה APPROX-SUBSET-SUM אינו קטן מ־$(1 - \epsilon)$ פעמים ערכו של הפתרון האופטימלי y^*.

כדי להראות שזוהי סכמת קירוב פולינומיאלית לחלוטין, נמצא חסם על אורכה של L_i. לאחר גיזום, איברים עוקבים z ו־z' ב־L_i מקיימים בהכרח את הקשר $z/z' > 1/(1 - \epsilon/n)$. דהיינו, הם בהכרח שונים זה מזה בגורם של $1/(1 - \epsilon/n)$ לפחות. לכן, מספר האיברים בכל L_i הוא לכל היותר:

$$\log_{1/(1-\epsilon/n)} t = \frac{\ln t}{-\ln(1 - \epsilon/n)} \le \frac{n \ln t}{\epsilon}$$

תוך שימוש במשוואה (2.10). חסם זה פולינומיאלי ב־n, שהוא מספר הקלטים הנתונים, וב־$\lg t$, שהוא מספר הסיביות הדרושות לייצוג t, וב־$1/\epsilon$. מאחר שזמן הריצה של APPROX-SUBSET-SUM הוא פולינומיאלי באורכן של הרשימות L_i, השגרה APPROX-SUBSET-SUM היא סכמת קירוב פולינומיאלית לחלוטין. ■

תרגילים

37.4-1

הוכח את משוואה (37.11).

37.4-2

הוכח את משוואות (37.12) ו־(37.13).

37.4-3

כיצד היית משנה את סכמת הקירוב שהוצגה בסעיף זה כדי למצוא קירוב טוב לערך הקטן ביותר שאינו קטן מ־t והוא סכום האיברים של איזושהי תת־קבוצה של רשימת הקלט הנתונה?

בעיות

37-1 *אריזה בתיבות*

נניח שנתונה לנו קבוצה של n עצמים, כאשר הגודל s_i של העצם ה-i מקיים $0 < s_i < 1$. ברצוננו לארוז את כל העצמים בתוך המספר המינימלי של תיבות בגודל יחידה. כל תיבה יכולה להכיל כל תת-קבוצה של עצמים שגודלם הכולל אינו עולה על 1.

א. הוכח שהבעיה של קביעת מספר התיבות המינימלי הדרוש היא NP-קשה. (**רמז:** עשה רדוקציה מבעיית סכום התת-קבוצה.)

היוריסטיקת **המתאימה-הראשונה** (first-fit) בוחנת כל עצם בתורו ומכניסה אותו לתיבה הראשונה שיש בה די מקום עבורו. נסמן $S = \sum_{i=1}^{n} s_i$.

ב. הראה כי המספר האופטימלי של תיבות שדרוש הוא לפחות $\lceil S \rceil$.

ג. הראה כי היוריסטיקת המתאימה-הראשונה מותירה לכל היותר תיבה אחת מלאה עד פחות מחציה.

ד. הוכח שכאשר משתמשים בהיוריסטיקת המתאימה-הראשונה, מספר התיבות הדרושות לעולם אינו עולה על $\lceil 2S \rceil$.

ה. הוכח שהיוריסטיקת המתאימה-הראשונה היא בעלת חסם היחס 2.

ו. מצא מימוש יעיל של היוריסטיקת המתאימה-הראשונה, ונתח את זמן הריצה שלו.

37-2 *מציאת קירוב לגודלה של קליקה מקסימלית*

יהי $G = (V, E)$ גרף בלתי-מכוון. עבור כל $k \geq 1$, נגדיר את $G^{(k)}$ כגרף הבלתי-מכוון $(V^{(k)}, E^{(k)})$, כאשר $V^{(k)}$ היא קבוצת כל ה-k-יות הסדורות של קדקודים מ-V ו-$E^{(k)}$ מוגדרת כך ש-$(v_1, v_2, \ldots, v_k)$ סמוכה ל-$(w_1, w_2, \ldots, w_k)$ אם ורק אם עבור כל i, $1 \leq i \leq k$, הקדקוד v_i סמוך לקדקוד w_i ב-G או $v_i = w_i$.

א. הוכח שגודלה של קליקה מקסימלית ב-$G^{(k)}$ שווה לחזקה ה-k של גודלה של קליקה מקסימלית ב-G.

ב. הראה כי אם קיים אלגוריתם קירוב בעל חסם יחס קבוע למציאת קליקה בגודל מקסימלי, אזי קיימת עבור הבעיה סכמת קירוב פולינומיאלית לחלוטין.

37-3 *הגרסה המשוקללת של בעיית כיסוי הקבוצה*

נניח שנכליל את בעיית כיסוי הקבוצה כך שנצרף משקל w_i לכל קבוצה S_i השייכת למשפחה $\mathcal{F}$, ומשקלו של כיסוי $\mathcal{C}$ יהיה $\sum_{S_i \in \mathcal{C}} w_i$. ברצוננו למצוא כיסוי בעל משקל מינימלי. (סעיף 37.3 עוסק במקרה שבו $w_i = 1$ עבור כל i.)

הראה שניתן להכליל באופן טבעי את ההיוריסטיקה החמדנית לכיסוי קבוצה כך שתניב פתרון מקורב לכל מופע של הגרסה המשוקללת של בעיית כיסוי הקבוצה. הראה שההיוריסטיקה שלך היא בעלת חסם יחס $H(d)$, כאשר d הוא גודלה המקסימלי של קבוצה כלשהי S_i.

הערות ביבליוגרפיות לפרק 37

קיימת ספרות ענפה בנושא של אלגוריתמי קירוב. מקום טוב להתחיל בו הוא Garey & Johnson [79]. גם אצל Papadimitriou & Steiglitz [154] מופיעה הצגה מצוינת של אלגוריתמי קירוב. Lawler, Lenstra, Rinnooy Kan & Shmoys [133] מטפלים באופן יסודי ומקיף בבעיית הסוכן הנוסע.

Papadimitriou & Steiglitz מייחסים את האלגוריתם APPROX-VERTEX-TOUR ל-F. Gavril ול-M. Yannakakis. האלגוריתם APPROX-TSP-TOUR מופיע במאמר מצוין מאת Rosenkrantz, Stearns & Lewis [170]. משפט 37.3 הוא של Sahni & Gonzalez [172]. ניתוח ההיוריסטיקה החמדנית עבור בעיית כיסוי הקבוצה מבוסס על ההוכחה של תוצאה כללית יותר שפורסמה על-ידי Chvátal [42]; תוצאה בסיסית זו, כפי שהיא מוצגת כאן, היא של Johnson [113] ו-Lovász [141]. האלגוריתם APPROX-SUBSET-SUM וניתוחו מבוססים באופן רופף על אלגוריתמי קירוב דומים עבור בעיית התרמיל ובעיית סכום התת-קבוצה שהוצגו על-ידי Ibarra & Kim [111].

ביבליוגרפיה

[1] Milton Abramowitz and Irene A. Stegun, editors. *Handbook of Mathematical Functions*. Dover, 1965.

[2] G. M. Adel'son-Vel'skii and E. M. Landis. An algorithm for the organization of information. *Soviet Mathematics Doklady*, 3:1259–1263, 1962.

[3] Leonard M. Adleman, Carl Pomerance, and Robert S. Rumely. On distinguishing prime numbers from composite numbers. *Annals of Mathematics*, 117:173–206, 1983.

[4] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[5] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.

[6] Ravindra K. Ahuja, Kurt Mehlhorn, James B. Orlin, and Robert E. Tarjan. Faster algorithms for the shortest path problem. Technical Report 193, MIT Operations Research Center, 1988.

[7] Howard H. Aiken and Grace M. Hopper. The automatic sequence controlled calculator. In Brian Randell, editor, *The Origins of Digital Computers*, pages 203–222. Springer-Verlag, third edition, 1982.

[8] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 1–9, 1983.

[9] Selim G. Akl. *The Design and Analysis of Parallel Algorithms*. Prentice-Hall, 1989.

[10] Richard J. Anderson and Gary L. Miller. Deterministic parallel list ranking. In John H. Reif, editor, *1988 Aegean Workshop on Computing*, volume 319 of *Lecture Notes in Computer Science*, pages 81–90. Springer-Verlag, 1988.

[11] Richard J. Anderson and Gary L. Miller. A simple randomized parallel algorithm for list-ranking. Unpublished manuscript, 1988.

[12] Tom M. Apostol. *Calculus*, volume 1. Blaisdell Publishing Company, second edition, 1967.

[13] A. J. Atrubin. A one-dimensional real-time iterative multiplier. *IEEE Transactions on Electronic Computers*, EC-14(1):394–399, 1965.

[14] Sara Baase. *Computer Algorithms: Introduction to Design and Analysis*. Addison-Wesley, second edition, 1988.

[15] Eric Bach. Private communication, 1989.

[16] Eric Bach. Number-theoretic algorithms. In *Annual Review of Computer Science*, volume 4, pages 119–172. Annual Reviews, Inc., 1990.

[17] R. Bayer. Symmetric binary B-trees: Data structure and maintenance algorithms. *Acta Informatica*, 1:290–306, 1972.

[18] R. Bayer and E. M. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189, 1972.

[19] Paul W. Beame, Stephen A. Cook, and H. James Hoover. Log depth circuits for division and related problems. *SIAM Journal on Computing*, 15(4):994–1003, 1986.

[20] Pierre Beauchemin, Gilles Brassard, Claude Crépeau, Claude Goutier, and Carl Pomerance. The generation of random numbers that are probably prime. *Journal of Cryptology*, 1:53–64, 1988.

[21] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[22] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[23] Michael Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 80–86, 1983.

[24] Jon L. Bentley. *Writing Efficient Programs*. Prentice-Hall, 1982.

[25] Jon L. Bentley. *Programming Pearls*. Addison-Wesley, 1986.

[26] Jon L. Bentley, Dorothea Haken, and James B. Saxe. A general method for solving divide-and-conquer recurrences. *SIGACT News*, 12(3):36–44, 1980.

[27] William H. Beyer, editor. *CRC Standard Mathematical Tables*. The Chemical Rubber Company, 1984.

[28] Patrick Billingsley. *Probability and Measure*. John Wiley & Sons, second edition, 1986.

[29] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

[30] Béla Bollobás. *Random Graphs*. Academic Press, 1985.

[31] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. American Elsevier, 1976.

[32] Robert S. Boyer and J. Strother Moore. A fast string-searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[33] Gilles Brassard and Paul Bratley. *Algorithmics: Theory and Practice*. Prentice-Hall, 1988.

[34] Richard P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21(2):201–206, 1974.

[35] Richard P. Brent. An improved Monte Carlo factorization algorithm. *BIT*, 20(2):176–184, 1980.

[36] Mark R. Brown. *The Analysis of a Practical and Nearly Optimal Priority Queue*. PhD thesis, Computer Science Department, Stanford University, 1977. Technical Report STAN-CS-77-600.

[37] Mark R. Brown. Implementation and analysis of binomial queue algorithms. *SIAM Journal on Computing*, 7(3):298–319, 1978.

[38] Arthur W. Burks, editor. *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966.

[39] Joseph J. F. Cavanagh. *Digital Computer Arithmetic*. McGraw-Hill, 1984.

[40] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–507, 1952.

[41] Kai Lai Chung. *Elementary Probability Theory with Stochastic Processes*. Springer-Verlag, 1974.

[42] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[43] V. Chvátal, D. A. Klarner, and D. E. Knuth. Selected combinatorial research problems. Technical Report STAN-CS-72-292, Computer Science Department, Stanford University, 1972.

[44] Alan Cobham. The intrinsic computational difficulty of functions. In *Proceedings of the 1964 Congress for Logic, Methodology, and the Philosophy of Science*, pages 24–30. North-Holland, 1964.

[45] H. Cohen and H. W. Lenstra, Jr. Primality testing and jacobi sums. *Mathematics of Computation*, 42(165):297–330, 1984.

[46] Richard Cole. Parallel merge sort. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pages 511–516. IEEE Computer Society, 1986.

[47] Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.

[48] D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, 1979.

[49] Stephen Cook. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[50] Stephen Cook, Cynthia Dwork, and Rüdiger Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM Journal on Computing*, 15(1):87–97, 1986.

[51] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, April 1965.

[52] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 1–6, 1987.

[53] George B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[54] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[55] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[56] John D. Dixon. Factorization and primality tests. *The American Mathematical Monthly*, 91(6):333–352, 1984.

[57] Alvin W. Drake. *Fundamentals of Applied Probability Theory*. McGraw-Hill, 1967.

[58] James R. Driscoll, Harold N. Gabow, Ruth Shrairman, and Robert E. Tarjan. Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation. *Communications of the ACM*, 31(11):1343–1354, 1988.

[59] James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. Making data structures persistent. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 109–121, 1986.

[60] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1987.

[61] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[62] Jack Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1:126–136, 1971.

[63] Jack Edmonds and Richard M. Karp. Theoretical improvements in the algorithmic efficiency for network flow problems. *Journal of the ACM*, 19:248–264, 1972.

[64] G. Estrin, B. Gilchrist, and J. H. Pomerene. A note on high-speed digital multiplication. *IRE Transactions on Electronic Computers*, 5(3):140, 1956.

[65] Shimon Even. *Graph Algorithms*. Computer Science Press, 1979.

[66] William Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, third edition, 1968.

[67] M. J. Fischer and A. R. Meyer. Boolean matrix multiplication and transitive closure. In *Proceedings of the Twelfth Annual Symposium on Switching and Automata Theory*, pages 129–131. IEEE Computer Society, 1971.

[68] Robert W. Floyd. Algorithm 97 (SHORTEST PATH). *Communications of the ACM*, 5(6):345, 1962.

[69] Robert W. Floyd. Algorithm 245 (TREESORT). *Communications of the ACM*, 7:701, 1964.

[70] Robert W. Floyd and Ronald L. Rivest. Expected time bounds for selection. *Communications of the ACM*, 18(3):165–172, 1975.

[71] Lestor R. Ford, Jr., and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[72] Lestor R. Ford, Jr., and Selmer M. Johnson. A tournament problem. *The American Mathematical Monthly*, 66:387–389, 1959.

[73] Steven Fortune and James Wyllie. Parallelism in random access machines. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, pages 114–118, 1978.

[74] Michael L. Fredman and Michael E. Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, 1989.

[75] Michael L. Fredman and Robert E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

[76] Harold N. Gabow and Robert E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30(2):209–221, 1985.

[77] Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989.

[78] Zvi Galil and Joel Seiferas. Time-space-optimal string matching. *Journal of Computer and System Sciences*, 26(3):280–294, 1983.

[79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[80] Fănică Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.

[81] Alan George and Joseph W-H Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, 1981.

[82] Andrew V. Goldberg. *Efficient Graph Algorithms for Sequential and Parallel Computers*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1987.

[83] Andrew V. Goldberg, Éva Tardos, and Robert E. Tarjan. Network flow algorithms. Technical Report STAN-CS-89-1252, Computer Science Department, Stanford University, 1989.

[84] Andrew V. Goldberg and Serge A. Plotkin. Parallel $(\delta + 1)$ coloring of constant-degree graphs. *Information Processing Letters*, 25(4):241–245, 1987.

[85] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum flow problem. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 136–146, 1986.

[86] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[87] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[89] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1983.

[90] G. H. Gonnet. *Handbook of Algorithms and Data Structures*. Addison-Wesley, 1984.

[91] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1:132–133, 1972.

[92] R. L. Graham and Pavol Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

[93] Leo J. Guibas and Robert Sedgewick. A diochromatic framework for balanced trees. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, pages 8–21. IEEE Computer Society, 1978.

[94] Frank Harary. *Graph Theory*. Addison-Wesley, 1969.

[95] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[96] Frederick J. Hill and Gerald R. Peterson. *Introduction to Switching Theory and Logical Design*. John Wiley & Sons, second edition, 1974.

[97] C. A. R. Hoare. Algorithm 63 (partition) and algorithm 65 (find). *Communications of the ACM*, 4(7):321–322, 1961.

[98] C. A. R. Hoare. Quicksort. *Computer Journal*, 5(1):10–15, 1962.

[99] W. Hoeffding. On the distribution of the number of successes in independent trials. *Annals of Mathematical Statistics*, 27:713–721, 1956.

[100] Micha Hofri. *Probabilistic Analysis of Algorithms*. Springer-Verlag, 1987.

[101] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[102] John E. Hopcroft and Robert E. Tarjan. Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

[103] John E. Hopcroft and Jeffrey D. Ullman. Set merging algorithms. *SIAM Journal on Computing*, 2(4):294–303, 1973.

[104] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[105] Ellis Horowitz and Sartaj Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, 1978.

[106] T. C. Hu and M. T. Shing. Some theorems about matrix multiplication. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, pages 28–35. IEEE Computer Society, 1980.

[107] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[108] Kai Hwang. *Computer Arithmetic: Principles, Architecture, and Design*. John Wiley & Sons, 1979.

[109] Kai Hwang and Fayé A. Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill, 1984.

[110] Kai Hwang and Doug DeGroot. *Parallel Processing for Supercomputers and Artificial Intelligence*. McGraw-Hill, 1989.

[111] Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4):463–468, 1975.

[112] R. A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2:18–21, 1973.

[113] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[114] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, 1977.

[115] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[116] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[117] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. Technical Report TR-31-81, Aiken Computation Laboratory, Harvard University, 1981.

[118] Richard M. Karp and Vijaya Ramachandran. A survey of parallel algorithms for shared-memory machines. Technical Report UCB/CSD 88/408, Computer Science Division (EECS), University of California, Berkeley, 1988.

[119] A. V. Karzanov. Determining the maximal flow in a network by the method of preflows. *Soviet Mathematics Doklady*, 15:434–437, 1974.

[120] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(2):287–299, 1986.

[121] Donald E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, 1968. Second edition, 1973.

[122] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 1969. Second edition, 1981.

[123] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, 1973.

[124] Donald E. Knuth. Big omicron and big omega and big theta. *ACM SIGACT News*, 8(2):18–23, 1976.

[125] Donald E. Knuth, James H. Morris, Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[126] Zvi Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, 1970.

[127] Bernhard Korte and László Lovász. Mathematical structures underlying greedy algorithms. In F. Gecseg, editor, *Fundamentals of Computation Theory*, number 117 in Lecture Notes in Computer Science, pages 205–209. Springer-Verlag, 1981.

[128] Bernhard Korte and László Lovász. Structural properties of greedoids. *Combinatorica*, 3:359–374, 1983.

[129] Bernhard Korte and László Lovász. Greedoids—a structural framework for the greedy algorithm. In W. Pulleybank, editor, *Progress in Combinatorial Optimization*, pages 221–243. Academic Press, 1984.

[130] Bernhard Korte and László Lovász. Greedoids and linear objective functions. *SIAM Journal on Algebraic and Discrete Methods*, 5(2):229–238, 1984.

[131] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.

[132] Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart, and Winston, 1976.

[133] Eugene L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem*. John Wiley & Sons, 1985.

[134] C. Y. Lee. An algorithm for path connection and its applications. *IRE Transactions on Electronic Computers*, EC-10(3):346–365, 1961.

[135] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Networks and Algorithms*. Morgan-Kaufmann, in preparation.

[136] Debra A. Lelewer and Daniel S. Hirschberg. Data compression. *ACM Computing Surveys*, 19(3):261–296, 1987.

[137] H. W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of Mathematics*, 126:649–673, 1987.

[138] L. A. Levin. Universal sorting problems. *Problemy Peredachi Informatsii*, 9(3):265–266, 1973. In Russian.

[139] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, 1981.

[140] C. L. Liu. *Introduction to Combinatorial Mathematics*. McGraw-Hill, 1968.

[141] László Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[142] Udi Manber. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley, 1989.

[143] William J. Masek and Michael S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

[144] Kurt Mehlhorn. *Sorting and Searching*, volume 1 of *Data Structures and Algorithms*. Springer-Verlag, 1984.

[145] Kurt Mehlhorn. *Graph Algorithms and NP-Completeness*, volume 2 of *Data Structures and Algorithms*. Springer-Verlag, 1984.

[146] Kurt Mehlhorn. *Multidimensional Searching and Computational Geometry*, volume 3 of *Data Structures and Algorithms*. Springer-Verlag, 1984.

[147] Gary L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300–317, 1976.

[148] Louis Monier. *Algorithmes de Factorisation D'Entiers*. PhD thesis, L'Université Paris-Sud, Centre D'Orsay, 1980.

[149] Louis Monier. Evaluation and comparison of two efficient probabilistic primality testing algorithms. *Theoretical Computer Science*, 12(1):97–108, 1980.

[150] Edward F. Moore. The shortest path through a maze. In *Proceedings of the International Symposium on the Theory of Switching*, pages 285–292. Harvard University Press, 1959.

[151] Ivan Niven and Herbert S. Zuckerman. *An Introduction to the Theory of Numbers*. John Wiley & Sons, fourth edition, 1980.

[152] Yu. Ofman. On the algorithmic complexity of discrete functions. *Soviet Physics Doklady*, 7(7):589–591, 1963. English translation.

[153] Alan V. Oppenheim and Alan S. Willsky, with Ian T. Young. *Signals and Systems*. Prentice-Hall, 1983.

[154] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[155] Michael S. Paterson, 1974. Unpublished lecture, Ile de Berder, France.

[156] J. M. Pollard. A Monte Carlo method for factorization. *BIT*, 15:331–334, 1975.

[157] Carl Pomerance. On the distribution of pseudoprimes. *Mathematics of Computation*, 37(156):587–593, 1981.

[158] Carl Pomerance. The quadratic sieve factoring algorithm. In T. Beth, N. Cot, and I. Ingemarrson, editors, *Advances in Cryptology*, volume 209 of *Lecture Notes in Computer Science*, pages 169–182. Springer-Verlag, 1984.

[159] Carl Pomerance, editor. *Proceedings of the AMS Symposia in Applied Mathematics: Computational Number Theory and Cryptography*. American Mathematical Society, to appear.

[160] Franco P. Preparata and Micheal Ian Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[161] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1986.

[162] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1988.

[163] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[164] Paul W. Purdom, Jr., and Cynthia A. Brown. *The Analysis of Algorithms*. Holt, Rinehart, and Winston, 1985.

[165] Michael O. Rabin. Probabilistic algorithms. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 21–39. Academic Press, 1976.

[166] Michael O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12:128–138, 1980.

[167] Edward M. Reingold, Jürg Nievergelt, and Narsingh Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice-Hall, 1977.

[168] Hans Riesel. *Prime Numbers and Computer Methods for Factorization*. Progress in Mathematics. Birkhäuser, 1985.

[169] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. See also U.S. Patent 4,405,829.

[170] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6:563–581, 1977.

[171] Y. A. Rozanov. *Probability Theory: A Concise Course*. Dover, 1969.

[172] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23:555–565, 1976.

[173] John E. Savage. *The Complexity of Computing*. John Wiley & Sons, 1976.

[174] Robert Sedgewick. Implementing quicksort programs. *Communications of the ACM*, 21(10):847–857, 1978.

[175] Robert Sedgewick. *Algorithms*. Addison-Wesley, second edition, 1988.

[176] Michael I. Shamos and Dan Hoey. Geometric intersection problems. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, pages 208–215. IEEE Computer Society, 1975.

[177] Daniel D. Sleator and Robert E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[178] Daniel D. Sleator and Robert E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, 1985.

[179] Joel Spencer. *Ten Lectures on the Probabilistic Method*. Regional Conference Series on Applied Mathematics (No. 52). SIAM, 1987.

[180] Staff of the Computation Laboratory. *Description of a Relay Calculator*, volume XXIV of *The Annals of the Computation Laboratory of Harvard University*. Harvard University Press, 1949.

[181] Gilbert Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 1986.

[182] Gilbert Strang. *Linear Algebra and Its Applications*. Harcourt Brace Jovanovich, third edition, 1988.

[183] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 14(3):354–356, 1969.

[184] T. G. Szymanski. A special case of the maximal common subsequence problem. Technical Report TR-170, Computer Science Laboratory, Princeton University, 1975.

[185] Robert E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[186] Robert E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

[187] Robert E. Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences*, 18(2): 110–127, 1979.

[188] Robert E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, 1983.

[189] Robert E. Tarjan. Amortized computational complexity. *SIAM Journal on Algebraic and Discrete Methods*, 6(2):306–318, 1985.

[190] Robert E. Tarjan and Jan van Leeuwen. Worst-case analysis of set union algorithms. *Journal of the ACM*, 31(2):245–281, 1984.

[191] Robert E. Tarjan and Uzi Vishkin. An efficient parallel biconnectivity algorithm. *SIAM Journal on Computing*, 14(4):862–874, 1985.

[192] George B. Thomas, Jr., and Ross L. Finney. *Calculus and Analytic Geometry*. Addison-Wesley, seventh edition, 1988.

[193] Leslie G. Valiant. Parallelism in comparison problems. *SIAM Journal on Computing*, 4(3):348–355, 1975.

[194] P. van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, pages 75–84. IEEE Computer Society, 1975.

[195] Uzi Vishkin. Implementation of simultaneous memory address access in models that forbid it. *Journal of Algorithms*, 4(1):45–50, 1983.

[196] Jean Vuillemin. A data structure for manipulating priority queues. *Communications of the ACM*, 21(4):309–315, 1978.

[197] C. S. Wallace. A suggestion for a fast multiplier. *IEEE Transactions on Electronic Computers*, EC-13(1):14–17, 1964.

[198] Stephen Warshall. A theorem on boolean matrices. *Journal of the ACM*, 9(1):11–12, 1962.

[199] A. Weinberger and J. L. Smith. A one-microsecond adder using one-megacycle circuitry. *IRE Transactions on Electronic Computers*, EC-5(2), 1956.

[200] Hassler Whitney. On the abstract properties of linear dependence. *American Journal of Mathematics*, 57:509–533, 1935.

[201] Herbert S. Wilf. *Algorithms and Complexity*. Prentice-Hall, 1986.

[202] J. W. J. Williams. Algorithm 232 (heapsort). *Communications of the ACM*, 7:347–348, 1964.

[203] S. Winograd. On the algebraic complexity of functions. In *Actes du Congrès International des Mathématiciens*, volume 3, pages 283–288, 1970.

[204] James C. Wyllie. *The Complexity of Parallel Computations*. PhD thesis, Department of Computer Science, Cornell University, 1979.

[205] Andrew C.-C. Yao. A lower bound to finding convex hulls. *Journal of the ACM*, 28(4):780–787, 1981.

מפתח מונחים עברי־אנגלי

סימונים

× מכפלה וקטורית cross product **455**

∧ וגם AND **245, 327**

¬ לא NOT **245**

∨ או OR **245, 327**

⊕ אופרטור אלגברי algebraic operator **326**

אופרטור על חבורות group operator **391**

אופרטור סיכום עבור חוגים־למחצה סגורים sum operator for closed semirings **170**

אר־מוציא XOR exclusive-or **245**

⊙ אופרטור אלגברי algebraic operator **327**

אופרטור הרחבה עבור חוגים־למחצה סגורים extension operator for closed semirings **170**

אופרטור מצבי נשא carry-status operator **251**

אופרטור קונבולוציה convolution operator **358**

אופרטור לחישוב מקבילי של רישות parallel-prefix operator **281**

* אופרטור סגור closure operator **173, 484**

| היחס "מחלק את" divides relation **380**

$\equiv$ שקול מודולו n equivalent modulo n n **381**

$\not\equiv$ לא שקול מודולו n not equivalent modulo n n **381**

$[a]_n$ מחלקת שקילות מודולו n equivalence class modulo n n **381**

$+_n$ חיבור מודולו n addition modulo n n **391**

$\cdot_n$ כפל מודולו n multiplication modulo n n **391**

ε המחרוזת הריקה empty string **427, 484**

$\preceq$ יחס הרישא prefix relation **427**

$\succeq$ יחס הסיפא suffix relation **427**

$\sim$ יחס הדמיון similar relation **450**

$\succeq_L$ היחס "מעל" above relation **460**

$\leq_p$ היחס "ניתן לרדוקציה בזמן פולינומיאלי" polynomial-time reducibility relation **492**

Z_n^* איברי החבורה הכפלית מודולו n elements of multiplicative group modulo n **392**

Z_n^+ איברים שונים מאפס של Z_n^* nonzero elements of Z_n^* **413**

א

אב parent

בחיפוש בעץ לרוחב in a breadth-first tree **84**

אב קדמון ביותר forefather **101**

אב קדמון משותף עמוק ביותר least common ancestor **74** (בעיה)

אוגר register **266**

אוטומט automaton

התאמת מחרוזות string-matching **435-439**

סופי finite **435**

תאי cellular **310**

אוטומט התאמת־מחרוזות string-matching automaton **435-439, 440** (תרגילים)

אוטומט סופי finite automaton **435**

עבור התאמת מחרוזות for string matching **435-439**

אוטומט תאי cellular automaton **310**

אונרי unary **483**

אופרטור הרחבה עבור חוגים־למחצה סגורים extension operator for closed semirings **170**

אופרטור מצבי נשא $\otimes$ carry-status operator **251**

אופרטור סיכום $\oplus$ summary operator **170**

איבר יחידה identity **170, 326, 391**

אידמפוטנטיות תחת אופרטור idempotence under an operator **170**

איחוד joining

של עצי 2-3-4 of 2-3-4 trees **19** (בעיה)

איחוד union

מימוש של UNION על־ידי יער קבוצות זרות disjoint-set-forest implementation of **63**

מימוש של UNION על־ידי רשימות מקושרות linked-list implementation of **59-61**, **61** (תרגיל)

של שפות of languages **484**

איחוד uniting

של ערימות of heaps **21**

של ערימות בינומיות of binomial heaps **27-32**

של ערימות פיבונצ'י of Fibonacci heaps **43**

של ערימות 2-3-4 of 2-3-4 heaps **36-37** (בעיה)

איחוד על פי דרגה union by rank **62**

אי־יציבות נומרית numerical instability **331**

אי־שוויון המשולש triangle inequality **525**

אי־שוויון ערכי פונקציית הסיפא suffix-function inequality **437**

אי־תלות לינארית linear independence **316**

איטרציית ניוטון Newton iteration **271** (בעיה)

אילוץ קיבול capacity constraint **179**

אילוצי הפרשים difference constraints **142-147**

אימות verification **487-491**

אינדקס של איבר ב־$\mathbf{Z}_n^*$ index of an element of $\mathbf{Z}_n^*$ **403**

אינטרפולציה באמצעות פולינום interpolation by a polynomial **359**

בשורשי היחידה המרוכבים at complex roots of unity **368-369**

אינטרפולציה באמצעות פונקציית ספליין ממעלה שלישית interpolation by a cubic spline **353** (בעיה)

איפוס נשא carry kill **251**

איפוס סיבית נשא kill a carry bit **251**

אישור certificate

במערכת הצפנה in a cryptosystem **411**

עבור אלגוריתמי אימות for verification algorithms **488**

אלגוריתם KMP KMP algorithm **440-446**

אלגוריתם PRAM יעיל בעבודה work-efficient PRAM algorithm **280**

חישוב רישות מקבילי parallel prefix computation **296-301**

כפל מטריצה בווקטור על EREW PRAM EREW PRAM **296** (תרגיל)

כפל מטריצות על CRCW PRAM matrix multiplication on a CRCW PRAM **296** (תרגיל)

אלגוריתם gcd בינרי binary gcd algorithm **423** (בעיה)

אלגוריתם אדמונדס־קארפ Edmonds-Karp algorithm **193**

אלגוריתם אקראי randomized algorithm

הטלת מטבע ב־ coin flipping in **299**

היוריסטיקת רו של פולארד Pollard's rho heuristic **419-422**

מבחן הראשוניות של מילר ורבין Miller-Rabin primality test **414-418**

עבור חישוב רישות מקבילי יעיל־בעבודה for work-efficient parallel prefix **296-301**

עבור כפל מטריצות בוליאניות for boolean matrix multiplication **352** (בעיה)

אלגוריתם בויאר־מור Boyer-Moore algorithm **447-452**

אלגוריתם בלמן־פורד Bellman-Ford algorithm **136-139**

באלגוריתם של ג'ונסון in Johnson's algorithm **169**

השיפור של ין Yen's improvement to **147**

ופונקציות מטרה and objective functions **147** (תרגילים)

לפתרון מערכות של אילוצי הפרשים to solve systems of difference constraints **145**

אלגוריתם דחיפת־קדם־זרימה preflow-push algorithm **200-218, 222**

אלגוריתם הרם־וקדם lift-to-front algorithm **209-218**

אלגוריתם כללי generic algorithm **203-208**

למציאת זיווג מקסימלי בגרף דו־צדדי to find a maximum bipartite matching **209** (תרגיל)

על־ידי פריקת קדקוד גולש שגובהו מקסימלי

by discharging an overflowing vertex of maximum height **218** (תרגיל)

עם תור של קדקודים גולשים with a queue of overflowing vertices **218** (תרגיל)

פעולות בסיסיות ב־ basic operations in **202-203**

אלגוריתם האיכר הרוסי Russian peasant's algorithm **268**

אלגוריתם האליפסואיד ellipsoid algorithm **142**, **151**

אלגוריתם הכפל של בית הספר היסודי grade-school multiplication algorithm **259**

אלגוריתם הנפה־הריבועית עבור פירוק לגורמים quadratic-sieve factorization algorithm **425**

אלגוריתם הסימפלקס simplex algorithm **142**, **151**

אלגוריתם הרם־וקדם lift-to-front algorithm **209-218**

אלגוריתם חמדן greedy algorithm

האלגוריתם של דייקסטרה Dijkstra's algorithm **131-135**

האלגוריתם של פרים Prim's algorithm **115-117**

האלגוריתם של קרוסקל Kruskal's algorithm **113-115**

למציאת כיסוי קדקודים אופטימלי עבור עץ for finding optimal vertex cover for a tree **525** (תרגיל)

עבור בעיית כיסוי הקבוצה for the set-covering problem **529-533**

עבור הבעיה המשוקללת של כיסוי קבוצה for the weighted set-covering problem **538** (בעיה)

עבור עץ פורש מינימלי for minimum spanning tree **109**

אלגוריתם כללי של דחיפת־קדם־זרימה generic preflow-push algorithm **203-208**

אלגוריתם מקבילי parallel algorithm **274-310**

ראה גם: מעגל מקבילי, רשת השוואה, CRCW, CREW, EREW, מכונת גישה אקראית מקבילית, רשת מיון

see also arithmetic circuit, comparison network, CRCW, CREW, EREW, parallel random-access machine, sorting network

אלגוריתם פלויד־וורשאל Floyd-Warshall algorithm **159-163**, **164** (תרגילים), **175** (תרגיל)

אלגוריתם קירוב approximation algorithm **521-539**

עבור אריזה בתיבות for bin packing **538** (בעיה)

עבור בעיית הכיסוי על־ידי קדקודים for the vertex-cover problem **523-525**

עבור בעיית הסוכן הנוסע for the traveling salesman problem **525-529**

עבור בעיית הקליקה המקסימלית for the maximum-clique problem **538** (בעיה)

עבור בעיית כיסוי הקבוצה for the set-covering problem **529-533**

עבור בעיית סכום התת־קבוצה for the subset-sum problem **533-537**

עבור הגרסה המשוקללת של בעיית כיסוי הקבוצה for the weighted set-covering problem **538** (בעיה)

אלגוריתם קנות־מוריס־פראט Knuth-Morris-Pratt algorithm **440-446**

אלגוריתם רבין־קארפ Rabin-Karp algorithm **430-434**

אלגוריתם רדוקציה reduction algorithm **492**

אלפבית alphabet **435**, **484**

אלפבית קלט input alphabet **435**

אפס של פולינום מודולו מספר ראשוני zero of a polynomial modulo a prime **399** (תרגיל)

ארביטראז' arbitrage **148** (בעיה)

אריזה בתיבות bin packing **538**

אריתמטיקה מודולרית modular arithmetic **390-396**

ב

בור sink **83** (תרגיל), **179**, **182-183**

בור־על supersink **183**

בעיה problem

אופטימיזציה optimization **482**

הכרעה decision **482**

לא ניתנת לפתרון סביר intractable **480**

מופשטת abstract **481**

ניתנת לפתרון סביר tractable **480**

פתרון ל־ solution to **482**

קונקרטית concrete **482**

בעיה בלתי־מקוונת off-line problem

אבות קדמונים משותפים עמוקים ביותר least common ancestors **74** (בעיה)

מינימום minimum **73** (בעיה)

בעיה מופשטת abstract problem **481**

בעיה קונקרטית concrete problem **482**

בעיית אופטימיזציה optimization problem **482**

אלגוריתמי קירוב עבור approximation algorithms for **521-539**

בעיית אפשרות feasibility problem **142**

בעיית דרכי מילוט escape problem **218** (בעיה)

בעיית האיזומורפיות של תת־גרפים subgraph-isomorphism problem **518** (תרגיל)

בעיית הכיסוי על־ידי קדקודים vertex-cover problem

אלגוריתם קירוב עבור approximation algorithm for **523-525**

ובעיית הקליקה and clique problem **525** (תרגיל)

שלמות ב־NP של NP-completeness of **508-509**

בעיית הכרעה decision problem **482**

בעיית המסלול ההמילטוני hamiltonian-path problem **518** (תרגיל)

בעיית המעגל ההמילטוני hamiltonian-cycle problem **488**, **512-517**

בעיית המעגל הפשוט הארוך ביותר longest-simple-cycle problem **518** (תרגיל)

בעיית הסוכן הנוסע traveling-salesman problem

אלגוריתם קירוב עבור approximation algorithm for **525-529**

בלי אי־שוויון המשולש without triangle inequality **528**

עם אי־שוויון המשולש with triangle inequality **525-527**

צוואר בקבוק bottleneck **529** (תרגיל)

שלמות ב־NP NP-completeness of **517-518**

בעיית הקליקה clique problem **506-508**

אלגוריתם קירוב עבור approximation algorithm for **538** (בעיה)

ובעיית הכיסוי על־ידי קדקודים and vertex-cover problem **525** (תרגיל)
בעיית הקמור המקווטת on-line convex-hull problem **473** (תרגיל)
בעיית זוג הנקודות הרחוקות ביותר farthest-pair problem **465**
בעיית חלוקת הקבוצה set-partition problem **518** (תרגיל)
בעיית כיסוי הקבוצה set-covering problem **533-529**
המשוקללת weighted **538** (בעיה)
בעיית סכום התת־קבוצה subset-sum problem
אלגוריתם קירוב עבור approximation algorithm for **537-533**
עם מטרה בייצוג אונרי with unary target **518** (תרגיל)
שלמות ב־NP של NP-completeness of **512-510**
בעיית ספיקות הנוסחאות formula-satisfiability problem **502-500**
בעיית עצירה halting problem **480**
בעיית צביעת־גרף graph-coloring problem **519** (בעיה)
בעיית צוואר הבקבוק של הסוכן הנוסע bottleneck traveling-salesman problem **529** (תרגיל)
בעיית קביעת עומק depth-determination problem **73** (בעיה)

ג

גובה height
של עץ בינומי of a binomial tree **23**
של עץ־B of a B-tree **9**
של ערימת פיבונצ'י of a Fibonacci heap **54** (תרגיל)
גודל size
של כיסוי על־ידי קדקודים of a vertex cover **523 ,508**
של מעגל צירופי of a combinational circuit **293 ,248**
של עץ בינומי of a binomial tree **23**
של עץ קבוצות־זרות of a disjoint-set tree **68**
של קלט לאלגוריתם of an algorithm's input **484-482 ,380-379**
של קליקה of a clique **506**
של תת־עץ בערימת פיבונצ'י of a subtree in a Fibonacci heap **52**
גורם factor **380**
גורם חזרה במחרוזת repetition factor of a string **452** (בעיה)
גיאומטריה חישובית computational geometry **479-454**
גיזום רשימה trimming of a list **534**
גיזום של ערימת פיבונצ'י pruning a Fibonacci heap **55** (בעיה)
גישת הפרד־ומשול divide-and-conquer method
להיפוך מטריצות for matrix inversion **345-344**
להמרה מייצוג בינרי לעשרוני for conversion of binary to decimal **384** (תרגיל)
למציאת הקמור for finding the convex hull **465**

למציאת זוג הנקודות הקרובות ביותר for finding the closest pair of points **477-473**
עבור אלגוריתם שטראסן for Strassen's algorithm **326-320**
עבור התמרת פוריה המהירה for Fast Fourier Transform **368-366**
עבור כפל for multiplication **375** (בעיה)
גמ"ל (גרף מכוון ללא־מעגלים) dag (directed acyclic graph)
אלגוריתם מסלולים קצרים ביותר ממקור יחיד עבור single-source shortest-paths algorithm for **141-139**
מיון טופולוגי של topological sort of **99-97**
ובעיית המסלול ההמילטוני and hamiltonian-path problem **491** (תרגיל)
וקשתות אחורה and back edges **98**
גרסת 0-1 של בעיית התרמיל 0-1 knapsack problem **487** (תרגיל)
גרסת 0-1 של בעיית התכנון בשלמים 0-1 integer-programming problem **518** (תרגיל)
גרעין של מצולע kernel of a polygon **473** (תרגיל)
גרף d־רגולרי d-regular graph **200** (תרגיל)
גרף graph
אלגוריתם על algorithms for **222-79**
דליל sparse **80**
המילטוני hamiltonian **487**
חיפוש לעומק של depth-first search of **97-90**
חיפוש לרוחב של breadth-first search of **90-83**
ייצוג על־ידי מטריצת סמיכויות adjacency-matrix representation of **81**
ייצוג על־ידי רשימות סמיכות adjacency-list representation of **80**
לא־המילטוני nonhamiltonian **487**
מטריצת קשירויות של incidence matrix of **83** (תרגיל), **510**
מסלול סיור בתוך tour of **517**
מסלולים קצרים ביותר בתוך shortest path in **86**
משוקלל weighted **81**
משלים של complement of **508**
צפוף dense **80**
ϵ־צפוף ϵ-dense **176** (בעיה)
ראה גם: גרף מכוון ללא מעגלים (גמ"ל), גרף מכוון, רשת זרימה, גרף בלתי־מכוון, עץ
see also directed acyclic graph (dag), directed graph, flow network, undirected graph, tree
גרף אילוצים constraint graph **145-144**
גרף בלתי מכוון undirected graph
גשר של bridge of **105** (בעיה)
המרה ל־, ממולטי־גרף converting to, from a multigraph **83** (תרגיל)
זיווג של matching of **197**
חישוב עץ פורש מינימלי עבור computing a minimum spanning tree in **119-108**
כיסוי על־ידי קדקודים של vertex cover of **523 ,508**

נקודת הפרדה של articulation point of **105** (בעיה)

צביעה של coloring of **302, 519** (בעיה)

קבוצה בלתי תלויה של independent set of **302, 519** (בעיה)

קליקה בתוך clique in **506**

רכיב דו־קשיר של biconnected component of **105** (בעיה)

שריג grid **218** (בעיה)

d־רגולרי d-regular **200** (תרגיל)

גרף דו־צדדי bipartite graph

רשת הזרימה המתאימה ל־ corresponding flow network of **197**

קביעה של determination of **90** (תרגיל)

גרף דינמי dynamic graph

אלגוריתם למציאת עץ פורש מינימלי של minimum-spanning-tree algorithm for **118** (תרגיל)

סגור טרנזיטיבי של transitive closure of **176** (בעיה)

גרף דליל sparse graph **80**

גרף חד־קשיר singly connected graph **97** (תרגיל)

גרף לא־המילטוני nonhamiltonian graph **487**

גרף מכוון directed graph

גרף אילוצים constraint graph **144-145**

גרף מוחלף של transpose of **82** (תרגיל)

גרף רכיבים component graph **104-105** (תרגילים)

דיאגרמת PERT PERT chart **141, 141** (תרגיל)

חד־קשיר singly connected **97** (תרגיל)

כיסוי על־ידי מסלולים של path cover of **219** (בעיה)

מסלולים קצרים ביותר בין כל הזוגות בתוך all-pairs shortest paths in **152-177**

מסלול־אוילר Euler-tour of **106** (בעיה)

מסלול קצר ביותר בתוך shortest path in **120**

מסלולים קצרים ביותר ממקור יחיד בתוך single-source shortest paths in **120-151**

סגור טרנזיטיבי של transitive closure of **163**

קשיר למחצה semiconnected **105** (תרגיל)

ריבוע של square of **83** (תרגיל)

תחשיב מסלולים ב־ calculus of paths in **170-173**

ראה גם: מעגל, גרף מכוון ללא מעגלים (גמ"ל), גרף, רשת

see also circuit, directed acyclic graph (dag), graph, network

גרף מכוון ללא־מעגלים (גמ"ל) directed acyclic graph (dag)

אלגוריתם למציאת מסלולים קצרים ביותר ממקור יחיד עבור single-source shortest-paths algorithm for **139-141**

ובעיית המסלול ההמילטוני and hamiltonian-path problem **491** (תרגיל)

וקשתות אחורה and black edges **98**

מיון טופולוגי של topological sort of **97-99**

גרף צפוף dense graph **80**

ϵ־צפוף ϵ-dense **176** (בעיה)

גרף קשיר־למחצה semiconnected graph **105** (תרגיל)

גרף רכיבים component graph **104-105** (תרגילים)

גרף ϵ־צפוף ϵ-dense graph **176** (בעיה)

גשר bridge **105** (בעיה)

ד

דומה ($\sim$) similar **450**

דחייה rejection

על־ידי אוטומט סופי by a finite automaton **435**

על־ידי אלגוריתם by an algorithm **485**

דחיפה בלתי־מרווה nonsaturating push **203, 208**

דחיפה מרווה saturating push **203, 207**

דטרמיננטה determinant **317**

וכפל מטריצות and matrix multiplication **346** (תרגיל)

דיאגרמת PERT PERT chart **141, 141** (תרגיל)

דיסק disk **6-20**

ראה גם: התקן אחסון משני *see also* secondary storage

דירוג scaling

בזרימה מקסימלית in maximum flow **200** (בעיה)

במסלולים קצרים ביותר ממקור יחיד in single-source shortest paths **148** (בעיה)

דף על דיסק page on a disk **6, 19** (בעיה)

דרגה degree

מינימלית, של עץ־B minimum, of a B-tree **8**

מקסימלית, של ערימת פיבונצ'י maximum, of a Fibonacci heap **40, 48** (תרגיל), **52-54**

של שורש של עץ בינומי of a binomial-tree root **23**

דרגה rank

מלאה full **317**

של מטריצה of a matrix **316**

של עמודות column **316**

של צומת ביער קבוצות־זרות of a node in a disjoint-set forest **62, 68-69**

של שורות row **316**

דרגה מינימלית של עץ־B minimum degree of a B-tree **8**

דרגה מלאה full rank **317**

דרגה מקסימלית בערימת פיבונצ'י maximum degree in a Fibonacci heap **40, 48** (תרגיל), **52-54**

דרגת עמודות column rank **316**

דרגת שורות row rank **316**

ה

האלגוריתם המדורג של גאבו עבור מסלולים קצרים ביותר ממקור יחיד
Gabow's scaling algorithm for single-source shortest paths **148** (בעיה)
האלגוריתם הנאיבי להתאמת מחרוזות naive algorithm for string matching **430-428**
האלגוריתם של אוקלידס Euclid's algorithm **385-390, 423** (בעיה)
האלגוריתם של גולדברג, ראה: אלגוריתם דחיפת־קדם־זרימה Goldberg's algorithm, *see* preflow-push algorithm
האלגוריתם של ג'ונסון Johnson's algorithm **166-169**
האלגוריתם של דייקסטרה Dijkstra's algorithm **131-135**
באלגוריתם של ג'ונסון in Johnson's algorithm **169**
דמיון לאלגוריתם של פרים similarity to Prim's algorithm **115, 135**
דמיון לחיפוש לרוחב similarity to breadth-first search **135**
מימוש באמצעות ערימה implemented with a heap **134**
מימוש באמצעות ערימת פיבונצ'י implemented with a Fibonacci heap **134**
עם משקלי קשתות שלמים with integer edge weights **135** (תרגיל)
האלגוריתם של Tarjan לבעיה הלא־מקוונת של האבות הקדמונים המשותפים העמוקים ביותר
Tarjan's off-line least-common-ancestors algorithm **74** (בעיה)
האלגוריתם של פרים Prim's algorithm **115-117**
במציאת מסלול כמעט אופטימלי עבור בעיית הסוכן הנוסע in finding a near optimal traveling-salesman tour **525**
דמיון לאלגוריתם של דייקסטרה similarity to Dijkstra's algorithm **115, 135**
מימוש עם ערימה implemented with a heap **117**
מימוש עם ערימת פיבונצ'י implemented with a Fibonacci heap **117**
עבור גרפים דלילים for sparse graphs **118** (בעיה)
עם מטריצת סמיכויות with an adjacency matrix **117** (תרגיל)
עם משקלות קשתות שלמים with integer edge weights **117** (תרגיל)
האלגוריתם של קארמארקר Karmarkar's algorithm **142, 151**
האלגוריתם של קארפ למציאת מעגל בעל משקל ממוצע מינימלי
Karp's minimum mean-weight cycle algorithm **149** (בעיה)
האלגוריתם של קרוסקל Kruskal's algorithm **113-115**
עם משקלות קשתות שלמים with integer edge weights **117** (תרגיל)
האלגוריתם של שטראסן Strassen's algorithm **320-326**
וכפל מטריצות בוליאניות and boolean matrix multiplication **328-329, 352** (בעיה)
האלגוריתם של שמיר לכפל מטריצות בוליאניות Shamir's boolean matrix multiplication algorithm **352** (בעיה)
הגרסה המשוקללת של כיסוי קבוצה weighted set-covering problem **538** (בעיה)
הדמיה יעילה בעבודה work-efficient simulation **294-295**
הופכי inverse
בחבורה in a group **391**
חיבורי additive **327**
כפלי multiplicative **329**

כפלי מודולו n multiplicative modulo n **398**
של מטריצה מפירוק LUP of a matrix from a LUP decomposition **343**
של מטריצה of a matrix **316**
הופכי חיבורי additive inverse **327**
הופכי כפלי multiplicative inverse **329**
מודולו n modulo n **398**
הזוג הקרוב ביותר, מציאה של closest pair, finding **473-477**
היוריסטיקת האיחוד המשוקלל weighted-union heuristic **60**
היוריסטיקת המתאימה־הראשונה first-fit heuristic **538** (בעיה)
היוריסטיקת הנקודה הקרובה ביותר closest-point heuristic **529** (תרגיל)
היוריסטיקת הסיפא־הטובה good-suffix heuristic **450-452**
היוריסטיקת התו הרע bad-character heuristic **448-450**
היוריסטיקת רו של פולארד Pollard's rho heuristic **419-422**
היחס "מחלק את" ($|$) divides relation **380**
היחס "שליטי" dominates relation **478** (בעיה)
הימצאות על ישר אחד collinearity **455, 458** (תרגיל)
היסט בהתאמת מחרוזות shift in string matching **426**
היסט בלתי־תקף invalid shift **426**
היסט תקף valid shift **426**
היפוך מטריצות matrix inversion **342-346**
הכנסה insertion
לעצי־B into B-trees **13-15**
לערימות בינומיות into binomial heaps **32**
למצבי קו סריקה into sweep-line statuses **461**
לערימות פיבונצ'י into Fibonacci heaps **41-42**
לערימות 2-3-4 into 2-3-4 heaps **36** (בעיה)
הכרעה על־ידי אלגוריתם decision by an algorithm **485**
המחרוזת הריקה (ε) empty string **427, 484**
המילטוני hamiltonian
גרף graph **487**
מסלול path **491** (תרגיל)
מעגל cycle **487**
המסלולים הקצרים ביותר בין כל הזוגות all-pairs shortest paths **121**
אלגוריתם פלויד־וורשאל Floyd-Warshall algorithm **159-163**
באמצעות העלאה חוזרת בריבוע by repeated squaring **157-158**
באמצעות כפל מטריצות by matrix multiplication **154-159**
בגרפים ϵ־צפופים in ϵ-dense graphs **176** (בעיה)
האלגוריתם של ג'ונסון Johnson's algorithm **166-169**

המשפט זרימה־מקסימלית חתך־מינימלי max-flow min-cut theorem 191-190

הסריקה של גראהם Graham's scan 471-466

העלאה בחזקה exponentiation

חוזרת repeated 66-65

מודולרית modular 404

העלאה חוזרת בחזקה repeated exponentiation 66-65

העלאה חוזרת בריבוע repeated squaring

עבור העלאת מספר בחזקה for raising a number to a power 404

עבור מסלולים קצרים ביותר בין כל הזוגות for all-pairs shortest paths 158-157

העלאה מודולרית בחזקה modular exponentiation 404

הפונקציה AND (∧) AND function 327 ,245

על CRCW PRAM CRCW PRAM 289

הצבה אחורה back substitution 334-333

הצבה קדימה forward substitution 333-332

הצעדה של ג'ארביס Jarvis's march 472-471

הצפנת אפס־ידע zero-knowledge encryption 425

הקטנת ערך מפתח decreasing a key

בערימות בינומיות in binomial heaps 35-33

בערימות פיבונאצ'י in Fibonacci heaps 51-48

בערימות 2-3-4 in 2-3-4 heaps 36 (בעיה)

הקלה relaxation 131-125

הקפצת מצביעים pointer jumping 285-277

השיפור של ין לאלגוריתם בלמן־פורד Yen's improvement to the Bellman-Ford algorithm 147 (בעיה)

השמה assignment

מספקת satisfying 500 ,494

של ערכי אמת truth 500 ,494

השמה מספקת satisfying assignment 500 ,494

השמת ערכי אמת truth assignment 500 ,494

השפה הריקה (ϕ) empty language 484

התאמה לעקום curve fitting 351-348

התאמת מחרוזות string matching 453-426

אלגוריתם בוייר־מור Boyer-Moore algorithm 452-447

אלגוריתם קנות'־מוריס־פראט Knuth-Morris-Pratt algorithm 446-440

אלגוריתם רבין־קארפ Rabin-Karp algorithm 434-430

באמצעות אוטומטים סופיים by finite automata 440-434

האלגוריתם הנאיבי naive algorithm 430-428

המבוססת על גורמי חזרה based on repetition factors 452 (בעיה)

על CREW PRAM on a CREW PRAM 453 (בעיה)

עם תווי מרווח with gap characters 429 (תרגיל), 440 (תרגיל)

התאמת תבניות, ראה: התאמת מחרוזות pattern matching, *see* string matching

התייצבות ערכים במעגל settling of circuit values 246

התמרת chirp chirp transform 370 (תרגיל)

התמרת פוריה הבדידה Discrete Fourier Transform (DFT) 366-365

התמרת פוריה המהירה Fast Fourier Transform (FFT)

באמצעות אריתמטיקה מודולרית using modular arithmetic 377 (בעיה)

מימוש איטרטיבי של iterative implementation of 374-370

מימוש רקורסיבי של recursive implementation of 368-366

מעגל עבור circuit for 375-374

התפלגות distribution

בעלת קמור דליל sparse-hulled 479 (בעיה)

של המספרים הראשוניים of prime numbers 412

התפלגות בעלת קמור דליל sparse-hulled distribution 479 (בעיה)

התקן אחסון משני secondary storage

מחסניות על stacks on 19 (בעיה)

עץ חיפוש עבור search tree for 20-5

ו

וקטור vector 317-315 ,312

במישור in the plane 455

קונבולוציה של convolution of 358

מכפלה וקטורית של cross product of 455

וקטור יחידה unit vector 312

וקטור מְאפֵס null vector 317

וקטור עמודה column vector 312

וקטור שורה row vector 312

ז

זווית קוטבית, מיון נקודות לפי polar angle, sorting points by 458 (תרגיל)

זיווג matching

בגרף דו־צדדי משוקלל weighted bipartite 55

וזרימה מקסימלית and maximum flow 200-196

מחרוזות of strings 453-426

זיווג בגרף דו־צדדי bipartite matching 55, 196-200, 222

זיווג בגרף דו־צדדי משוקלל weighted bipartite matching 55

זיווג מושלם perfect matching 200 (תרגיל)

זיווג מקסימלי בגרף דו־צדדי maximum bipartite matching 196-200, 209 (תרגיל), 222

זיכרון ראשי primary (main) memory 6

זמן היע"מ CPU time 6

זמן השהיה propagation delay 246

זמן מחזור של שעון clock period 266

זמן על־פולינומיאלי superpolynomial time 480

זמן פולינומיאלי polynomial-time

אימות ב־ verification 491-487

אלגוריתם algorithm 480 ,379

הכרעה ב־ decision 485

ניתנות לחישוב ב־ computability 483

סכמת קירוב ב־ approximation scheme 522

פתירות ב־ solvability 482

קבלה ב־ acceptance 485

רדוקציה ב־ reducibility ($\leq_p$) 492

זמן פולינומיאלי לא דטרמיניסטי nondeterministic polynomial time 489 (הערה)

ראה גם: NP *see also* NP

זמן ריצה running time

של אלגוריתם על גרף of a graph algorithm 79

של מעגל צירופי of a combinational circuit 248

של רשת השוואה of a comparison network 229

זרים בזוגות pairwise relatively prime 383

זרים זה לזה relatively prime 383

זרימה flow 185-179

בעלת ערכים שלמים integer-valued 198

סכום sum 184 (תרגיל)

עודפת excess 201

ערך של value of 180-179

זרימה בעלת ערכים שלמים integer-valued flow 198

זרימה מקסימלית maximum flow 222-178

אלגוריתם הרם־וקדם lift-to-front algorithm 219-209

אלגוריתם מדורג scaling algorithm 220 (בעיה)

אלגוריתמי דחיפת־קדם־זרימה preflow-push algorithms 218-200

וזיווג מקסימלי בגרף דו־צדדי and maximum bipartite matching 200-196

עדכון updating 220 (בעיה)

עם קיבולים שליליים with negative capacities 221 (בעיה)

שיטת פורד־פולקרסון Ford-Fulkerson method 196-185

זרימה נטו net flow 180-179

זרימה נטו חיובית positive net flow 180

זרימה עודפת excess flow 201

ח

חבורה group 396-390

מעגלית cyclic 403

חבורה אבלית abelian group 391

חבורה חיבורית מודולו n additive group modulo n 391

חבורה כפלית מודולו n multiplicative group modulo n 392

חבורה מעגלית cyclic group 403

חבורה סופית finite group 391

חוג ring 328-327

חוג־למחצה סגור closed semiring 174-173 ,170

בתור מעין־חוג as a quasiring 327

עבור עץ פורש מינימלי for minimum spanning tree 177 (בעיה)

חותמת זמן timestamp 97-96 ,91 (תרגילים)

חזקה power

לא טריוויאלית nontrivial power 385 (תרגיל)

של איבר מודולו n of an element modulo n 406-402

k־ית kth power 385 (תרגיל)

חזקה לא טריוויאלית nontrivial power 385 (תרגיל)

חזקה k־ית kth power 385 (תרגיל)

חיבור addition

מגלגל־נשא ripple-carry 250-249

מודולו n modulo n $(+_n)$ 391

מטריצות of matrices 314

מעגלים עבור circuits for 259-249

סיביות סדרתי bit-serial 267-265

צופה נשא carry-lookahead 255-251

שומר־נשא carry-save 257-256

של פולינומים of polynomials 356

חיבור בחומרה hardwired 250

חיבור מגלגל־נשא ripple-carry addition 250-249

בהשוואה לחיבור סיביות סדרתי compared to bit-serial addition 267

חיבור סיביות סדרתי bit-serial addition 267-265

חיבור צופה־נשא carry-lookahead addition 255-251

חיבור שומר־נשא carry-save addition 264 ,257-256 (תרגיל)

חילוף תמונת סריקה transposing a raster image **309** (בעיה)

חיפוש searching

בעצי־B in B-trees **11-10**

חיפוש בינרי binary search

בחיפוש בעצי־B in searching B-trees **16** (תרגיל)

חיפוש לעומק depth-first search **97-90**

במיון טופולוגי in topological sorting **99-97**

במציאת נקודות הפרדה, גשרים ורכיבים דו־קשירים

in finding articulation points, bridges, and biconnected components **105** (בעיה)

במציאת רכיבים קשירים היטב in finding strongly connected components **105-100**

חיפוש לרוחב breadth-first search **90-83**

דמיון לאלגוריתם של דייקסטרה similarity to Dijkstra's algorithm **135**

ומסלולים קצרים ביותר and shortest paths **120 ,88-86**

חישוב רֵישׁוֹת prefix computation

בחיבור צופה־נשא in carry-lookahead addition **252**

גרסה יעילה בעבודה על EREW PRAM work-efficient version on an EREW PRAM **301-296**

על־ידי מעגל רֵישׁוֹת מקבילי by a parallel prefix circuit **255-253**

על רשימה on a list **281-280**

על EREW PRAM on an EREW PRAM **296 ,282-280** (תרגיל)

חישוב רֵישׁוֹת מקבילי parallel prefix

אלגוריתם מקבילי יעיל בעבודה work-efficient parallel algorithm **301-296**

מעגל עבור circuit for **258-257 ,255-253** (תרגילים)

מקוטע segmented **307** (בעיה)

על מערך on an array **296** (תרגיל)

על רשימה on a list **282-280**

שימוש בשיטת מסלול־אוילר used in Euler-tour technique **283**

חישוב רֵישׁוֹת מקבילי מקוטע segmented parallel prefix **307** (בעיה)

חיתוך intersection

מציאת כולם, בקבוצה של קטעי ישרים finding all, in a set of line segments **464** (תרגיל)

קביעה, עבור קבוצת קטעי ישרים determining, for a set of line segments **464-459**

קביעה, עבור שני מצולעים פשוטים determining, for two simple polygons **464** (תרגיל)

קביעה, עבור שני עיגולים determining, for two disks **464** (תרגיל)

קביעה, עבור שני קטעי ישרים determining, for two line segments **458-457**

של שפות of languages **484**

חסם bound

יחס ratio **521**

שגיאה יחסית relative error **522**

חסם יחס ratio bound **521**

חסם על המעלה degree-bound **356**

חסם שגיאה יחסית relative error bound **522**

חציית חתך crossing a cut **110**

חתימה דיגיטלית digital signature **408**

חתך cut

של גרף בלתי מכוון of an undirected graph **110**

של רשת זרימה of a flow network **191-189**

ט

טאוטולוגיה tautology **491** (תרגיל), **505** (תרגיל)

טבלת אמת truth table **245**

טקסט בהתאמת מחרוזת text in string matching **426**

י

יוצר generator

של תת־חבורה of a subgroup **395**

של Z_n^* of Z_n^* **403**

ייצוג בעזרת מקדמים coefficient representation **358**

וכפל מהיר and fast multiplication **362-361**

ייצוג על־ידי מטריצת סמיכויות adjacency-matrix representation **81**

ייצוג על־ידי ערכי נקודות point-value representation **359-358**

ייצוג על־ידי רשימות סמיכות adjacency-list representation **80**

יער forest

מציאת שורשים של, על CREW PRAM finding roots of, on a CREW PRAM **288-286**

מציאת שורשים של, על EREW PRAM finding roots of, on an EREW PRAM **292** (תרגיל)

עומק depth-first **91**

קבוצות זרות disjoint-set **65-62**

יער עומק depth-first forest **91**

יער קבוצות זרות disjoint-set forest **65-62**

ניתוח של analysis of **72-69**

תכונות הדרגות של rank properties of **69-68**

ראה גם: מבנה נתונים של קבוצות זרות *see also* disjoint-set data structure

יציבות של ערכים במעגל stability of circuit values **246**

יצירת נשא carry generate **251**

יצירת סיבית נשא generate a carry bit **251**

כ

כוכב קליני (*) Kleene star **484**

כופל מערך־לינארי linear-array multiplier **271-268**

כופל מערכי array multiplier **262-259**

כופל עץ־וואלאס Wallace-tree multiplier **264-262**

כיווץ מסלולים path compression **63**

כיסוי cover

מסלול path **219** (בעיה)

על־ידי קדקודים vertex **508, 523**

על־ידי תת־קבוצה by a subset **530**

כיסוי אופטימלי על־ידי קדקודים optimal vertex cover **523**

כיסוי מינימלי על־ידי מסלולים minimum path cover **219** (בעיה)

כיסוי על־ידי מסלולים path cover **219** (בעיה)

כיסוי על־ידי קדקודים vertex cover **508, 523**

עבור עץ for a tree **525** (תרגיל)

כלל הורנר Horner's rule **358**

כפולה multiple **314, 380**

של איבר מודולו n of an element modulo n **396-399**

כפולה משותפת מינימלית least common multiple (lcm) **390** (תרגיל)

כפולה סקלרית scalar multiple **314**

כפל multiplication

אלגוריתם בית הספר היסודי עבור grade-school algorithm for **259**

אלגוריתם האיכר הרוסי עבור Russian peasant's algorithm for **268**

כופל מערכי array multiplier **259-262**

כופל מערך־לינארי linear-array multiplier **268-271**

כופל עץ וואלאס Wallace-tree multiplier **262-264**

מודולו n modulo n $(\cdot_n)$ **391**

מטריצות of matrices **315**

מעגלים עבור circuits for **259-265, 268-271**

פולינומים of polynomials **357**

שיטת הפרד ומשול עבור divide-and-conquer method for **375** (בעיה)

של מספרים מרוכבים of complex numbers **326** (תרגיל)

כפל מטריצה בווקטור matrix-vector multiplication **296** (תרגיל)

כפל מטריצות matrix multiplication

באמצעות אלגוריתם CRCW יעיל בעבודה by a work-efficient CRCW algorithm **296** (תרגיל)

בוליאניות boolean **328-329, 330** (תרגיל), **346** (תרגיל), **352** (בעיה)

האלגוריתם של שטראסן Strassen's algorithm **320-326**

והיפוך מטריצות and matrix inversion **343-345**

וחישוב הדטרמיננטה and computing the determinant **346** (תרגיל)

ופירוק LUP and LUP decomposition **346** (תרגיל)

עבור מסלולים קצרים ביותר בין כל הזוגות for all-pairs shortest paths **154-159**

על CRCW PRAM on a CRCW PRAM **292** (תרגיל)

על EREW PRAM on an EREW PRAM **292** (תרגיל)

שיטת פאן Pan's method **325** (תרגיל)

כפל מטריצות בוליאניות boolean matrix multiplication

וסגור טרנזיטיבי and transitive closure **330** (תרגילים), **346** (תרגיל)

תוך שימוש באלגוריתם של שטראסן using Strassen's algorithm **328-329, 352** (בעיה)

תוך שימוש באקראיות using randomization **352** (בעיה)

כתב צופן ciphertext **407**

כתיבה בו־זמנית concurrent-write **275**

כתיבה־בלעדית exclusive-write **275**

ל

לא ניתן לפתרון סביר intractable **480**

לוגריתם בדיד discrete logarithm **403**

לוח זמנים של נקודות אירוע event-point schedule **461**

ליטרל literal **502**

למת ההפעלה החוזרת של פונקציית הרישא prefix-function iteration lemma **443**

למת המחצית halving lemma **365**

למת הסיפות החופפות overlapping-suffix lemma **427**

למת הסכום summation lemma **365**

למת הצמצום cancellation lemma **364**

למת הרקורסיה של פונקציית הסיפא suffix-function recursion lemma **438**

למת משלים־שור Schur complement lemma **348**

מ

מאפֵס annihilator **170, 326**

מבחן הראשוניות של מילר ורבין Miller-Rabin primality test **414-418**

מבחן ראשוניות primality testing **411-418, 424-425**

בדיקת פסאודו־ראשוניות pseudoprimality testing **413**

מבחן מילר־רבין Miller-Rabin test **414-418**

מבנה נתונים data structure **3-75**

מתמיד persistent **4**

עבור קבוצות זרות for disjoint sets **56-75**

על התקן אחסון משני on secondary storage **6-7**

עצי־B B-trees **5-20**

עצי splay splay trees **4**

עצי 2-3 2-3 trees **20**

עצים דינמיים dynamic trees **4**

ערימות בינומיות binomial heaps **37-21**

ערימות ניוחות relaxed heaps **55**

ערימות פיבונצ'י Fibonacci heaps **55-38**

ערימות 2-3-4 2-3-4 heaps **36** (בעיה)

van Emde Boas **4**

ראה גם: קבוצה דינמית *see also* dynamic set

מבנה נתונים לקבוצות זרות disjoint-set data structure **75-56**

באלגוריתם של קרוסקל in Kruskal's algorithm **113**

בבעיית המינימום הבלתי־מקוונת in off-line minimum **73** (בעיה)

בבעיה הבלתי־מקוונת של האבות הקדמונים העמוקים ביותר

in off-line least common ancestors **74** (בעיה)

בקביעת עומק in depth determination **73** (בעיה)

מימוש באמצעות יער קבוצות זרות disjoint-set-forest implementation of **65-62**

מימוש על־ידי רשימה מקושרת linked-list implementation of **61-58**

מקרה פרטי שמתבצע בזמן לינארי linear-time special case of **75**

ניתוח של analysis of **72-69**

מבנה נתונים מתמיד persistent data structure **4**

מבנה נתונים van Emde Boas van Emde Boas data structure **4**

מדד סיבוכיות complexity measure **485**

מהפך inverter **245**

מודל CRCW־משותף common-CRCW model **276**

מוחלף transpose

צמוד conjugate **346** (תרגיל)

של גרף מכוון of a directed graph **82** (תרגיל)

של מטריצה of a matrix **312 ,81**

מוחלפת צמודה conjugate transpose **346** (תרגיל)

מולטי־גרף multigraph

המרה לגרף בלתי־מכוון שקול converting to equivalent undirected graph **83** (תרגיל)

מונה בינרי binary counter

וערימות בינומיות and binomial heaps **36** (תרגיל)

מונואיד monoid **326 ,170**

מופע של בעיה מופשטת instance of an abstract problem **481**

מופע של תבנית occurrence of a pattern **426**

מחיקה deletion

ממצבי קו־סריקה from sweep-line statuses **461**

מעצי־B from B-trees **18-16**

מערימות בינומיות from binomial heaps **35**

מערימות פיבונצ'י from Fibonacci heaps **51, 54** (בעיה)

מערימות 2-3-4 from 2-3-4 heaps **36-37** (בעיה)

מחלק divisor **380**

משותף common **382**

ראה גם: מחלק משותף מקסימלי *see also* greatest common divisor

מחלק טריוויאלי trivial divisor **380**

מחלק משותף common divisor **382**

מקסימלי, *see* greatest common divisor ראה: מחלק משותף מקסימלי

מחלק משותף מקסימלי greatest common divisor (gcd) **383-382**

חישוב על־ידי האלגוריתם של אוקלידס computed by Euclid's algorithm **390-385**

חישוב על־ידי אלגוריתם gcd בינרי computed by binary gcd algorithm **423** (בעיה)

משפט הרקורסיה עבור recursion theorem for **386**

עם יותר משני ארגומנטים with more than two arguments **390** (תרגיל)

מחלקה class

של סיבוכיות complexity **485**

מחלקת סיבוכיות complexity class **485**

co-NP **489**

NP **489**

NPC **493**

P **483**

מחלקות שקילות מודולו n equivalence classes modulo n ($\mathbf{Z}_n$) **381**

מחלקת שקילות equivalence class

מודולו n modulo n ($[a]_n$) **381**

מחסנית stack

בסריקה של גראהם in Graham's scan **466**

על התקן אחסון משני on secondary storage **19** (בעיה)

מחרוזת string **426**

מטריצה matrix **319-311**

הרמיטית Hermitian **346** (תרגיל)

וחוגים and rings **328-327**

ומעין־חוגים and quasirings **327**

טפליץ Toeplitz **375** (בעיה)

מוחלפת transpose of **312 ,81**

מוחלפת צמודה conjugate transpose of **346** (תרגיל)

סימטרית חיובית־לחלוטין symmetric positive-definite **348-346**

סמיכויות adjacency **81**

פסאודו־הופכית pseudoinverse of **350**

קודמים predecessor **153**

קשירויות incidence **83** (תרגיל), **510**

ראה גם: היפוך מטריצות, כפל מטריצות *see also* matrix inversion, matrix multiplication

מטריצה אלכסונית diagonal matrix **312**

פירוק LUP של LUP decomposition of **342** (תרגיל)

מטריצה בלתי־הפיכה noninvertible matrix **316**

מטריצה הפיכה invertible matrix **316**

מטריצה הרמיטית Hermitian matrix **346** (תרגיל)

מטריצה חיובית־לחלוטין positive-definite matrix **318**

מטריצה משולשית triangular matrix **313**, **318** (תרגיל)

מטריצה משולשית עליונה upper-triangular matrix **313**

מטריצה משולשית תחתונה lower-triangular matrix **313**

מטריצה סימטרית symmetric matrix **314**, **319** (תרגילים)

מטריצה סימטרית חיובית־לחלוטין symmetric positive-definite matrix **346-348**

מטריצה סינגולרית singular matrix **316**

מטריצה רגולרית nonsingular matrix **316**

מטריצה ריבועית square matrix **312**

מטריצה תלת־אלכסונית tridiagonal matrix **313**

מטריצת אפס zero matrix **312**

מטריצת ואנדרמונדה Vandermonde matrix **319** (תרגיל)

מטריצת טפליץ Toeplitz matrix **375** (בעיה)

מטריצת יחידה identity matrix **313**

מטריצת יחידה משולשית עליונה unit upper-triangular matrix **313**

מטריצת יחידה משולשית תחתונה unit lower-triangular matrix **314**

מטריצת קודמים predecessor matrix **153**

מטריצת קשירויות incidence matrix

ואילוצי הפרשים and difference constraints **144**

של גרף בלתי־מכוון of an undirected graph **510**

של גרף מכוון of a directed graph **83** (תרגיל)

מטריצת תמורה permutation matrix **314**, **318** (תרגיל)

פירוק LUP של LUP decomposition of **342** (תרגיל)

מיון sorting

באמצעות רשתות, ראה: רשת מיון using networks, *see* sorting network

על EREW PRAM on an EREW PRAM **295**

של נקודות על־פי זוויות קוטביות of points by polar angle **458** (תרגיל)

מיון הכנסה insertion sort

מימוש על־ידי רשת מיון sorting-network implementation of **230** (תרגיל)

מיון השוואה comparison sort

וערימות מתמזגות and mergeable heaps **48** (תרגיל)

מיון טופולוגי topological sort **97-99**

בחישוב מסלולים קצרים ביותר ממקור יחיד בגמ"ל in computing single-source shortest paths on a dag **139**

מיון מקדים presorting **476**

מיון-מיזוג merge sort

מימוש על־ידי רשת מיון sorting network implementation of **239-241**

מיזוג merge

על CRCW PRAM on a CRCW PRAM **292** (תרגיל)

תוך שימוש ברשת השוואה using a comparison network **237-239**

מיזוג רשימת שורשים בערימת פיבונצ'י consolidating a Fibonacci-heap root list **44**, **48** (תרגיל)

מינור של מטריצה minor of a matrix **317**

מינימום minimum

בלתי־מקוון off-line **73** (בעיה)

בעצי־B in B-trees **15** (תרגיל)

בערימות בינומיות in binomial heaps **27**

בערימות פיבונצ'י in Fibonacci heaps **42**

מיקום ברשימה בחישוב מקבילי list ranking in parallel **278-280**

מכונת גישה־אקראית random-access machine

לעומת רשתות השוואה vs. comparison networks **227**

מקבילית parallel **274-276**, **295**, **296** (תרגילים)

מכונת גישה־אקראית מקבילית parallel random-access machine **274-277**, **295**, **296** (תרגילים)

ראה גם: CRCW, CREW, ERCW, EREW *see also*

מכפלה product

וקטורית cross **455**

חיצונית outer **315**

חלקית partial **259**

סקלרית של זרימה scalar flow **184-185** (תרגיל)

פנימית inner **315**

של מטריצות of matrices **315**

של פולינומים of polynomials **357**

מכפלה וקטורית (×) cross product **455**

מכפלה חיצונית outer product **315**

מכפלה חלקית partial product **259**

מכפלה סקלרית של זרימה scalar flow product **184-185** (תרגיל)

מכפלה פנימית inner product **315**

מנה quotient **381**

מסלול path

גילוי find **62**

המילטוני hamiltonian **491** (תרגיל)

משקל של weight of **120**

קצר ביותר shortest **86, 120**

קריטי critical **141, 258** (תרגיל)

שיפור augmenting **188**

שרשור concatentation of **171**

תווית של label of **171**

תחשיב של, בגרפים מכוונים calculus of, in directed graphs **170-173**

מסלול tour

אוילר Euler **283**

של גרף of a graph **517**

מסלול אוילר Euler tour **106** (בעיה)

מסלול גילוי find path **62**

מסלול קריטי critical path

בגרף מכוון ללא־מעגלים (גמ"ל) in a directed acyclic graph **141**

במעגל צירופי in a combinational circuit **258** (תרגיל)

מסלול שיפור augmenting path **188**

מסלולים קצרים ביותר shortest paths **120-177**

אומדן של estimate of **125**

אלגוריתם בלמן־פורד Bellman-Ford algorithm **136-139**

אלגוריתם פלויד־וורשאל Floyd-Warshall algorithm **160-163**

באמצעות העלאה חוזרת בריבוע by repeated squaring **157-158**

באמצעות כפל מטריצות by matrix multiplication **154-159**

בגרף בלתי משוקלל in an unweighted graph **86**

בגרף משוקלל in a weighted graph **120**

בגרפים $\in$־צפופים in $\in$-dense graphs **176** (בעיה)

בין כל הזוגות all-pairs **152-177**

גרסאות של הבעיה problem variants **121**

האלגוריתם המדורג של גאבו Gabow's scaling algorithm **148** (בעיה)

האלגוריתם של ג'ונסון Johnson's algorithm **166-169**

האלגוריתם של דייקסטרה Dijkstra's algorithm **131-135**

ואילוצי הפרשים and difference constraints **142-147**

והקלה and relaxation **125-128**

וחיפוש לרוחב and breadth-first search **86-88, 120**

ומעגלים בעלי משקל שלילי and negative-weight cycles **121**

ממקור יחיד single-source **120-151**

על גרף מכוון ללא מעגלים on a directed acyclic graph **139-141**

עם קשתות בעלות משקל שלילי with negative-weight edges **121-122**

עץ של tree of **123, 128-130**

תת־מבנה אופטימלי של optimal substructure of **124-125**

מסלולים קצרים ביותר ליעד יחיד single-destination shortest paths **121**

מסלול קצר ביותר בין זוג קדקודים אחד single-pair shortest path **121, 146** (תרגיל)

מסלולים קצרים ביותר ממקור יחיד single-source shortest paths **120-151**

אלגוריתם בלמן־פורד Bellman-Ford algorithm **136-139**

בגרפים $\in$־צפופים in $\in$-dense graphs **176** (בעיה)

האלגוריתם המדורג של גאבו Gabow's scaling algorithm **148** (בעיה)

האלגוריתם של דייקסטרה Dijkstra's algorithm **131-135**

ואילוצי הפרשים and difference constraints **142-147**

על גרף מכוון ללא מעגלים on a directed acyclic graph **139-141**

מספר כרומטי, *ראה:* צביעה chromatic number, *see* coloring

מספר פריק composite number **380**

מספר קארמייקל Carmichael number **413, 418** (תרגיל)

מספר ראשוני prime number **380**

צפיפות של density of **412**

מספרי פיבונצ'י Fibonacci numbers **52**

חישוב של computation of **423** (בעיה)

מספרים מרוכבים, כפל של complex numbers, multiplication of **326** (תרגיל)

מעבד אחראי responsible processor **278**

מעבד לא פעיל inactive processor **296** (תרגיל)

מעגל circuit

מבוקר־שעון clocked **265-271**

מבוקר־שעון גלובלית globally clocked **266**

מונה tally **258** (תרגיל)

סדרתי sequential **265-271**

עבור התמרת פוריה המהירה for Fast Fourier Transform **374-375**

עבור חיבור for addition **249-259, 265-267**

עבור חילוק for division **271** (בעיה)

עבור חישוב רישות מקבילי for parallel prefix **253-255**

עבור כפל for multiplication **259-265, 268-271**

צירופי combinational **244-249**

ראה גם: רשת השוואה *see also* comparison network

מעגל בגרף cycle of a graph

בעל משקל ממוצע מינימלי minimum mean-weight **149** (בעיה)

בעל משקל שלילי negative-weight *ראה:* מעגל בעל משקל שלילי *see* negative-weight cycle

גילוי של detection of **99** (תרגיל)

המילטוני hamiltonian **487**

מעגל בעל משקל ממוצע מינימלי minimum mean-weight cycle **149** (בעיה)

מעגל בעל משקל שלילי negative-weight cycle

גילוי של, על־ידי אלגוריתם למציאת מסלולים קצרים ביותר בין כל הזוגות

detection of, by all-pairs shortest-paths algorithm **159** (תרגיל)

גילוי של, על־ידי אלגוריתם פלויד־וורשאל detection of , by Floyd-Warshall algorithm **165**

ואילוצי הפרשים and difference constraints **145**

והקלה and relaxation **131** (תרגיל)

ומסלולים קצרים ביותר and shortest paths **121**

מציאה של finding **139** (תרגיל), **159** (תרגיל)

מעגל הזזה shifter **265** (תרגיל)

מעגל הזזה גלילית barrel shifter **265** (תרגיל)

מעגל הזזה מעגלית cyclic shifter **265** (תרגיל)

מעגל חילוק division circuit **271** (בעיה)

מעגל מבוקר־שעון clocked circuit **271-265**

מעגל מבוקר־שעון גלובלית globally clocked circuit **266**

מעגל מנייה tally circuit **258** (תרגיל)

מעגל צירופי combinational circuit **293 ,249-244**

הדמיה על־ידי CRCW PRAM simulated by a CRCW PRAM **296** (תרגיל)

הדמיה על־ידי CREW PRAM simulated by a CREW PRAM **294-293**

הדמיה על־ידי EREW PRAM simulated by an EREW PRAM **295-294**

ומעגל מבוקר־שעון שקול and equivalent clocked circuit **267**

ראה גם: מעגל, רשת השוואה *see also* circuit, comparison network

מעגלים אריתמטיים arithmetic circuits **273-244**

מעין־חוג quasiring **327-326**

מעין־חוג בוליאני boolean quasiring **327**

מעלה של פולינום degree of a polynomial **356**

מערכת אילוצי הפרשים system of difference constraints **147-142**

מערכת ההצפנה RSA מבוססת מפתח ציבורי RSA public-key cryptosystem **411-406**

מערכת הצפנה cryptosystem **411-406**

מערכת הצפנה מבוססת מפתח ציבורי public-key cryptosystem **411-406**

מערכת לינארית תלת־אלכסונית tridiagonal linear system **353** (בעיה)

מערכת משוואות לינאריות system of linear equations **353 ,342-330**

מערכת משוואות לינאריות מוגדרת־יתר overdetermined system of linear equations **331**

מערכת משוואות לינאריות תת־מוגדרת undetermined system of linear equations **331**

מפתח key

חציון, של צומת בעץ־B median, of a B-tree node **11**

סודי secret **409 ,406**

ציבורי public **409 ,406**

מפתח חציון של צומת בעץ־B median key of a B-tree node **11**

מפתח מינימלי minimum key

בערימות 2-3-4 in 2-3-4 heaps **36** (בעיה)

מפתח סודי secret key **409 ,406**

מפתח ציבורי public key **409 ,406**

מצב התחלתי start state **435**

מצב מקבל accepting state **435**

מצב נשא carry status **251**

מצב קו סריקה sweep-line status **461**

מצב של אוטומט סופי state of a finite automaton **435**

מצולע polygon

בדיקת פשטות simplicity testing **464** (תרגיל)

בצורת כוכב star-shaped **473** (תרגיל)

גרעין של kernel of **473** (תרגיל)

חישוב שטח של computing area of **459** (תרגיל)

קביעה אם שני מצולעים פשוטים נחתכים determining whether two simple ones intersect **464** (תרגיל)

מצולע בעל צורת כוכב star-shaped polygon **473** (תרגיל)

מצולע פשוט simple polygon **464** (תרגיל)

מציאת מקסימום finding the maximum **292 ,289-288** (תרגיל), **308** (בעיה)

מציאת ערכו של פולינום evaluation of a polynomial **362 ,358** (תרגיל)

בנקודות מרובות at multiple points **376** (בעיה)

ונגזרותיו and its derivatives **376** (בעיה)

מקדם coefficient

של פולינום of a polynomial **356**

מקור source **183 ,182-179 ,121 ,83**

מקור־על supersource **183**

מקסימלי maximal

נקודה point **478** (בעיה)

שכבות layers **478** (בעיה)

מרחק distance

אוקלידי euclidean **473**

מנהטן Manhattan **477** (תרגיל)

של מסלול קצר ביותר of a shortest path **86**

L_m **477** (תרגיל)

מרחק אוקלידי euclidean distance **473**

מרחק מנהטן Manhattan distance **477** (תרגיל)

מרחק־L_m L_m-distance **477** (תרגיל)

משוואה נורמלית normal equation **350**

משוואות לינאריות linear equations

פתרון מערכת של solving systems of **342-330**

פתרון מערכת תלת־אלכסונית של solving tridiagonal systems of **353** (בעיה)

פתרון משוואות מודולריות solving modular **399-396**

משוואות לינאריות מודולריות modular linear equations **399-396**

משוואת הפרשים, ראה: נוסחת נסיגה difference equation, *see* recurrence

משווה comparator **227**, **230** (תרגילים), **239** (תרגילים), **240** (תרגילים)

משולש triangle

בגרף בלתי־מכוון in an undirected graph **330** (תרגיל)

משלים cofactor **317**

משלים complement

שור Schur **336**, **347**

של גרף of a graph **508**

של שפה of a language **484**

משלים־שור Schur complement **336**, **347**

משפט אוילר Euler's theorem **403**, **418** (תרגיל)

משפט ברנט Brent's theorem **293-294**, **296** (תרגיל)

משפט הול Hall's theorem **200** (תרגיל)

משפט החילוק division theorem **381**

משפט הלוגריתם הבדיד discrete logarithm theorem **403**

משפט המסלול הלבן white-path theorem **94**

משפט המספרים הראשוניים prime number theorem **412**

משפט הסוגריים parenthesis theorem **93**

משפט הפתרון בשלמים integrality theorem **199**

משפט הקונבולוציה convolution theorem **369**

משפט השאריות הסיני Chinese remainder theorem **399-402**

משפט לאגראנז' Lagrange's theorem **394**

משפט לאמה Lamé's theorem **388**

משפט פרמה Fermat's theorem **403**

משקל weight

מינימקסימלי minimax **177** (בעיה)

ממוצע mean **149** (בעיה)

של מסלול of a path **120**

של קשת of an edge **81**

משקל מינימקסימלי minimax weight **177** (בעיה)

משקל ממוצע של מעגל mean weight of a cycle **149** (בעיה)

נ

נוסחה בוליאנית boolean formula **491** (תרגיל), **500**, **505** (תרגילים)

נוסחה ספיקה satisfiable formula **500**

נוסחת לאגראנז' Lagrange's formula **360**

נורמה אוקלידית euclidean norm **315-316**

נורמה של וקטור norm of a vector **315-316**

ניסוי ברנולי Bernoulli trial **300**

ניסיון חילוק trial division **412**

ניתוח אלגוריתמים analysis of algorithms

ראה גם: ניתוח הסתברותי *see also* probabilistic analysis

ניתוח הסתברותי probabilistic analysis

של אלגוריתם מקבילי יעיל־בעבודה לחישוב רישות

of work-efficient parallel prefix algorithm **299-301**, **301** (תרגילים)

של אלגוריתם רבין־קארפ of Rabin-Karp algorithm **433**

של היוריסטיקת רו של פולארד of Pollard's rho heuristic **420-422**, **422** (תרגילים)

של השוואת קבצים of file comparison **434** (תרגיל)

של כפל מטריצות בוליאניות of boolean matrix multiplication **352** (בעיה)

של מבחן הראשוניות של מילר ורבין of Miller-Rabin primality test **415-418**

של קידום נשא בסוכם מגלגל־נשא of carry propagation in a ripple-carry adder **259** (תרגיל)

של קמור מעל התפלגות בעלת קמור דליל of convex hull over a sparse-hulled distribution **479** (בעיה)

ניתוח לשיעורין amortized analysis

עבור אלגוריתם CRCW למציאת רכיבים קשירים for CRCW connected-components algorithm **308** (בעיה)

עבור אלגוריתם קנות־מוריס־פראט for Knuth-Morris-Pratt algorithm **442**

עבור האלגוריתם הכללי לדחיפת קדם־זרימה for generic preflow-push algorithm **208**

עבור הסריקה של גראהם for Graham's scan **470-471**

עבור מבני נתונים של קבוצות זרות for disjoint-set data structures **60**, **61** (תרגיל), **65** (תרגילים), **69-72**

עבור מחסניות על התקן אחסון משני for stacks on secondary storage **19** (בעיה)

עבור ערימות פיבונצ'י for Fibonacci heaps **40-43**, **47**, **50-51**, **51** (תרגיל)

ניתוק שרשרת cascading cut **49**, **54** (תרגיל)

ניתוק, בערימת פיבונצ'י cutting. in a Fibonacci heap **49**

ניתן לפתרון סביר tractable **480**

נציג של קבוצה representative of a set **56**

נקודה point

בגיאומטריה חישובית in computational geometry **454-455**

קיצון extreme **142** (הערה)

נקודת הפרדה articulation point **105** (בעיה)

נקודת הראשית origin **455**

נקודת קיצון extreme point **142** (הערה)

נקודת קצה endpoint
של קטע ישר of a line segment **455**
נקודת קשר, של פונקציית ספליין knot, of a spline **353-354** (בעיה)

ס

סביבה neighborhood **200** (תרגיל)
סגור closure
אופרטור (*) operator **173**
טרנזיטיבי, ראה: סגור טרנזיטיבי transitive, *see* transitive closure
של שפה of a language **484**
תחת אופרטור under an operator **170, 326**
תכונה של קבוצה group property **391**
סגור טרנזיטיבי transitive closure **163-164, 330** (תרגילים)
וכפל מטריצות בוליאניות and boolean matrix multiplication **330** (תרגיל), **346** (תרגיל)
של גרפים דינמיים of dynamic graphs **176** (בעיה)
סדר של חבורה order of a group **395**
סדר הערימה heap order **24**
סדרה sequence (⟨ ⟩) **228, 234**
סדרה ביטונית bitonic sequence **234**
סדרה נקייה clean sequence **234**
סדרת פלט output sequence **228**
סדרת קלט input sequence **228**
סוכם מלא full adder **246-247**
סיבית הסכום sum bit **250**
סיבית נשא־יוצא carry-out bit **250**
סיבית נשא־נכנס carry-in bit **250**
סיווג קשתות classification of edges
בחיפוש לעומק in depth-first search **95-96**
בחיפוש לרוחב in breadth-first search **105** (בעיה)
סיווג קשתות בחיפוש לעומק classification of edges in depth-first search **96-97** (תרגילים)
סיכום של תוויות מסלולים summary of path labels **171**
סימון הסיכום המובלע implicit summation notation **183**
סימטריה נגדית skew symmetry **179**
סימן לז'אנדר Legendre symbol $\left(\frac{a}{p}\right)$ **424** (בעיה)
סימפלקס simplex **142** (הערה)
סיפא suffix (>) **427**
סכום sum
זרימות flow **184** (תרגיל)

מטריצות of matrices **314**
פולינומים of polynomials **356**
קרטזי Cartesian **363** (תרגיל)
סכום קרטזי Cartesian sum **363** (תרגיל)
סכמת קירוב approximation scheme **522-523**
סכמת קירוב פולינומיאלית לחלוטין fully polynomial-time approximation scheme **522**
עבור בעיית הקליקה המקסימלית for the maximum clique problem **538** (בעיה)
עבור בעיית סכום התת־קבוצה for the subset-sum problem **533-537**
ספיקות satisfiability **494, 500-502**
ספיקות מעגלים circuit satisfiability **494-498**
ספיקות נוסחאות 2-CNF 2-CNF satisfiability **505** (תרגיל)
ספיקות נוסחאות 3-CNF 3-CNF satisfiability **502-505**
סריקה sweeping **459-464, 478** (בעיה)
סריקה מלאה של עץ full walk of a tree **527**
סריקה סיבובית rotational sweep **465-472**

ע

עבודה work **280, 296** (תרגיל)
עד לפריקותו של מספר witness to the compositeness of a number **414**
עומק depth
של מעגל צירופי of a combinational circuit **248, 293**
של רכיב צירופי of a combinational element **248, 293**
של רשת השוואה of a comparison network **229**
של רשת מיון of a sorting network **230** (תרגיל)
של SORTER of SORTER **240** (תרגיל)
עטיפת חבילות package wrapping **471**
עטיפת מתנות gift wrapping **471**
עיגול disk **464** (תרגיל)
עץ tree
בינומי binomial **22-24, 41**
דינמי dynamic **4**
כוכב star **308** (בעיה)
כיסוי אופטימלי על־ידי קדקודים של optimal vertex cover of **525** (תרגיל)
מסלולים קצרים ביותר shortest-paths **123, 128-130**
מצביעים pointer **308** (בעיה)
מציאת השורש של, על CREW PRAM finding the root of, on a CREW PRAM **291** (תרגיל)
סריקה מלאה של full walk of **527**
עומק depth-first **91**

פורש spanning **108**

פורש מינימלי minimum spanning **119-108**

פיסוק parse **503**

קוטר של diameter of **90** (תרגיל)

רוחב breadth-first **83, 89**

splay **4**

2-3 **20**

2-3-4 **8**

עץ אדום־שחור red-black tree

בקביעה אם ישנם קטעי ישרים נחתכים in determining whether any line segments intersect **461**

השוואה לעצי B־ comparison with B-trees **9**

ועצי 2-3-4 and 2-3-4 trees **10** (תרגיל)

עץ בינומי binomial tree **22-24**

לא סדור unordered **41**

עץ בינומי לא־סדור unordered binomial tree **41**

עץ בינרי binary tree

מציאת השורש של, על CREW PRAM finding the root of, on a CREW PRAM **291** (תרגיל)

עץ דינמי dynamic tree **4**

עץ חיפוש, ראה: עץ חיפוש מאוזן, עץ חיפוש בינארי, עץ B־, עץ אדום־שחור, עץ splay, עץ 2-3, עץ 2-3-4

search tree, *see* balanced search tree, binary search tree, B-tree, red-black tree, splay tree, 2-3 tree, 2-3-4 tree

עץ חיפוש לרוחב breadth-first tree **83-84, 88-89**

עץ חיפוש מאוזן balanced search tree

עץ כוכב star tree **308** (בעיה)

עץ מצביעים pointer tree **308** (בעיה)

עץ עומק depth-first tree **91**

עץ פורש spanning tree **108**

עץ פורש בעל עלות מינימלית, ראה: עץ פורש מינימלי minimum-cost spanning tree, *see* minimum spanning tree

עץ פורש מינימלי minimum spanning tree **108-119**

אלגוריתם כללי generic algorithm **109-112**

במציאת מסלול כמעט אופטימלי לבעיית הסוכן הנוסע in finding a near-optimal traveling-salesman tour **525**

בנייה באמצעות ערימות מתמזגות constructed using mergeable heaps **37** (בעיה)

האלגוריתם של פרים Prim's algorithm **115-117**

האלגוריתם של קרוסקל Kruskal's algorithm **113-115**

כחוג־למחצה סגור as a closed semiring **177** (בעיה)

על גרפים דינמיים on dynamic graphs **118** (תרגילים)

שני הכי טוב second-best **118** (בעיה)

עץ פורש מינימלי שני הכי טוב second-best minimum spanning tree **118** (בעיה)

עץ פיסוק parse tree **503**

עץ־B B-tree **5-20**

גובה של height of **9**

דרגה מינימלית של minimum degree of **8**

הכנסה לתוך insertion into **13-15**

חיפוש ב־ searching **10-11**

יצירה creating **11**

מחיקה מתוך deletion from **16-18**

מפתח מינימלי ב־ minimum key of **15** (תרגיל)

עצי 2-3-4 2-3-4 trees **8**

פיצול צומת ב־ splitting a node in **11-13**

צומת מלא בתוך full node in **8**

תכונות של properties of **8-10**

עץ splay splay tree **4**

עץ 2-3 2-3 tree **20**

עץ 2-3-4 2-3-4 tree **8**

איחוד joining **19** (בעיה)

ועצים אדומים־שחורים and red-black trees **10** (תרגיל)

פיצול splitting **19** (בעיה)

עקרון ה־0-1 zero-one principle **231-233, 237** (תרגיל), **238** (תרגיל)

ערימה *d*־ית *d*-ary heap

באלגוריתמי מסלולים קצרים ביותר in shortest-paths algorithms **176** (בעיה)

ערימה heap

d־ית *d*-ary **176** (בעיה)

באלגוריתם של ג'ונסון in Johnson's algorithm **169**

באלגוריתם של דייקסטרה in Dijkstra's algorithm **134**

באלגוריתם של פרים in Prim's algorithm **117**

זמני הריצה של פעולות על running times of operations on **22** (איור)

למימוש ערימה מתמזגת to implement a mergeable heap **21**

מתמזגת mergeable **21**

נינוחה relaxed **55**

2-3-4 **36** (בעיה)

ערימה בינומית binomial heap **21-37**

איחוד uniting, **27-32**

הכנסה לתוך insertion into **32**

המפתח המינימלי של minimum key of **27**

הקטנת ערך מפתח ב־ decreasinig a key in **33-35**

ומונה בינרי וחיבור בינרי and binary counter and binary addition **36** (תרגיל)

זמני ריצה של פעולות על running times of operations on **22** (איור)

יצירה של creating **26**

מחיקה מתוך deletion from **35**

שליפת המפתח המינימלי מתוך extracting the minimum key from **33-32**

תכונות של properties of **24**

ערימה מתמזגת mergeable heap **21**

באלגוריתם למציאת עץ פורש מינימלי in minimum-spanning-tree algorithm **37** (בעיה)

זמני ריצה של פעולות על running times of operations on **22** (איור)

ערימות נינוחות relaxed heaps **55**

ערימות 2-3-4 2-3-4 heaps **36** (בעיה)

ראה גם: ערימה בינומית, ערימת פיבונצ'י *see also* binomial heap, Fibonacci heap

ערימה נינוחה relaxed heap **55**

ערימת פיבונצ'י Fibonacci heap **55-38**

איחוד uniting **43**

באלגוריתם של ג'ונסון in Johnson's algorithm **169**

באלגוריתם של דייקסטרה in Dijkstra's algorithm **134**

באלגוריתם של פרים in Prim's algorithm **117**

גיזום pruning **55** (בעיה)

דרגה מקסימלית של maximum degree of **40, 52-54**

הכנסה לתוך insertion into **41-42**

הקטנת ערכו של מפתח ב־ decreasing a key in **48-51**

זמני הריצה של פעולות על running times of operations on **22** (איור)

יצירה של creating **41**

מחיקה מתוך deletion from **51, 54** (בעיה)

מפתח מינימלי של minimum key of **42**

פונקציית פוטנציאל עבור potential function for **40**

שינוי ערך מפתח ב־ changing a key in **55** (בעיה)

שליפת המפתח המינימלי מתוך extracting the minimum key from **43-47**

ערימת 2-3-4 2-3-4 heap **36** (בעיה)

ערך של זרימה value of a flow **179-180**

פ

פגיעה מדומה spurious hit **431**

פולינום polynomial **356**

אינטרפולציה על־ידי interpolation by **359**

חיבור של addition of **356**

ייצוג על־ידי מקדמים coefficient representation of **358**

ייצוג על־ידי ערכי נקודות point-value representation of **358-359**

כפל של multiplication of **357, 361-362, 375** (בעיה)

מציאת ערך של evaluation of **358, 362** (תרגיל), **376** (בעיה)

פונקציה סימטרית symmetric function **272** (בעיה)

פונקציית XOR ("או־מוציא") (⊕) XOR (exclusive-or) function **245**

פונקציית OR (∨) OR function **245, 327**

על CRCW PRAM on a CRCW PRAM **292** (תרגיל)

פונקציית אקרמן Ackermann's function **65-67**

פונקציית אקרמן ההופכית (α) inverse Ackermann's function **67**

פונקציית גובה, באלגוריתמי דחיפת־קדם־ורימה height function, in preflow-push algorithms **202**

פונקציית גיבוב חד־כיוונית one-way hash function **410**

פונקציית ההתפלגות של הראשוניים prime distribution function **412**

פונקציית הזוגיות parity function **246**

פונקציית הלוגריתם הבדיד (log) discrete logarithm function **403**

פונקציית הלוגריתם החוזר iterated logarithm function **65-66**

פונקציית המופע האחרון last-occurrence function **450**

פונקציית הסיפא suffix function **436**

פונקציית הסיפא־הטובה good-suffix function **450**

פונקציית הרוב majority function **246, 272** (בעיה)

פונקציית השלילה (¬) NOT function **245**

פונקציית מטרה objective function **142, 147** (תרגיל)

פונקציית מעברים transition function **435, 439, 446** (תרגיל)

פונקציית מצבים סופיים final-state function **435**

פונקציית משקל עבור גרף weight function for a graph **81**

פונקציית ספליין spline **353** (בעיה)

פונקציית ספליין ממעלה שלישית cubic spline **353** (בעיה)

פונקציית פי phi function **393**

פונקציית פי של אוילר Euler's phi function **393**

פונקציית רדוקציה reduction function **492**

פונקציית רישא prefix function **440-442**

פונקציית שיוך תוויות בתחשיב מסלולים labeling function in path calculus **170**

פילוג, דיסטריבוטיביות distributivity **170, 327**

פיצול splitting

של עצי 2-3-4 of 2-3-4 trees **19** (בעיה)

של צמתים בעץ־B of B-tree nodes **11-13**

פירוק LU LU decomposition **335-338**

פירוק LUP LUP decomposition **332**

בהיפוך מטריצות in matrix inversion **343**

השימוש ב־ use of **332-335**

וכפל מטריצות and matrix multiplication **346** (תרגיל)

חישוב computation of **341-338**

של מטריצה אלכסונית of a diagonal matrix **342** (תרגיל)

של מטריצת תמורה of a permutation matrix **342** (תרגיל)

פירוק יחיד לגורמים של מספרים שלמים unique factorization of integers **384**

פירוק לגורמים factorization **425 ,422-418**

יחיד unique **384**

פירוק לרכיבים קשירים היטב decomposition into strongly connected components **105-100**

פלט output

של מעגל צירופי of a combinational circuit **246**

של שער לוגי of a logic gate **245**

פסאודו־הופכי pseudoinverse **350**

פסאודו־ראשוני pseudoprime **413**

פעולה אסוציאטיבית associative operation **391 ,326 ,170**

פעולה של מעגל צירופי מקבילי parallel combinational circuit operation **244**

פעולת MAKE-SET MAKE-SET **56**

מימוש על־ידי יער קבוצות זרות of disjoint-set-forest implementation **63**

מימוש על־ידי רשימה מקושרת of linked-list implementation **61 ,59** (תרגילים)

פעולת FIND-SET FIND-SET **57**

מימוש על־ידי יער קבוצות זרות of disjoint-set-forest implementation **64, 65-64** (תרגילים), **75**

מימוש על־ידי רשימות מקושרות of linked-list implementation **61 ,59** (תרגילים)

פעולת דחיפה (באלגוריתמי דחיפת־קדם־זרימה) push operation (in preflow-push algorithms) **203**

פעולת הרמה lift operation **207 ,203**

פעולת פרפר butterfly operation **371**

פעימת שעון tick of a clock **266**

פריקה של קדקוד גולש discharge of an overflowing vertex **211**

פרישת יציאה fan-out **293 ,246**

פרישת כניסה fan-in **293 ,247**

פתרון solution

אפשרי feasible **142**

לבעיה מופשטת to an abstract problem **481**

לבעיה קונקרטית to a concrete problem **482**

למערכת משוואות לינאריות to a system of linear equations **331**

פתרון אפשרי feasible solution **142**

צ

צביעה coloring **519 ,302** (בעיה)

על EREW PRAM on an EREW PRAM **305-303**

צומת המינימום בערימת פיבונצ'י minimum node of a Fibonacci heap **40**

צומת מלא full node **8**

צומת מסומן marked node **40, 49-51, 51** (תרגיל)

צורה דיסיונקטיבית נורמלית (DNF) disjunctive normal form **504**

צורה קוניונקטיבית נורמלית (CNF) conjunctive normal form **502**

צורה תלת־קוניונקטיבית נורמלית 3-conjunctive normal form **502**

ציר pivot **337**

צירוף קמור של נקודות convex combination of points **454**

צל של נקודה shadow of a point **473** (תרגיל)

צפיפות המספרים הראשוניים density of prime numbers **412**

ק

קבוצה k־שליטה k-ruling set **307** (תרגיל)

קבוצה ({ }) set

בלתי־תלויה independent **519 ,302** (בעיה)

שליטה ruling **307** (תרגיל)

קבוצה בלתי־תלויה independent set **519 ,302** (בעיה)

קבוצה בלתי־תלויה מקסימלית (MIS) maximal independent set **302**

חישוב מתוך צביעת־6 על EREW PRAM computed from a 6-coloring on an EREW PRAM **306**

קבוצה דינמית dynamic set

ערימות בינומיות binomial heaps **37-21**

מבנה נתונים לקבוצות זרות disjoint-set data structure **75-56**

מבנה נתונים של van Emde Boas van Emde Boas data structure **4**

מתמידה persistent **4**

עצי־B B-trees **20-5**

עצי splay splay trees **4**

עצי 2-3 2-3 trees **20**

ערימות נינוחות relaxed heaps **55**

ערימות פיבונצ'י Fibonacci heaps **55-38**

ערימות 2-3-4 2-3-4 heaps **36** (בעיה)

קבוצה שליטה ruling set **307** (תרגיל)

קבלה acceptance

על־ידי אוטומט סופי by a finite automaton **435**

על־ידי אלגוריתם by an algorithm **485**

קדם־זרימה preflow **201**

קדקוד vertex

ביניים intermediate **160**

נקודת הפרדה articulation point **105** (בעיה)

קדקוד אפור gray vertex **91 ,84**

קדקוד ביניים intermediate vertex 160
קדקוד בלתי־מזווג unmatched vertex 197
קדקוד גולש overflowing vertex 201
קדקוד יעד destination vertex 121
קדקוד לבן white vertex 84, 91
קדקוד מורם lifted vertex 203
קדקוד מזווג matched vertex 197
קדקוד שהסתיים finished vertex 91
קדקוד שהתגלה discovered vertex 84, 91
קדקוד שחור black vertex 84, 91
קו סריקה sweep line 459
קודם predecessor
בעצי חיפוש לרוחב in breadth-first trees 84
בעצי מסלולים קצרים ביותר in shortest-paths trees 122
בעצי B‑ in B-trees 15 (תרגיל)
קוטר של עץ diameter of a tree 90 (תרגיל)
קומוטטיביות תחת אופרטור commutativity under an operator 170, 326, 329, 391
קונבולוציה (⊗) convolution 358
קונסיסטנטיות של ליטרלים consistency of literals 507
קופסאות מקוננות nesting boxes 148 (בעיה)
קופסה חוסמת bounding box 457
קופסת KPG KPG box 255
קטע ישר line segment 455
מציאת כל החיתוכים, בקבוצה finding all intersections of, in a set 464 (תרגיל)
קביעה אם קיימים קטעים נחתכים determining whether any intersect 459-464
קביעה אם שניים נחתכים determining whether two intersect 457-458
קביעת כיוון הפנייה של determining turn of 456
קטע מכוון directed segment 455-456
קטע, ראה: קטע מכוון, קטע ישר segment, *see* directed segment, line segment
קטעים בני־השוואה comparable line segments 460
קיבול capacity
שיורי residual 186, 188
של חתך of a cut 189
של קשת of an edge 179
קיבול שיורי residual capacity 186, 188
קידוד של מופעי בעיה encoding of problem instances 482-484
קידום נשא carry propagate 251
קיזוז זרימה cancellation of flow 181-182

קירוב על־ידי ריבועים פחותים least-squares approximation 348-351
קישור link
של עצים בינומיים of binomial trees 22
של שורשים בערימת פיבונאצ'י of Fibonacci-heap roots 44
קלט input
למעגל צירופי to a combinational circuit 246
לשער לוגי to a logic gate 245
קליקה clique 506
קמור convex hull 464-473, 479 (בעיה)
קריאה בו־זמנית concurrent-read 275
קריאה בו־זמנית, כתיבה בו־זמנית CRCW (concurrent-read, concurrent-write) 276
בהשוואה ל־EREW compared to EREW 285-292
הדמיה על־ידי EREW simulated by EREW 290-291
הדמיה של מעגל צירופי simulation of a combinational circuit 296 (תרגיל)
הדמיה של, על־ידי EREW simulation of, by EREW 292 (תרגילים), 296 (תרגיל)
כפל מטריצות matrix multiplication 292 (תרגיל)
כפל מטריצות יעיל־בעבודה work-efficient matrix multiplication 296 (תרגיל)
מיזוג שני מערכים ממוינים merging two sorted arrays 292 (תרגיל)
מציאת המקסימום finding the maximum 288-289, 292 (תרגיל), 308 (בעיה)
מציאת רכיבים קשירים finding connected components 308 (בעיה)
משותף common 276
פונקציית AND AND function 289
פונקציית OR OR function 292 (תרגיל)
קריאה בו־זמנית, כתיבה בלעדית CREW (concurrent-read, exclusive-write) 276
הדמיה של מעגל צירופי simulation of a combinational circuit 293-294
התאמת מחרוזות string matching 453 (בעיה)
מציאת השורש של עץ בינרי finding the the root of a binary tree 291 (תרגיל)
מציאת שורשים ביער finding roots in a forest 286-288
קריאה־בלעדית exclusive-read 275
קריאה בלעדית, כתיבה בו־זמנית ERCW (exclusive-read concurrent-write) 276
קריאה בלעדית, כתיבה בלעדית EREW (exclusive-read, exclusive-write) 276
בהשוואה ל־CRCW compared to CRCW 285-292
הדמיה של CRCW simulation of CRCW 290-291, 292 (תרגילים), 296 (תרגיל)
הדמיה של מעגל צירופי simulation of a combinational circuit 294-295
הקפצת מצביעים pointer jumping 277-285
חישוב קבוצה בלתי־תלויה מקסימלית מצביעת־6
computing a maximal independent set from a 6-coloring 306
חישוב רישות מקבילי parallel prefix 280-282, 296-301

חישוב רישות מקבילי מקוטע segmented parallel prefix **307-308** (בעיה)
כפל מטריצה בווקטור matrix-vector multiplication **296** (תרגיל)
כפל מטריצות matrix multiplication **292** (תרגיל)
מיון sorting **295**
מיקום איברים ברשימה list ranking **278-280**
מציאת שורשים ביער finding roots in a forest **292** (תרגיל)
פונקציית AND AND function **289**
צביעה coloring **303-305**
שבירת סימטריה באופן דטרמיניסטי deterministic symmetry breaking **301-307**
שיטת מסלול-אוילר Euler-tour technique **282-284**
קשורים פולינומיאלית polynomially related **483**
קשירות קשתות edge connectivity **196** (תרגיל)
קשר connective **500**
קשר בוליאני boolean connective **500**
קשת edge
אחורה back **95**
בטוחה safe **109**
בלתי מותרת inadmissible **210**
בעלת משקל שלילי negative-weight **121-122**
גשר bridge **105** (בעיה)
חוצה cross **95**
מותרת admissible **210**
משקל של weight of **81**
סיווג בחיפוש לעומק classification in depth-first search **95-96**
סיווג בחיפוש לרוחב classification in breadth-first search **105** (בעיה)
עץ tree **89, 91, 95**
קדימה forward **95**
קיבול של capacity of **179**
קלה light **110**
קריטית critical **195**
רוויה saturated **203**
שיורית residual **186**
תווית של label of **170**
קשת אחורה back edge **95, 98**
קשת בטוחה safe edge **109**
קשת בלתי־מותרת inadmissible edge **210**
קשת בעלת משקל שלילי negative-weight edge
באלגוריתם של דייקסטרה in Dijkstra's algorithm **135** (תרגיל)

ומסלולים קצרים ביותר and shortest paths **121-122**
קשת חוצה cross edge **95**
קשת מותרת admissible edge **210**
קשת עץ tree edge **89, 91, 95**
קשת קדימה forward edge **95**
קשת קלה light edge **110**
קשת קריטית critical edge **195**
קשת רוויה saturated edge **203**
קשת שיורית residual edge **186**

ר

רדוקציה reducibility **492-493**
רוטציה מעגלית cyclic rotation **446** (תרגיל)
ריבוי מקורות ובורות multiple sources and sinks **182-183**
ריבוע של גרף מכוון square of a directed graph **83** (תרגיל)
רישא של מחרוזת (≺) prefix of a string **427**
רכיב בנייה widget **512**
רכיב דו־קשיר biconnected component **105** (בעיה)
רכיב צירופי combinational element **245-246, 293**
רכיב צירופי בוליאני boolean combinational element **245**
רכיב קשיר connected component
זיהוי באמצעות חיפוש לעומק identified using depth-first search **97** (תרגיל)
זיהוי באמצעות מבני נתונים לקבוצות זרות identified using disjoint-set data structures **57-58**
זיהוי על CRCW PRAM identified on a CRCW PRAM **308** (בעיה)
פירוק לרכיבים קשירים היטב decomposition into strongly connected components **100-105**
רשימה מקושרת linked list
למימוש קבוצות זרות to implement disjoint sets **58-61**
קבוצה k־שליטה של k-ruling set of **307** (תרגיל)
רשימת בנים בערימת פיבונצ'י child list in a Fibonacci heap **39**
רשימת שורשים root list
של ערימה בינומית of a binomial heap **25**
של ערימת פיבונצ'י of a Fibonacci heap **40**
רשימת שכנים neighbor list **211**
רשת network
השוואה comparison **227-231**
זרימה, ראה: רשת זרימה flow, *see* flow network
חילוף transposition **241** (בעיה)
מותרת admissible **210**

מיון sorting **243-227**

מיון ביטונית bitonic sorting **237-234**

מיון זוג־פרד odd-even sorting **241** (בעיה)

מיזוג זוג־פרד odd-even merging **242** (בעיה)

עבור מיזוג for merging **239-237**

שיורית residual **188-186**

תמורה permutation **242** (בעיה)

רשת המיון AKS AKS sorting network **243**

רשת המיזוג זוג־פרד של באטצ'ר Batcher's odd-even merging network **242** (בעיה)

רשת השוואה comparison network **231-227**

רשת זרימה flow network **185-179**

המתאימה לגרף דו־צדדי corresponding to a bipartite graph **197**

חתך של cut of **191-189**

עם קיבולים שליליים with negative capacities **221** (בעיה)

ריבוי־מוצרים multicommodity **185** (תרגיל)

רשת זרימה מרובת מוצרים multicommodity flow network **185** (תרגיל)

רשת חילוף transposition network **241** (בעיה)

רשת מותרת admissible network **211-210**

רשת מיון sorting network **230, 230** (תרגילים)

ביטונית bitonic **237-234**

המבוססת על מיון־הכנסה based on insertion sort **230** (תרגיל)

המבוססת על מיון־מיזוג based on merge sort **241-239**

זוג־פרד odd-even **241** (בעיה)

AKS **243**

רשת מיון ביטונית bitonic sorting network **237-234**

רשת מיון זוג־פרד odd-even sorting network **241** (בעיה)

רשת מיזוג merging network **239-237**

זוג־פרד odd-even **242** (בעיה)

רשת שיורית residual network **188-186**

רשת תמורה permutation network **242** (בעיה)

ש

שארית remainder **381**

שארית residue **423, 381** (בעיה)

שארית ריבועית quadratic residue **423** (בעיה)

שבירת סימטריה symmetry breaking **307-301**

שבירת סימטריה באופן דטרמיניסטי deterministic symmetry breaking **307-301**

שגיאה יחסית relative error **521**

שדה אלגברי algebraic field **329**

שורש root

של היחידה of unity **363**

של Z_n^* of Z_n^* **403**

שורש מרוכב של היחידה complex root of unity **364-363**

אינטרפולציה ב־ interpolation at **369-368**

שורש פרימיטיבי של Z_n^* primitive root of Z_n^* **403**

שורש ראשי של היחידה principal root of unity **363**

שורש ריבועי לא טריוויאלי של 1, מודולו n nontrivial square root of 1, modulo n **404**

שורש ריבועי, מודולו מספר ראשוני square root, modulo a prime **423** (בעיה)

שטח של מצולע פשוט area of a simple polygon **459** (תרגיל)

שיטת גיזום־וחיפוש prune-and-search method **465**

שיטת הבנייה ההדרגתית עבור מציאת הקמור incremental design method for finding the convex hull **465**

שיטת החילוץ של גאוס Gaussian elimination **335**

שיטת המעבר היחיד one-pass method **75**

שיטת העקום האליפטי עבור פירוק לגורמים elliptic-curve factorization method **425**

שיטת הפוטנציאל potential method

עבור אלגוריתם CRCW למציאת רכיבים קשירים

for CRCW connected-components algorithm **308** (בעיה)

עבור אלגוריתם כללי לדחיפת קדם־זרימה for generic preflow-push algorithm **208**

עבור אלגוריתם קנותי־מוריס־פראט for Knuth-Morris-Pratt algorithm **442**

עבור ערימות פיבונצ'י for Fibonacci heaps **51, 47, 43-40**

שיטת הצבירה aggregate method

עבור הסריקה של גראהם for Graham's scan **471-470**

עבור מבני נתונים של קבוצות זרות for disjoint-set data structures **72-69** (תרגילים), **65** (תרגיל), **61, 60**

עבור ערימות פיבונצ'י for Fibonacci heaps **51** (תרגיל)

שיטת התכנון הדינמי dynamic-programming method

עבור אלגוריתם פלויד־וורשאל for Floyd-Warshall algorithm **163-159**

עבור מסלולים קצרים ביותר בין כל הזוגות for all-pairs shortest paths **163-154**

עבור סגור טרנזיטיבי for transitive closure **164-163**

עבור עץ פורש מינימלי for minimum spanning tree **177** (בעיה)

עבור תוויות של מסלולים מכוונים for directed-path labels **175-174**

שיטת מסלול אוילר Euler-tour technique **284-282**

שיטת פאן לכפל מטריצות Pan's method of matrix multiplication **325** (תרגיל)

שיטת פורד־פולקרסון Ford-Fulkerson method **196-185**

שיטת שני המעברים two-pass method **64**

שימור זרימה flow conservation **179**

שינוי ערך מפתח בערימת פיבונצ'י changing a key in a Fibonacci heap **55** (בעיה)

שכבות layers

מקסימליות maximal **478** (בעיה)

קמורות convex **477** (בעיה)

שכבות קמורות convex layers **477** (בעיה)

שלילי negative

במערכת אלגברית in an algebraic system **327**

של מטריצה of a matrix **314**

שליפת המפתח המינימלי extracting the minimum key

מערימות בינומיות from binomial heaps **33-32**

מערימות פיבונצ'י from Fibonacci heaps **47-43**

מערימות 2-3-4 from 2-3-4 heaps **37-36** (בעיה)

שלמות ב־NP NP-completeness **520-480**

הוכחה של, עבור שפה proving, of a language **500-499**

של בעיית האיזומורפיות של תת־גרף of the subgraph-isomorphism problem **518** (תרגיל)

של בעיית הכיסוי על־ידי קדקודים of the vertex-cover problem **509-508**

של בעיית המסלול ההמילטוני of the hamiltonian-path problem **518** (תרגיל)

של בעיית המעגל ההמילטוני of the hamiltonian-cycle problem **517-512**

של בעיית המעגל הפשוט הארוך ביותר of the longest-simple-cycle problem **518** (תרגיל)

של בעיית הסוכן הנוסע of the traveling-salesman problem **518-517**

של בעיית הקבוצה הבלתי־תלויה of the independent-set problem **519** (בעיה)

של בעיית הקליקה of the clique problem **508-506**

של בעיית חלוקת הקבוצה of the set-partition problem **518** (תרגיל)

של בעיית כיסוי הקבוצה of the set-covering problem **533** (תרגיל)

של בעיית סכום תת־קבוצה of the subset-sum problem **512-510**

של בעיית ספיקות המעגלים of the circuit-satisfiability problem **498-494**

של בעיית ספיקות הנוסחאות of the formula-satisfiability problem **502-500**

של בעיית הספיקות של 3-CNF of the 3-CNF-satisfiability problem **505-502**

של בעיית צביעת הגרף of the graph-coloring problem **519** (בעיה)

של הקביעה אם נוסחה בוליאנית היא טאוטולוגיה

of determining whether a boolean formula is a tautology **505** (תרגיל)

של גרסת 0-1 של בעיית התכנון בשלמים of the 0-1 integer-programming problem **518** (תרגיל)

שלמות של שפה completeness of a language **499** (תרגיל)

שעון clock **266**

שער gate **245**

שער לוגי logic gate **245**

שער AND AND gate **245**

שער NAND NAND gate **245**

שער NOR NOR gate **245**

שער NOT NOT gate **245**

שער OR OR gate **245**

שער XOR XOR gate **245**

שפה language **484**

אימות של verification of **489**

הוכחת שלמות ב־NP של proving NP-completeness of **500-499**

שלמות של completeness of **499** (תרגיל)

שקילות, מודולו equivalence, modular (≡) **381**

שקלול מחדש reweighting

במסלולים קצרים ביותר בין כל הזוגות in all-pairs shortest paths **166**

במסלולים קצרים ביותר ממקור יחיד in single-source shortest paths **148** (בעיה)

שריג grid **218** (בעיה)

שרשור concatenation

של מחרוזות of strings **427**

של מסלולים of paths **171**

של שפות of languages **484**

שרשרת של קמור chain of a convex hull **471**

ת

תבנית בהתאמת מחרוזות pattern in string matching **426**

תבנית מחרוזת נטולת חפיפות nonoverlappable string pattern **439** (תרגיל)

תו מרווח gap character **430-429** (תרגיל)

תור queue

באלגוריתמי דחיפת־קדם־זרימה in preflow-push algorithms **218** (תרגיל)

בחיפוש לרוחב in breadth-first search **85-84**

תור קדימויות priority queue

באלגוריתם של דייקסטרה in Dijkstra's algorithm **134**

באלגוריתם של פרים in Prim's algorithm **117-116, 115**

ראה גם: עץ חיפוש בינרי, ערימה בינומית, ערימת פיבונצ'י

see also binary search tree, binomial heap, Fibonacci heap

תחום אפשרי feasible region **142** (הערה)

תחשיב מסלולים בגרפים מכוונים calculus of paths in directed graphs **173-170**

תיל wire **228, 246**

תיל פלט output wire **228**

תיל קלט input wire **228**

תכנון לינארי linear programming

וזרימה מקסימלית and maximum flow **185** (תרגיל)

וזרימה של ריבוי מוצרים and multicommodity flow **185** (תרגיל)

ומסלולים קצרים ביותר ממקור יחיד and single source shortest paths **147–142**
תלות dependence
לינארית linear **316**
ראה גם: אי־תלות *see also* independence
תלות לינארית linear dependence **316**
תת־גרף של קודמים predecessor subgraph
בחיפוש לעומק in depth-first search **91**
בחיפוש לרוחב in breadth-first search **89–88**
במסלולים קצרים ביותר בין כל הזוגות in all-pairs shortest paths **153**
במסלולים קצרים ביותר ממקור יחיד in single-source shortest paths **123–122**
תת־חבורה subgroup **395–394**
תת־חבורה ממש proper subgroup **394**
תת־מבנה אופטימלי optimal substructure
של מסלולים קצרים ביותר of shortest paths **125–124, 154, 160**
תת־מטריצה מובילה leading submatrix **347**

מפתח מונחים אנגלי־עברי

Notations

× cross product מכפלה וקטורית **455**
∧ AND וגם **245, 327**
¬ NOT לא **245**
∨ OR או **245, 327**
⊕
algebraic operator אופרטור אלגברי **326**
group operator אופרטור על חבורות **391**
summary operator for closed semirings אופרטור סיכום עבור חוגים למחצה סגורים **170**
XOR exclusive-or או־מוציא **245**
⊙
algebraic operator אופרטור אלגברי **327**
extension operator for closed semirings אופרטור הרחבה עבור חוגים למחצה סגורים **170**
⊗
carry-status operator אופרטור מצבי נשא **251**
convolution operator אופרטור קונבולוציה **358**
parallel-prefix operator אופרטור לחישוב מקבילי של רישות **281**
* closure operator אופרטור סגור **173, 484**
| divides relation היחס "מחלק את" **380**
≡ equivalent modulo n n שקול מודולו **381**
≢ not equivalent modulo n n לא שקול מודולו **381**
$[a]_n$ equivalence class modulo n n מחלקת שקילות מודולו **381**
$+_n$ addition modulo n n חיבור מודולו **391**
$\cdot_n$ multiplication modulo n n כפל מודולו **391**
ε empty string המחרוזת הריקה **427, 484**
≺ prefix relation יחס הרישא **427**
≻ suffix relation יחס הסיפא **427**
∼ similar relation יחס הדמיון **450**
$\succ_x$ above relation היחס "מעל" **460**
$\leq_p$ polynomial-time reducibility relation היחס "ניתן לרדוקציה בזמן פולינומיאלי" **492**

A

abelian group חבורה אבלית **391**
abstract problem בעיה מופשטת **481**
acceptance קבלה
by an algorithm על־ידי אלגוריתם **485**

by a finite automaton על־ידי אוטומט סופי **435**
accepting state מצב מקבל **435**
Ackermann's function פונקציית אקרמן **65-67**
addition חיבור
bit-serial סיביות סדרתי **265-267**
carry-lookahead צופה נשא **251-255**
carry-save שומר נשא **256-257**
circuits for מעגלים עבור **249-259**
of matrices מטריצות **314**
modulo n $(+_n)$ מודולו n **391**
of polynomials של פולינומים **356**
ripple-carry מגלגל נשא **249-250**
additive group modulo n חבורה חיבורית מודולו n **391**
additive inverse הופכי חיבורי **327**
adjacency-list representation ייצוג על־ידי רשימות סמיכות **80**
adjacency-matrix representation ייצוג על־ידי מטריצת סמיכויות **81**
admissible edge קשת מותרת **210**
admissible network רשת מותרת **210-211**
aggregate method שיטת הצבירה
for disjoint-set data structures עבור מבני נתונים של קבוצות זרות **60,** (תרגיל) **61,** (תרגילים) **65, 69-72**
for Fibonacci heaps עבור ערימות פיבונצ'י (תרגיל) **51**
for Graham's scan עבור הסריקה של גראהם **470-471**
AKS sorting network רשת המיון AKS **243**
algebraic field שדה אלגברי **329**
all-pairs shortest paths מסלולים קצרים ביותר בין כל הזוגות **121**
in $\in$-dense graphs בגרפים $\in$־צפופים (בעיה) **176**
Floyd-Warshall algorithm אלגוריתם פלויד־וורשאל **159-163**
Johnson's algorithm האלגוריתם של ג'ונסון **166-169**
by matrix multiplication באמצעות כפל מטריצות **154-159**
by repeated squaring באמצעות העלאה חוזרת בריבוע **157-158**
alphabet אלפבית **435, 484**
amortized analysis ניתוח לשיעורין
for CRCW connected-components algorithm עבור אלגוריתם CRCW למציאת רכיבים קשירים (בעיה) **308**
for disjoint-set data structures עבור מבני נתונים של קבוצות זרות **60,** (תרגיל) **61,** (תרגילים) **65, 69-7**
for Fibonacci heaps עבור ערימות פיבונצ'י **40-43, 47, 50-51,** (תרגיל) **51**
For generic preflow-push algorithm עבור האלגוריתם הכללי לדחיפת קדם־זרימה **208**
for Graham's scan עבור הסריקה של גראהם **470-471**
for Knuth-Morris-Pratt algorithm עבור אלגוריתם קנות'־מוריס־פראט **442**

for stacks on secondary storage עבור מחסניות על התקני אחסון משניים (בעיה) **19**
analysis of algorithms ניתוח אלגוריתמים
see also probabilistic analysis ראה גם: ניתוח הסתברותי
ancestor אב קדמון
least common משותף עמוק ביותר (בעיה) **74**
AND function $(\wedge)$ AND הפונקציה **245, 327**
on a CRCW PRAM CRCW PRAM על **289**
AND gate AND שער **245**
annihilator מאפֵס **170, 326**
approximation algorithm אלגוריתם קירוב **521-539**
for bin packing עבור אריזה בתיבות (בעיה) **538**
for the maximum-clique problem עבור בעיית הקליקה המקסימלית (בעיה) **538**
for the set-covering problem עבור בעיית כיסוי הקבוצה **529-533**
for the subset-sum problem עבור בעיית סכום התת־קבוצה **533-537**
for the traveling salesman problem עבור בעיית הסוכן הנוסע **525-529**
for the vertex-cover problem עבור בעיית הכיסוי על־ידי קדקודים **523-525**
for the weighted set-covering problem עבור הגרסה המשוקללת של בעיית כיסוי הקבוצה (בעיה) **538**
approximation by least squares קירוב על־ידי ריבועים פחותים **348-351**
approximation scheme סכמת קירוב **522-523**
arbitrage ארביטרז' (בעיה) **148**
area of a simple polygon שטח של מצולע פשוט (תרגיל) **459**
arithmetic, modular אריתמטיקה מודולרית **390-396**
arithmetic circuits מעגלים אריתמטיים **244-273**
array multiplier כופל מערכי **259-262**
articulation point נקודת הפרדה (בעיה) **105**
assignment השמה
satisfying מספקת **494, 500**
truth של ערכי אמת **494, 500**
associative operation פעולה אסוציאטיבית **170, 326, 391**
augmenting path מסלול שיפור **188**
automaton אוטומט
cellular תאי **310**
finite סופי **435**
string-matching התאמת־מחרוזות **435-439**

B

back edge קשת אחורה **95, 98**
back substitution הצבה אחורה **333-334**

bad-character heuristic היוריסטיקת התו הרע **448-450**
balanced search tree עץ חיפוש מאוזן
B-trees B־עצי **5-20**
2-3-4 trees 2-3-4 עצי **8**
barrel shifter מעגל הזזה גלילית (תרגיל) **265**
Batcher's odd-even merging network רשת המיזוג זוג־פרד של באטצ'ר (בעיה) **242**
Bellman-Ford algorithm אלגוריתם בלמן־פורד **136-139**
in Johnson's algorithm באלגוריתם של ג'ונסון **169**
and objective functions ופונקציות מטרה (תרגילים) **147**
to solve systems of difference constraints לפתרון מערכות של אילוצי הפרשים **145**
Yen's improvement to השיפור של ין **147**
Bernoulli trial ניסוי ברנולי **300**
biconnected component רכיב דו־קשיר (בעיה) **105**
binary counter מונה בינרי
and binomial heaps וערימות בינומיות (תרגיל) **36**
binary gcd algorithm אלגוריתם gcd בינרי (בעיה) **423**
binary search חיפוש בינרי
in searching B-trees בחיפוש בעצי B־ (תרגיל) **16**
binary tree עץ בינרי
finding the root of, on a CREW PRAM מציאת השורש של, על CREW PRAM (תרגיל) **291**
binomial heap ערימה בינומית **21-37**
and binary counter and binary addition ומונה בינרי וחיבור בינרי (תרגיל) **36**
creating יצירה של **26**
decreasing a key in הקטנת ערך מפתח ב־ **33-35**
deletion from מחיקה מתוך **35**
extracting the minimum key from שליפת המפתח המינימלי מתוך **32-33**
insertion into הכנסה לתוך **32**
minimum key of המפתח המינימלי של **27**
properties of תכונות של **24**
running times of operations on זמני ריצה של פעולות על (איור) **22**
uniting איחוד **27-32**
binomial tree עץ בינומי **22-24**
unordered לא סדור **41**
bin packing אריזה בתיבות **538**
bipartite graph גרף דו־צדדי
corresponding flow network of רשת הזרימה המתאימה ל־ **197**
determination of קביעה של (תרגיל) **90**
bipartite matching זיווג בגרף דו־צדדי **55, 196-200, 222**

bitonic sequence סדרה ביטונית **234**
bitonic sorting network רשת מיון ביטונית **234-237**
bit-serial addition חיבור סיבית סדרתי **265-267**
black vertex קדקוד שחור **84, 91**
boolean combinational element רכיב צירופי בוליאני **245**
boolean connective קֶשֶׁר בוליאני **500**
boolean formula נוסחה בוליאנית (תרגיל) **491, 500,** (תרגילים) **505**
boolean matrix multiplication כפל מטריצות בוליאניות
and transitive closure וסגור טרנזיטיבי (תרגילים) **330,** (תרגיל) **346**
using randomization תוך שימוש באקראיות (בעיה) **352**
using Strassen's algorithm תוך שימוש באלגוריתם של שטראסן **328-329,** (בעיה) **352**
boolean quasiring מעין־חוג בוליאני **327**
bottleneck traveling-salesman problem בעיית צוואר הבקבוק של הסוכן הנוסע (תרגיל) **529**
bound חסם
ratio יחס **521**
relative error שגיאה יחסית **522**
bounding box קופסה חוסמת **457**
boyer-moore algorithim אלגוריתם בויאר־מור **447-452**
breadth-first search חיפוש לרוחב **83-90**
and shortest paths ומסלולים קצרים ביותר **86-88, 120**
similarity to Dijkstra's algorithm דמיון לאלגוריתם של דייקסטרה **135**
breadth-first tree עץ חיפוש לרוחב **83-84, 88-89**
Brent's theorem משפט ברנט **293-294,** (תרגיל) **296**
bridge גשר (בעיה) **105**
B-tree B־עץ **5-20**
creating יצירה **11**
deletion from מחיקה מתוך **16-18**
full node in צומת מלא בתוך **8**
height of גובה של **9**
insertion into הכנסה לתוך **13-15**
minimum degree of דרגה מינימלית של **8**
minimum key of מפתח מינימלי ב־ (תרגיל) **15**
properties of תכונות של **8-10**
searching חיפוש ב־ **10-11**
splitting a node in פיצול צומת ב־ **11-13**
2-3-4 trees 2-3-4 עצי **8**
butterfly operation פעולת פרפר **371**

C

calculus of paths in directed graphs תחשיב מסלולים בגרפים מכוונים **170–173**

cancellation lemma למת הצמצום **364**

cancellation of flow קיזוז זרימה **181–182**

capacity קיבול

  of a cut של חתך **189**

  of an edge של קשת **179**

  residual שיורי **186, 188**

capacity constraint אילוץ קיבול **179**

Carmichael number מספר קארמייקל **413,** (תרגיל) **418**

carry generate יצירת נשא **251**

carry-in bit סיבית נשא־נכנס **250**

carry kill איפוס נשא **251**

carry-lookahead addition חיבור צופה־נשא **251–255**

carry-out bit סיבית נשא־יוצא **250**

carry propagate קידום נשא **251**

carry-save addition חיבור שומר־נשא **256–257,** (תרגיל) **264**

carry status מצב נשא **251**

carry-status operator (⊗) אופרטור מצבי נשא **251**

Cartesian sum סכום קרטזי (תרגיל) **363**

cascading cut ניתוק שרשרת **49,** (תרגיל) **54**

cellular automaton אוטומט תאי **310**

certificate אישור

  in a cryptosystem במערכת הצפנה **411**

  for verification algorithms עבור אלגוריתמי אימות **488**

chain of a convex hull שרשרת של קמור **471**

changing a key in a Fibonacci heap שינוי ערך מפתח בערימת פיבונצ'י (בעיה) **55**

child list in a Fibonacci heap רשימת בנים בערימת פיבונצ'י **39**

Chinese remainder theorem משפט השאריות הסיני **399–402**

chirp transform chirp התמרת (תרגיל) **370**

chromatic number, *see* coloring מספר כרומטי, ראה: צביעה

ciphertext כתב צופן **407**

circuit מעגל

  for addition עבור חיבור **249–259, 265–267**

  clocked מבוקר־שעון **265–271**

  combinational צירופי **244–249**

  for division עבור חילוק (בעיה) **271**

  for Fast Fourier Transform עבור התמרת פוריה המהירה **374–375**

  globally clocked מבוקר־שעון גלובלית **266**

  for multiplication עבור כפל **259–265, 268–271**

  for parallel prefix עבור חישוב רישות מקבילי **253–255**

  sequential סדרתי **265–271**

  tally מונה (תרגיל) **258**

  see also comparison network ראה גם: רשת השוואה

circuit satisfiability ספיקות מעגלים **494–498**

class מחלקה

  complexity של סיבוכיות **485**

classification of edges סיווג קשתות

  in breadth-first search בחיפוש לרוחב (בעיה) **105**

  in depth-first search בחיפוש לעומק **95–96**

classification of edges in depth-first search סיווג קשתות בחיפוש לעומק (תרגילים) **96–97**

clean sequence סדרה נקייה **234**

clique קליקה **506**

clique problem בעיית הקליקה **506–508**

  approximation algorithm for אלגוריתם קירוב עבור (בעיה) **538**

  and vertex-cover problem ובעיית הכיסוי על־ידי קדקודים (תרגיל) **525**

clock שעון **266**

clocked circuit מעגל מבוקר־שעון **265–271**

clock period זמן מחזור של שעון **266**

closed semiring חוג־למחצה סגור **170, 173–174**

  for minimum spanning tree עבור עץ פורש מינימלי (בעיה) **177**

  as a quasiring בתור מעין־חוג **327**

closest pair, finding הזוג הקרוב ביותר, מציאה של **473–477**

closest-point heuristic היוריסטיקת הנקודה הקרובה ביותר (תרגיל) **529**

closure סגור

  group property תכונה של קבוצה **391**

  of a language של שפה **484**

  operator (*) אופרטור **173**

  transitive, *see* transitive closure טרנזיטיבי, ראה: סגור טרנזיטיבי

  under an operator תחת אופרטור **170, 326**

CNF (conjunctive normal form) צורה קוניונקטיבית נורמלית **502**

coefficient מקדם

  of a polynomial של פולינום **356**

coefficient representation ייצוג בעזרת מקדמים **358**

  and fast multiplication וכפל מהיר **361–362**

cofactor משלים **317**

collinearity הימצאות על ישר אחד **455,** (תרגיל) **458**

coloring צביעה **302,** (בעיה) **519**

on an EREW PRAM EREW PRAM על **303-305**

column rank דרגת עמודות **316**

column vector וקטור עמודה **312**

combinational circuit מעגל צירופי **244-249, 293**

and equivalent clocked circuit ומעגל מבוקר־שעון שקול **267**

simulated by a CRCW PRAM CRCW PRAM הדמיה על־ידי (תרגיל) **296**

simulated by a CREW PRAM CREW PRAM הדמיה על־ידי **293-294**

simulated by an EREW PRAM EREW PRAM הדמיה על־ידי **294-295**

see also circuit, comparison network ראה גם: מעגל, רשת השוואה

combinational element רכיב צירופי **245-246, ,293**

common-CRCW model מודל CRCW־משותף **276**

common divisor מחלק משותף **382**

greatest, *see* greatest common divisor מקסימלי, ראה: מחלק משותף מקסימלי

commutativity under an operator קומוטטיביות תחת אופרטור **170, 326, 329, 391**

comparable line segments קטעים בני־השוואה **460**

comparator משווה **227,** (תרגילים) **230,** (תרגילים) **239,** (תרגילים) **240**

comparison network רשת השוואה **227-231**

comparison sort מיון־השוואה

and mergeable heaps וערימות מתמזגות (תרגיל) **48**

complement משלים

of a graph של גרף **508**

of a language של שפה **484**

Schur שור **336, 347**

completeness of a language שלמות של שפה (תרגיל) **499**

complexity class מחלקת סיבוכיות **485**

co-NP **489**

NP **489**

NPC **493**

P **483**

complexity measure מדד סיבוכיות **485**

complex numbers, multiplication of מספרים מרוכבים, כפל של (תרגיל) **326**

complex root of unity שורש מרוכב של היחידה **363-364**

interpolation at אינטרפולציה ב־ **368-369**

component רכיב

biconnected דו־קשיר (בעיה) **105**

component graph גרף רכיבים (תרגילים) **104-105**

composite number מספר פריק **380**

computational geometry גיאומטריה חישובית **454-479**

concatenation שרשור

of languages של שפות **484**

of paths של מסלולים **171**

of strings של מחרוזות **427**

concrete problem בעיה קונקרטית **482**

concurrent-read קריאה בו־זמנית **275**

concurrent-write כתיבה בו־זמנית **275**

conjugate transpose מוחלפת צמודה (תרגיל) **346**

conjunctive normal form צורה קוניונקטיבית נורמלית **502**

connected component רכיב קשיר

identified on a CRCW PRAM CRCW PRAM זיהוי על (בעיה) **308**

identified using depth-first search זיהוי באמצעות חיפוש לעומק (תרגיל) **97**

identified using disjoint-set data structures זיהוי באמצעות מבני נתונים של קבוצות זרות **57-58**

connective קשר **500**

co-NP **489**

conservation of flow שימור זרימה **179**

consistency of literals קונסיסטנטיות של ליטרלים **507**

consolidating a Fibonacci-heap root list מיזוג רשימת שורשים בערימת פיבונצ'י **44,** (תרגיל) **48**

constraint graph גרף אילוצים **144-145**

convex combination of points צירוף קמור של נקודות **454**

convex hull קמור **464-473,** (בעיה) **479**

convex layers שכבות קמורות (בעיה) **477**

convolution (⊗) קונבולוציה **358**

convolution theorem משפט הקונבולוציה **369**

cover כיסוי

path מסלול (בעיה) **219**

by a subset על־ידי תת־קבוצה **530**

vertex על־ידי קדקודים **508, 523**

CPU time זמן היעי"מ **6**

CRCW (concurrent-read, concurrent-write) קריאה בו־זמנית, כתיבה בו־זמנית **276**

common משותף **276**

compared to EREW בהשוואה ל־EREW **285-292**

finding connected components מציאת רכיבים קשירים (בעיה) **308**

finding the maximum מציאת המקסימום **288-289,** (תרגיל) **292,** (בעיה) **308**

AND function AND פונקציית **289**

matrix multiplication כפל מטריצות (תרגיל) **292**

merging two sorted arrays מיזוג שני מערכים ממוינים (תרגיל) 292
OR function OR פונקציית (תרגיל) 292
simulated by EREW EREW הדמיה על־ידי 290-291
simulation of, by EREW EREW הדמיה של, על־ידי (תרגילים) 292, (תרגיל) 296
simulation of a combinational circuit הדמיה של מעגל צירופי (תרגיל) 296
work-efficient matrix multiplication כפל מטריצות יעיל־בעבודה (תרגיל) 296
CREW (concurrent-read, exclusive-write) קריאה בו־זמנית, כתיבה בלעדית 276
finding roots in a forest מציאת שורשים ביער 286-288
finding the root of a binary tree מציאת השורש של עץ בינרי (תרגיל) 291
simulation of a combinational circuit הדמיה של מעגל צירופי 293-294
string matching התאמת מחרוזות (בעיה) 453
critical edge קשת קריטית 195
critical path מסלול קריטי
in a combinational circuit במעגל צירופי (תרגיל) 258
in a directed acyclic graph בגרף מכוון ללא־מעגלים (גמ"ל) 141
cross edge קשת חוצה 95
crossing a cut חצייה חתך 110
cross product (×) מכפלה וקטורית 455
cryptosystem מערכת הצפנה 406-411
cubic spline פונקציית ספליין ממעלה שלישית (בעיה) 353
curve fitting התאמה לעקום 348-351
cut ניתוק
cascading שרשרת 49
cut חתך
of a flow network של רשת זרימה 189-191
of an undirected graph של גרף בלתי מכוון 110
cutting, in a Fibonacci heap ניתוק, בערימת פיבונצ'י 49
cycle of a graph מעגל בגרף
detection of גילוי של (תרגיל) 99
hamiltonian המילטוני 487
minimum mean-weight בעל משקל ממוצע מינימלי (בעיה) 149
negative-weight בעל משקל שלילי *see* negative-weight cycle ראה: מעגל בעל משקל שלילי
cyclic group חבורה מעגלית 403
cyclic rotation רוטציה מעגלית (תרגיל) 446
cyclic shifter מעגל הזזה מעגלית (תרגיל) 265

D

dag (directed acyclic graph) גמ"ל (גרף מכוון ללא־מעגלים)

and back edges וקשתות אחורה 98
and hamiltonian-path problem ובעיית המסלול ההמילטוני (תרגיל) 491
single-source shortest-paths algorithm for אלגוריתם מסלולים קצרים ביותר ממקור יחיד עבור 139-141
topological sort of מיון טופולוגי של 97-99
d-ary heap ערימה *d*־ית
in shortest-paths algorithms באלגוריתמי מסלולים קצרים ביותר (בעיה) 176
data structure מבנה נתונים 3-75
binomial heaps ערימות בינומיות 21-37
B-trees B־עצי 5-20
for disjoint sets עבור קבוצות זרות 56-75
dynamic trees עצים דינמיים 4
Fibonacci heaps ערימות פיבונצ'י 38-55
persistent מתמידים 4
relaxed heaps ערימות נינוחות 55
on secondary storage על התקן אחסון משני 6-7
splay trees splay עצי 4
2-3-4 heaps 2-3-4 ערימות (בעיה) 36
2-3 trees 2-3 עצי 20
van Emde Boas 4
see also dynamic set ראה גם: קבוצה דינמית
decision by an algorithm הכרעה על־ידי אלגוריתם 485
decision problem בעיית הכרעה 482
decreasing a key הקטנת ערך מפתח
in binomial heaps בערימות בינומיות 33-35
in Fibonacci heaps בערימות פיבונצ'י 48-51
in 2-3-4 heaps 2-3-4 בערימות (בעיה) 36
degree דרגה
of a binomial-tree root של שורש של עץ בינומי 23
maximum, of a Fibonacci heap מקסימלית, של ערימת פיבונצ'י 40, (תרגיל) 48, 52-54
minimum, of a B-tree מינימלית, של עץ־B 8
degree מעלה
of a polynominal של פולינום 356
degree-bound חסם על המעלה 356
deletion מחיקה
from binomial heaps מערימות בינומיות 35
from B-trees מעצי־B 16-18
from Fibonacci heaps מערימות פיבונצ'י 51, (בעיה) 54
from sweep-line statuses ממצבי קו־סריקה 461

from 2-3-4 heaps מערימות 2-3-4 (בעיה) **36-37**

dense graph גרף צפוף **80**

ϵ-dense ϵ־צפוף (בעיה) **176**

density of prime numbers צפיפות המספרים הראשוניים **412**

dependence תלות

linear לינארית **316**

see also independence ראה גם: אי־תלות

depth עומק

of a combinational circuit של מעגל צירופי **248, 293**

of a combinational element של רכיב צירופי **248, 293**

of a comparison network של רשת השוואה **229**

of SORTER SORTER של (תרגיל) **240**

of a sorting network של רשת מיון (תרגיל) **240**

depth-determination problem בעיית קביעת עומק (בעיה) **73**

depth-first forest יער עומק **91**

depth-first search חיפוש לעומק **90-97**

in finding articulation points, bridges, and biconnected components
במציאת נקודות הפרדה, גשרים ורכיבים דו־קשירים (בעיה) **105**

in finding strongly connected components במציאת רכיבים קשירים היטב **100-105**

in topological sorting במיון טופולוגי **97-99**

depth-first tree עץ עומק **91**

destination vertex קדקוד יעד **121**

determinant דטרמיננטה **317**

and matrix multiplication וכפל מטריצות (תרגיל) **346**

deterministic symmetry breaking שבירת סימטריה באופן דטרמיניסטי **301-307**

DFT (Discrete Fourier Transform) התמרת פוריה הבדידה **365-366**

diagonal matrix מטריצה אלכסונית **312**

LUP decomposition of של LUP פירוק (תרגיל) **342**

diameter of a tree קוטר של עץ (תרגיל) **90**

difference constraints אילוצי הפרשים **142-147**

difference equation, *see* recurrence משוואת הפרשים, ראה: נוסחת נסיגה

digital signature חתימה דיגיטלית **408**

Dijkstra's algorithm האלגוריתם של דייקסטרה **131-135**

implemented with a Fibonacci heap מימוש באמצעות ערימת פיבונצ'י **134**

implemented with a heap מימוש באמצעות ערימה **134**

with integer edge weights עם משקלי קשתות שלמים (תרגיל) **135**

in Johnson's algorithm באלגוריתם של ג'ונסון **169**

similarity to breadth-first search דמיון לחיפוש לרוחב **135**

similarity to Prim's algorithm דמיון לאלגוריתם של פרים **115, 135**

directed acyclic graph (dag) גרף מכוון ללא־מעגלים (גמ"ל)

and black edges וקשתות אחורה **98**

and hamiltonian-path problem ובעיית המסלול ההמילטוני (תרגיל) **491**

single-source shortest-paths algorithm for
אלגוריתם למציאת מסלולים קצרים ביותר ממקור יחיד עבור **139-141**

topological sort of מיון טופולוגי של **97-99**

directed graph גרף מכוון

all-pairs shortest paths in מסלולים קצרים ביותר בין כל הזוגות בתוך **152-177**

calculus of paths in תחשיב מסלולים ב־ **170-173**

component graph גרף רכיבים (תרגילים) **104-105**

constraint graph גרף אילוצים **144**

Euler tour of מסלול אוילר של (בעיה) **106**

path cover of כיסוי על־ידי מסלולים של (בעיה) **219**

PERT chart PERT דיאגרמת **141,** (תרגיל) **141**

semiconnected קשיר למחצה (תרגיל) **105**

shortest path in מסלול קצר ביותר בתוך **120**

single-source shortest paths in מסלולים קצרים ביותר ממקור יחיד בתוך **120-151**

singly connected חד־קשיר (תרגיל) **97**

square of ריבוע של (תרגיל) **83**

transitive closure of סגור טרנזיטיבי של **163**

transpose of גרף מוחלף של (תרגיל) **82**

see also circuit, directed acyclic graph (dag), graph, network
ראה גם: מעגל, גרף מכוון ללא מעגלים (גמ"ל), גרף, רשת

directed segment קטע מכוון **455-456**

discharge of an overflowing vertex פריקה של קדקוד גולש **211**

discovered vertex קדקוד שהתגלה **84, 91**

Discrete Fourier Transform התמרת פוריה הבדידה **366**

discrete logarithm function (log) פונקציית הלוגריתם בדיד **403**

discrete logarithm theorem משפט הלוגריתם הבדיד **403**

disjoint-set data structure מבנה נתונים של קבוצות זרות **56-75**

analysis of ניתוח של **69-72**

in depth determination בקביעת עומק (בעיה) **73**

disjoint-set-forest implementation of מימוש באמצעות יער קבוצות זרות **62-65**

in Kruskal's algorithm באלגוריתם של קרוסקל **113**

linear-time special case of מקרה פרטי שמתבצע בזמן לינארי **75**

linked-list implementation of מימוש על־ידי רשימה מקושרת **58-61**

in off-line least common ancestors

בגרסה הבלתי־מקוונת של האבות הקדמונים העמוקים ביותר (בעיה) **74**

in off-line minimum בבעיית המינימום הבלתי־מקוונת (בעיה) **73**

disjoint-set forest יער קבוצות זרות **62-65**

analysis of ניתוח של **69-72**

rank properties of תכונות הדרגות של **68-69**

see also disjoint-set data structure ראה גם: מבנה נתונים של קבוצות זרות

disjunctive normal form צורה דיסיונקטיבית נורמלית **504**

disk עיגול (תרגיל) **464**

disk, *see also* secondary storage דיסק, ראה גם: התקן אחסון משני **317**

distance מרחק

euclidean אוקלידי **473**

L_m (תרגיל) **477**

Manhattan מנהטן (תרגיל) **477**

of a shortest path של מסלול קצר ביותר **86**

distribution התפלגות

of prime numbers של המספרים הראשוניים **412**

sparse-hulled בעלת קמור דליל (בעיה) **479**

distributivity פילוג, דיסטריבוטיביות **170, 327**

divide-and-conquer method גישת הפרד־ומשול

for conversion of binary to decimal להמרה מייצוג בינרי לעשרוני (תרגיל) **384**

for Fast Fourier Transform עבור התמרת פוריה המהירה **366-368**

for finding the closest pair of points למציאת זוג הנקודות הקרובות ביותר **473-477**

for finding the convex hull למציאת הקמור **465**

for matrix inversion להיפוך מטריצות **344-345**

for multiplication עבור כפל (בעיה) **375**

for Strassen's algorithm עבור אלגוריתם שטראסן **320-326**

divides relation ($|$) היחס "מחלק את" **380**

division circuit מעגל חילוק (בעיה) **271**

division theorem משפט החילוק **381**

divisor מחלק **380**

common משותף **382**

see also greatest common divisor ראה גם: מחלק משותף מקסימלי

DNF (disjunctive normal form) צורה דיסיונקטיבית נורמלית **504**

dominates relation היחס "שליטי" (בעיה) **478**

d-regular graph גרף d־רגולרי (תרגיל) **200**

dynamic graph גרף דינמי

minimum-spanning-tree algorithm for אלגוריתם למציאת עץ פורש מינימלי של (תרגיל) **118**

transitive closure of סגור טרנזיטיבי של (בעיה) **176**

dynamic-programming method שיטת התכנון הדינמי

for all-pairs shortest paths עבור מסלולים קצרים ביותר בין כל הזוגות **154-163**

for directed-path labels עבור תוויות של מסלולים מכוונים **174-175**

for Floyd-Warshall algorithm עבור אלגוריתם פלויד־וורשאל **159-163**

for minimum spanning tree עבור עץ פורש מינימלי (בעיה) **177**

for transitive closure עבור סגור טרנזיטיבי **163-164**

dynamic set קבוצה דינמית

binomial heaps ערימות בינומיות **21-37**

B-trees B־עצי **5-20**

disjoint-set data structure מבנה נתונים של קבוצות זרות **56-75**

Fibonacci heaps ערימות פיבונצ'י **38-55**

persistent מתמידה **4**

relaxed heaps ערימות נינוחות **55**

splay trees splay עצי **4**

2-3-4 heaps 2-3-4 ערימות (בעיה) **36**

2-3 trees 2-3 עצי **20**

van Emde Boas data structure van Emde Boas מבנה נתונים של **4**

dynamic tree עץ דינמי **4**

E

edge קשת

admissible מותרת **210**

back אחורה **95**

bridge גשר (בעיה) **105**

capacity of קיבול של **179**

classification in breadth-first search סיווג בחיפוש לרוחב (בעיה) **105**

classification in depth-first search סיווג בחיפוש לעומק **95-96**

critical קריטית **195**

cross חוצה **95**

forward קדימה **95**

inadmissible בלתי־מותרת **210**

label of תווית של **170**

light קלה **110**

negative-weight בעלת משקל שלילי **121-122**

residual שיורית **186**

safe בטוחה **109**

saturated רוויה **203**

tree עץ **89, 91, 95**

weight of משקל של **81**

edge connectivity קשירות קשתות (תרגיל) **196**

Edmonds-Karp algorithm אלגוריתם אדמונדס־קארפ **193**

ellipsoid algorithm אלגוריתם האליפסואיד **142, 151**

elliptic-curve factorization method שיטת העקום האליפטי עבור פירוק לגורמים **425**

empty language (ϕ) השפה הריקה **484**

empty string (ε) המחרוזת הריקה **427, 484**

encoding of problem instances קידוד של מופעי בעיה **482-484**

endpoint נקודת קצה

of a line segment של קטע ישר **455**

ϵ-dense graph גרף ϵ־צפוף (בעיה) **176**

equivalence, modular ($\equiv$) שקילות, מודולו **381**

equivalence class מחלקת שקילות

modulo n ($[a]_n$) n מודולו **381**

ERCW (exclusive-read, concurrent-write) קריאה בלעדית, כתיבה בו־זמנית **276**

EREW (exclusive-read, exclusive-write) קריאה בלעדית, כתיבה בלעדית **276**

coloring צביעה **303-305**

compared to CRCW בהשוואה ל־CRCW **285-292**

computing a maximal independent set from a 6-coloring

חישוב קבוצה בלתי־תלויה מקסימלית מצביעת־6 **306**

deterministic symmetry breaking שבירת סימטריה באופן דטרמיניסטי **301-307**

Euler-tour technique שיטת מסלול אוילר **282-284**

finding roots in a forest מציאת שורשים ביער (תרגיל) **292**

AND function פונקציית AND **289**

list ranking מיקום איברים ברשימה **278-280**

matrix multiplication כפל מטריצות (תרגיל) **292**

matrix-vector multiplication כפל מטריצה בווקטור (תרגיל) **296**

parallel prefix חישוב רישות מקבילי **280-282, 296-301**

pointer jumping הקפצת מצביעים **277-285**

segmented parallel prefix חישוב רישות מקבילי מקוטע (בעיה) **307-308**

simulation of a combinational circuit הדמיה של מעגל צירופי **294-295**

simulation of CRCW הדמיה של CRCW **290-291,** (תרגילים) **292,** (תרגיל) **296**

sorting מיון **295**

escape problem בעיית דרכי מילוט (בעיה) **218**

Euclid's algorithm האלגוריתם של אוקלידס **385-390,** (בעיה) **423**

euclidean distance מרחק אוקלידי **473**

euclidean norm נורמה אוקלידית **315-316**

Euler's phi function פונקציית פי של אוילר **393**

Euler's theorem משפט אוילר **403,** (תרגיל) **418**

Euler tour מסלול אוילר (בעיה) **106**

Euler-tour technique שיטת מסלול אוילר **282-284**

evaluation of a polynomial מציאת ערכו של פולינום **358,** (תרגיל) **362**

and its derivatives ונגזרותיו (בעיה) **376**

at multiple points בנקודות מרובות (בעיה) **376**

event-point schedule לוח זמנים של נקודות אירוע **461**

excess flow זרימה עודפת **201**

exclusive-read קריאה־בלעדית **275**

exclusive-write כתיבה־בלעדית **275**

exponentiation העלאה בחזקה

modular מודולרית **404**

repeated חוזרת **65-66**

extension operator for closed semirings ($\circledcirc$) אופרטור הרחבה עבור חוגים־למחצה סגורים **170**

extracting the minimum key שליפת המפתח המינימלי

from binomial heaps מערימות בינומיות **32-33**

from Fibonacci heaps מערימות פיבונצ'י **43-47**

from 2-3-4 heaps מערימות 2-3-4 (בעיה) **36-37**

extreme point נקודת קיצון (הערה) **142**

F

factor גורם **380**

factorization פירוק לגורמים **418-422, 425**

unique יחיד **384**

fan-in פרישת כניסה **247, 293**

fan-out פרישת יציאה **246, 293**

farthest-pair problem בעיית זוג הנקודות הרחוקות ביותר **465**

Fast Fourier Transform (FFT) התמרת פוריה המהירה

circuit for מעגל עבור **374-375**

iterative implementation of מימוש איטרטיבי של **370-374**

recursive implementation of מימוש רקורסיבי של **366-368**

using modular arithmetic באמצעות אריתמטיקה מודולרית (בעיה) **377**

feasibility problem בעיית אפשרות **142**

feasible region תחום אפשרי (הערה) **142**

feasible solution פתרון אפשרי **142**

Fermat's theorem משפט פרמה **403**

FFT, *see* Fast Fourier Transform ראה: התמרת פוריה המהירה, FFT

Fibonacci heap ערימת פיבונצ'י **38-55**

changing a key in שינוי ערך מפתח ב־ (בעיה) 55
creating יצירה של 41
decreasing a key in הקטנת ערכו של מפתח ב־ 48-51
deletion from מחיקה מתוך 51, (בעיה) 54
in Dijkstra's algorithm באלגוריתם של דייקסטרה 134
extracting the minimum key from שליפת המפתח המינימלי מתוך 43-47
insertion into הכנסה לתוך 41-42
in Johnson's algorithm באלגוריתם של ג'ונסון 169
maximum degree of דרגה מקסימלית של 40, 52-54
minimum key of מפתח מינימלי של 42
potential function for פונקציית פוטנציאל עבור 40
in Prim's algorithm באלגוריתם של פרים 117
pruning גיזום (בעיה) 55
running times of operations on זמני הריצה של פעולות על (איור) 22
uniting איחוד 43
Fibonacci numbers מספרי פיבונצ'י 52
computation of חישוב של (בעיה) 423
final-state function פונקציית מצבים סופיים 435
finding the maximum מציאת מקסימום 288-289, (תרגיל) 292, (בעיה) 308
find path מסלול גילוי 62
FIND-SET פעולת FIND-SET 57
disjoint-set-forest implementation of מימוש באמצעות יער קבוצות זרות 64, (תרגילים) 64-65, 75
linked-list implementation of מימוש באמצעות רשימות מקושרות 59, (תרגילים) 61
finished vertex קדקוד שהסתיים 91
finite automaton אוטומט סופי 435
for string matching עבור התאמת מחרוזות 435-439
finite group חבורה סופית 391
first-fit heuristic היוריסטיקת המתאימה־הראשונה (בעיה) 538
flow זרימה 179-185
excess עודפת 201
integer-valued בעלת ערכים שלמים 198
sum סכום (תרגיל) 184
value of ערך של 179-180
flow conservation שימור זרימה 179
flow network רשת זרימה 179-185
corresponding to a bipartite graph המתאימה לגרף דו־צדדי 197
cut of חתך של 189-191
multicommodity ריבוי־מוצרים (תרגיל) 185

with negative capacities עם קיבולים שליליים (בעיה) 221
Floyd-Warshall algorithm אלגוריתם פלויד־וורשאל 159-163, (תרגילים) 164, (תרגיל) 175
Ford-Fulkerson method שיטת פורד־פולקרסון 185-196
forefather אב קדום ביותר 101
forest יער
depth-first עומק 91
disjoint-set קבוצות זרות 62-65
finding roots of, on a CREW PRAM מציאת שורשים של, על CREW PRAM 286-288
finding roots of, on an EREW PRAM מציאת שורשים של, על EREW PRAM (תרגיל) 292
formula-satisfiability problem בעיית ספיקות הנוסחאות 500-502
forward edge קשת קדימה 95
forward substitution הצבה קדימה 332-333
full adder סוכם מלא 246-247
full node צומת מלא 8
full rank דרגה מלאה 317
full walk of a tree סריקה מלאה של עץ 527
fully polynomial-time approximation scheme סכמת קירוב פולינומיאלית לחלוטין 522
for the maximum clique problem עבור בעיית הקליקה המקסימלית (בעיה) 538
for the subset-sum problem עבור בעיית סכום התת־קבוצה 533-537

G

Gabow's scaling algorithm for single-source shortest paths
האלגוריתם המדורג של גאבו עבור מסלולים קצרים ביותר ממקור יחיד (בעיה) 148
gap character תו מרווח (תרגיל) 429-430
gate שער 245
Gaussian elimination שיטת החילוץ של גאוס 335
gcd 382-383
see also greatest common divisor ראה גם: מחלק משותף גדול ביותר
generate a carry bit יצירת סיבית נשא 251
generator יוצר
of a subgroup של תת־חבורה 395
of $\mathbf{Z}_n^*$ של $\mathbf{Z}_n^*$ 403
generic preflow-push algorithm אלגוריתם כללי של דחיפת־קדם־זרימה 203-208
gift wrapping עטיפת מתנות 471
globally clocked circuit מעגל מבוקר־שעון גלובלית 266
Goldberg's algorithm, *see* preflow-push algorithm
האלגוריתם של גולדברג, ראה: אלגוריתם דחיפת־קדם־זרימה
good suffix function פונקציית הסיפא־הטובה 450

good suffix heuristic היוריסטיקת הסיפא־הטובה **450-452**
grade-school multiplication algorithm אלגוריתם הכפל של בית הספר היסודי **259**
Graham's scan הסריקה של גראהם **466-471**
graph גרף
  adjacency-list representation of ייצוג על־ידי רשימות סמיכות **80**
  adjacency-matrix representation of ייצוג על־ידי מטריצת סמיכויות **81**
  algorithms for אלגוריתמים על **79-222**
  breadth-first search of חיפוש לרוחב ב־ **83-90**
  complement of משלים של **508**
  dense צפוף **80**
  depth-first search of חיפוש לעומק ב־ **90-97**
  ϵ-dense ϵ־צפוף (בעיה) **176**
  hamiltonian המילטוני **487**
  incidence matrix of מטריצת קשירויות של (תרגיל) **83 ,510**
  nonhamiltonian לא־המילטוני **487**
  shortest path in מסלולים קצרים ביותר בתוך **86**
  sparse דליל **80**
  tour of מסלול סיור בתוך **517**
  weighted משוקלל **81**
  see also directed acyclic graph (dag), directed graph, flow network, undirected graph, tree
  ראה גם: גרף מכוון ללא מעגלים (גמ"ל), גרף מכוון, רשת זרימה, גרף בלתי־מכוון, עץ
graph-coloring problem בעיית צביעת־גרף (בעיה) **519**
gray vertex קדקוד אפור **84, 91**
greatest common divisor (gcd) מחלק משותף מקסימלי **382-383**
  computed by binary gcd algorithm חישוב על־ידי אלגוריתם gcd בינרי (בעיה) **423**
  computed by Euclid's algorithm חישוב על־ידי האלגוריתם של אוקלידס **385-390**
  with more than two arguments עם יותר משני ארגומנטים (תרגיל) **390**
  recursion theorem for משפט הרקורסיה עבור **386**
greedy algorithm אלגוריתם חמדן
  Dijkstra's algorithm האלגוריתם של דייקסטרה **131-135**
  for finding optimal vertex cover for a tree למציאת כיסוי קדקודים אופטימלי עבור עץ (תרגיל) **525**
  Kruskal's algorithm האלגוריתם של קרוסקל **113-115**
  for minimum spanning tree עבור עץ פורש מינימלי **109**
  Prim's algorithm האלגוריתם של פרים **115-117**
  for the set-covering problem עבור בעיית כיסוי הקבוצה **529-533**
  for the weighted set-covering problem עבור הבעיה המשוקללת של כיסוי קבוצה (בעיה) **538**
grid שריג (בעיה) **218**
group חבורה **390-396**
  cyclic מעגלית **403**

H

Hall's theorem משפט הול (תרגיל) **200**
halting problem בעיית עצירה **480**
halving lemma למת המחצית **365**
hamiltonian המילטוני
  cycle מעגל **487**
  graph גרף **487**
  path מסלול (תרגיל) **491**
hamiltonian-cycle problem בעיית המעגל ההמילטוני **488 ,512 -517**
hamiltonian-path problem בעיית המסלול ההמילטוני (תרגיל) **518**
hardwired חיבור בחומרה **250**
heap ערימה
  d-ary *d*־ית (בעיה) **176**
  in Dijkstra's algorithm באלגוריתם של דייקסטרה **134**
  to implement a mergeable heap למימוש ערימה מתמזגת **21**
  in Johnson's algorithm באלגוריתם של גינסון **169**
  mergeable מתמזגת **21**
  in Prim's algorithm באלגוריתם של פרים **117**
  relaxed נינוחה **55**
  running times of operations on זמני הריצה של פעולות על (איור) **22**
  2-3-4 (בעיה) **36**
heap order סדר הערימה **24**
height גובה
  of a binomial tree של עץ בינומי **23**
  of a B-tree B־ של עץ **9**
  of a Fibonacci heap של ערימת פיבונצ'י (תרגיל) **54**
height function, in preflow-push algorithms פונקציית גובה, באלגוריתמי דחיפת־קדם־זרימה **202**
Hermitian matrix מטריצה הרמיטית (תרגיל) **346**
Horner's rule כלל הורנר **358**

I

idempotence under an operator אידמפוטנטיות תחת אופרטור **170**
identity איבר יחידה **170, 326, 391**
identity matrix מטריצת יחידה **313**
implicit summation notation סימון הסיכום המובלע **183**
inactive processor מעבד לא פעיל (תרגיל) **296**
inadmissible edge קשת בלתי־מותרת **210**
incidence matrix מטריצת קשירויות

additive חיבורי **327**

in a group בחבורה **391**

of a matrix של מטריצה **316**

of a matrix from a LUP decomposition של מטריצה מפירוק LUP **343**

multiplicative כפלי **329**

multiplicative modulo n כפלי מודולו n **398**

inverse Ackermann's function (α) פונקציית אקרמן ההופכית **67**

inverter מהפך **245**

invertible matrix מטריצה הפיכה **316**

iterated logarithm function פונקציית הלוגריתם החוזר **65-66**

J

Jarvis's march הצעדה של ג'ארביס **471-472**

Johnson's algorithm האלגוריתם של ג'ונסון **166-169**

joining איחוד

of 2-3-4 trees של עצי 2-3-4 (בעיה) **19**

K

Karmarkar's algorithm האלגוריתם של קארמארקר **142, 151**

Karp's minimum mean-weight cycle algorithm

האלגוריתם של קארפ למציאת מעגל בעל משקל ממוצע מינימלי (בעיה) **149**

kernel of a polygon גרעין של מצולע (תרגיל) **473**

key מפתח

median, of a B-tree node חציון, של צומת בעץ־B **11**

public ציבורי **406, 409**

secret סודי **406, 409**

kill a carry bit איפוס סיבית נשא **251**

Kleene star (*) כוכב קליני **484**

KMP algorithm אלגוריתם KMP **440-446**

knot, of a spline נקודת קשר, של פונקציית ספליין (בעיה) **353-354**

Knuth-Morris-Pratt algorithm אלגוריתם קנות־מוריס־פראט **440-446**

KPG box קופסת KPG **255**

k-ruling set קבוצה k־שליטה (תרגיל) **307**

Kruskal's algorithm האלגוריתם של קרוסקל **113-115**

with integer edge weights עם משקלות קשתות שלמים (תרגיל) **117**

kth power חזקה k־ית (תרגיל) **385**

and difference constraints ואילוצי הפרשים **144**

of a directed graph של גרף מכוון (תרגיל) **83**

of an undirected graph של גרף בלתי־מכוון **510**

incremental design method for finding the convex hull שיטת הבנייה ההדרגתית עבור מציאת הקמור **465**

independent set קבוצה בלתי־תלויה **302,** (בעיה) **519**

index of an element of $\mathbf{Z}_n^*$ אינדקס של איבר ב־$\mathbf{Z}_n^*$ **403**

inner product מכפלה פנימית **315**

input קלט

to a combinational circuit למעגל צירופי **246**

to a logic gate לשער לוגי **245**

input alphabet אלפבית קלט **435**

input sequence סדרת קלט **228**

input wire תיל קלט **228**

insertion הכנסה

into binomial heaps לערימות בינומיות **32**

into B-trees לעצי־B **13-15**

into Fibonacci heaps לערימות פיבונצ'י **41-42**

into sweep-line statuses למצבי קו סריקה **461**

into 2-3-4 heaps לערימות 2-3-4 (בעיה) **36**

insertion sort מיון הכנסה

sorting-network implementation of מימוש על־ידי רשת מיון (תרגיל) **230**

instance of an abstract problem מופע של בעיה מופשטת **481**

integer-valued flow זרימה בעלת ערכים שלמים **198**

integrality theorem משפט הפתרון בשלמים **199**

intermediate vertex קדקוד ביניים **160**

interpolation by a cubic spline אינטרפולציה באמצעות פונקציית ספליין ממעלה שלישית (בעיה) **353**

interpolation by a polynomial אינטרפולציה באמצעות פולינום **359**

at complex roots of unity בשורשי היחידה המרוכבים **368-369**

intersection חיתוך

determining, for a set of line segments קביעה, עבור קבוצת קטעי ישרים **459-464**

determining, for two disks קביעה, עבור שני עיגולים (תרגיל) **464**

determining, for two line segments קביעה, עבור שני קטעי ישרים **457-458**

determining, for two simple polygons קביעה, עבור שני מצולעים פשוטים (תרגיל) **464**

finding all, in a set of line segments מציאת כולם, בקבוצה של קטעי ישרים (תרגיל) **464**

of languages של שפות **484**

intractable לא ניתן לפתרון סביר **480**

invalid shift היסט בלתי־תקף **426**

inverse הופכי

L

labeling function in path calculus פונקציית שיוך תוויות בתחשיב מסלולים **170**

Lagrange's formula נוסחת לאגראנז' **360**

Lagrange's theorem משפט לאגראנז' **394**

Lamé's theorem משפט לאמה **388**

language שפה **484**

completeness of שלמות של (תרגיל) **499**

proving NP-completeness of הוכחת שלמות ב־NP של **499-500**

verification of אימות של **489**

last-occurrence function פונקציית המופע האחרון **450**

layers שכבות

convex קמורות (בעיה) **477**

maximal מקסימליות (בעיה) **478**

lcm (least common multiple) כפולה משותפת מינימלית (תרגיל) **390**

leading submatrix תת־מטריצה מובילה **347**

least common ancestor אב קדמון משותף עמוק ביותר (בעיה) **74**

least common multiple כפולה משותפת מינימלית (תרגיל) **390**

least-squares approximation קירוב על־ידי ריבועים פחותים **348-351**

Legendre symbol $\left(\frac{a}{p}\right)$ סימן לז'נדאר (בעיה) **424**

lg* (iterated logarithm function) פונקציית הלוגריתם החוזר

and repeated exponentiation והעלאה חוזרת בחזקה **66**

lifted vertex קדקוד מורם **203**

lift operation פעולת הרמה **203, 207**

lift-to-front algorithm אלגוריתם הרם־וקדם **209-218**

light edge קשת קלה **110**

linear-array multiplier כופל מערך לינארי **268-271**

linear dependence תלות לינארית **316**

linear equations משוואות לינאריות

solving modular פתרון משוואות מודולריות **396-399**

solving systems of פתרון מערכת של **330-342**

solving tridiagonal systems of פתרון מערכת תלת־אלכסונית של (בעיה) **353**

linear independence אי־תלות לינארית **316**

linear programming תכנון־לינארי

and maximum flow וזרימה מקסימלית (תרגיל) **185**

and multicommodity flow וזרימה של ריבוי מוצרים (תרגיל) **185**

and single source shortest paths ומסלולים קצרים ביותר ממקור יחיד **142-147**

line segment קטע ישר **455**

determining turn of קביעת כיוון הפנייה של **456**

determining whether any intersect קביעה אם קיימים קטעים נחתכים **459-464**

determining whether two intersect קביעה אם שניים נחתכים **457-458**

finding all intersections of, in a set מציאת כל החיתוכים, בקבוצה (תרגיל) **464**

link קישור

of binomial trees של עצים בינומיים **22**

of Fibonacci-heap roots של שורשים בערימת פיבונצ'י **44**

linked list רשימה מקושרת

to implement disjoint sets למימוש קבוצות זרות **58-61**

k-ruling set of קבוצה k־שליטה של (תרגיל) **307**

list ranking in parallel מיקום ברשימה בחישוב מקבילי **278-280**

literal ליטרל **502**

L_m-distance מרחק־L_m (תרגיל) **477**

logic gate שער לוגי **245**

longest-simple-cycle problem בעיית המעגל הפשוט הארוך ביותר (תרגיל) **518**

lower-triangular matrix מטריצה משולשית תחתונה **313**

LU decomposition LU פירוק **335-338**

LUP decomposition LUP פירוק **332**

computation of חישוב **338-341**

of a diagonal matrix של מטריצה אלכסונית (תרגיל) **342**

in matrix inversion בהיפוך מטריצות **343**

and matrix multiplication וכפל מטריצות (תרגיל) **346**

of a permutation matrix של מטריצת תמורה (תרגיל) **342**

use of השימוש ב־ **332-335**

M

main memory זיכרון ראשי **6**

majority function פונקציית הרוב **246,** (בעיה) **272**

MAKE-SET MAKE-SET פעולת **56**

disjoint-set-forest implementation of מימוש על־ידי יער קבוצות זרות **63**

linked-list implementation of מימוש על־ידי רשימה מקושרת **59,** (תרגילים) **61**

Manhattan distance מרחק מנהטן (תרגיל) **477**

marked node צומת מסומן **40, 49-51,** (תרגיל) **51**

matched vertex קדקוד מזווג **197**

matching זיווג

and maximum flow וזרימה מקסימלית **196-200**

of strings מחרוזות **426-453**

weighted bipartite בגרף דו־צדדי משוקלל **55**

matrix מטריצה **311-319**

adjacency סמיכויות **81**
conjugate transpose of מוחלפת צמודה (תרגיל) **346**
Hermitian הרמיטית (תרגיל) **346**
incidence קשירויות (תרגיל) **83, 510**
predecessor קודמים **153**
pseudoinverse of פסאודו־הופכית **350**
and quasirings ומעין־חוגים **327**
and rings וחוגים **327-328**
symmetric positive-definite סימטרית חיובית־לחלוטין **346-348**
Toeplitz טפליץ (בעיה) **375**
transpose of מוחלפת **81, 312**
see also matrix inversion, matrix multiplication ראה גם: היפוך מטריצות, כפל מטריצות
matrix inversion היפוך מטריצות **342-346**
matrix multiplication כפל מטריצות
for all-pairs shortest paths עבור מסלולים קצרים ביותר בין כל הזוגות **154-159**
boolean בוליאניות **328-329,** (תרגיל) **330,** (תרגיל) **346,** (בעיה) **352**
and computing the determinant וחישוב הדטרמיננטה (תרגיל) **346**
on a CRCW PRAM CRCW PRAM על (תרגיל) **292**
on an EREW PRAM EREW PRAM על (תרגיל) **292**
and LUP decomposition LUP ופירוק (תרגיל) **346**
and matrix inversion והיפוך מטריצות **343-345**
Pan's method שיטת פאן (תרגיל) **325**
Strassen's algorithm האלגוריתם של שטראסן **320-326**
by a work-efficient CRCW algorithm באמצעות אלגוריתם CRCW יעיל בעבודה (תרגיל) **296**
matrix-vector multiplication כפל מטריצה בווקטור (תרגיל) **296**
max-flow min-cut theorem המשפט זרימה־מקסימלית חתך־מינימלי **190-191**
maximal מקסימלי
layers שכבות (בעיה) **478**
point נקודה (בעיה) **478**
maximal independent set קבוצה בלתי־תלויה מקסימלית **302**
computed from a 6-coloring on an EREW PRAM EREW PRAM חישוב מתוך צביעת־6 על **306**
maximum bipartite matching זיווג מקסימלי בגרף דו־צדדי **196-200,** (תרגיל) **209, 222**
maximum degree in a Fibonacci heap דרגה מקסימלית בערימת פיבונצ'י **40,** (תרגיל) **48, 52-54**
maximum flow זרימה מקסימלית **178-222**
Ford-Fulkerson method שיטת פורד־פולקרסון **185-196**
lift-to-front algorithm אלגוריתם הרם־וקדם **209-219**
and maximum bipartite matching וזיווג מקסימלי בגרף דו־צדדי **196-200**
preflow-push algorithms אלגוריתמי דחיפת־קדם־זרימה **200-218**

scaling algorithm אלגוריתם מדורג (בעיה) **220**
updating עדכון (בעיה) **220**
with negative capacities עם קיבולים שליליים (בעיה) **221**
mean weight of a cycle משקל ממוצע של מעגל (בעיה) **149**
median key of a B-tree node מפתח חציון של צומת בעץ־B **11**
merge מיזוג
on a CRCW PRAM CRCW PRAM על (תרגיל) **292**
using a comparison network תוך שימוש ברשת השוואה **237-239**
mergeable heap ערימה מתמזגת **21**
in minimum-spanning-tree algorithm באלגוריתם למציאת עץ פורש מינימלי (בעיה) **37**
relaxed heaps ערימות נינוחות **55**
running times of operations on זמני ריצה של פעולות על (איור) **22**
2-3-4 heaps ערימות 2-3-4 (בעיה) **36**
see also binomial heap, Fibonacci heap ראה גם: ערימה בינומית, ערימת פיבונצ'י
merge sort מיון־מיזוג
sorting network implementation of מימוש על־ידי רשת מיון **239-241**
merging network רשת מיזוג **237-239**
odd-even זוג־פרד (בעיה) **242**
Miller-Rabin primality test מבחן הראשוניות של מילר ורבין **414-418**
minimax weight משקל מינימקסימלי (בעיה) **177**
minimum מינימום
in binomial heaps בערימות בינומיות **27**
in B-trees בעצי־B (תרגיל) **15**
in Fibonacci heaps בערימות פיבונצ'י **42**
off-line בלתי־מקוון (בעיה) **73**
minimum-cost spanning tree, *see* minimum spanning tree
עץ פורש בעל עלות מינימלית, ראה: עץ פורש מינימלי
minimum degree of a B-tree דרגה מינימלית של עץ־B **8**
minimum key מפתח מינימלי
in 2-3-4 heaps בערימות 2-3-4 (בעיה) **36**
minimum mean-weight cycle מעגל בעל משקל ממוצע מינימלי (בעיה) **149**
minimum node of a Fibonacci heap צומת המינימום בערימת פיבונצ'י **40**
minimum path cover כיסוי מינימלי על־ידי מסלולים (בעיה) **219**
minimum spanning tree עץ פורש מינימלי **108-119**
as a closed semiring כחוג־למחצה סגור (בעיה) **177**
constructed using mergeable heaps בנייה באמצעות ערימות מתמזגות (בעיה) **37**
on dynamic graphs על גרפים דינמיים (תרגילים) **118**
in finding a near-optimal traveling-salesman tour

במציאת מסלול כמעט אופטימלי לבעיית הסוכן הנוסע **525**
generic algorithm אלגוריתם כללי **109-112**
Kruskal's algorithm האלגוריתם של קרוסקל **113-115**
Prim's algorithm האלגוריתם של פרים **115-117**
second-best שני הכי טוב (בעיה) **118**
minor of a matrix מינור של מטריצה **317**
MIS (maximal independent set) קבוצה בלתי־תלויה מקסימלית **302**
computed from a 6-coloring on an EREW PRAM חישוב מתוך צביעת־6 על EREW PRAM **306**
modular arithmetic אריתמטיקה מודולרית **390-396**
modular exponentiation העלאה מודולרית בחזקה **404**
modular linear equations משוואות לינאריות מודולריות **396-399**
monoid מונואיד **170, 326**
multicommodity flow network רשת זרימה מרובת מוצרים (תרגיל) **185**
multigraph מולטי־גרף
converting to equivalent undirected graph המרה לגרף בלתי־מכוון שקול (תרגיל) **83**
multiple כפולה **314, 380**
of an element modulo n של איבר מודולו n **396-399**
multiple sources and sinks ריבוי מקורות ובורות **182-183**
multiplication כפל
array multiplier כופל מערכי **259-262**
circuits for מעגלים עבור **259-265, 268-271**
of complex numbers של מספרים מרוכבים (תרגיל) **326**
divide-and-conquer method for שיטת הפרד ומשול עבור (בעיה) **375**
grade-school algorithm for אלגוריתם בית הספר היסודי עבור **259**
linear-array multiplier כופל מערך־לינארי **268-271**
of matrices מטריצות **315**
modulo n $(\cdot_n)$ מודולו n **391**
of polynomials פולינומים **357**
Russian peasant's algorithm for אלגוריתם האיכר הרוסי עבור **268**
Wallace-tree multiplier כופל עץ וואלאס **262-264**
multiplicative group modulo n חבורה כפלית מודולו n **392**
multiplicative inverse הופכי כפלי **329**
modulo n מודולו n **398**

N

naive algorithm for string matching האלגוריתם הנאיבי להתאמת מחרוזות **428-430**
NAND gate שער NAND **245**
negative שלילי

in an algebraic system במערכת אלגברית **327**
of a matrix של מטריצה **314**
negative-weight cycle מעגל בעל משקל שלילי
detection of, by all-pairs shortest-paths algorithm
גילוי של, על־ידי אלגוריתם למציאת מסלולים קצרים ביותר בין כל הזוגות (תרגיל) **159**
detection of, by Floyd-Warshall algorithm גילוי של, על־ידי אלגוריתם פלויד־וורשאל **165**
and difference constraints ואילוצי הפרשים **145**
finding מציאה של (תרגיל) **139,** (תרגיל) **159**
and relaxation והקלה (תרגיל) **131**
and shortest paths ומסלולים קצרים ביותר **121**
negative-weight edge קשת בעלת משקל שלילי
in Dijkstra's algorithm באלגוריתם של דייקסטרה (תרגיל) **135**
and shortest paths ומסלולים קצרים ביותר **121-122**
neighborhood סביבה (תרגיל) **200**
neighbor list רשימת שכנים **211**
nesting boxes קופסאות מקוננות (בעיה) **148**
net flow זרימה נטו **179-180**
across a cut דרך חתך **189**
network רשת
admissible מותרת **210**
bitonic sorting מיון ביטונית **234-237**
comparison השוואה **227-231**
flow, *see* flow network זרימה, ראה: רשת זרימה
for merging עבור מיזוג **237-239**
odd-even merging מיזוג זוג־פרד (בעיה) **242**
odd-even sorting מיון זוג־פרד (בעיה) **241**
permutation תמורה (בעיה) **242**
residual שיורית **186-188**
sorting מיון **227-243**
transposition חילוף (בעיה) **241**
Newton iteration איטרציית ניוטון (בעיה) **271**
nondeterministic polynomial time זמן פולינומיאלי לא דטרמיניסטי (הערה) **489**
see also NP ראה גם: NP
nonhamiltonian graph גרף לא־המילטוני **487**
noninvertible matrix מטריצה בלתי־הפיכה **316**
nonoverlappable string pattern תבנית מחרוזת נטולת חפיפות (תרגיל) **439**
nonsaturating push דחיפה בלתי־מרווה **203, 208**
nonsingular matrix מטריצה רגולרית **316**

nontrivial power חזקה לא טריוויאלית (תרגיל) **385**
nontrivial square root of 1, modulo n שורש ריבועי לא טריוויאלי של 1, מדולו n **404**
NOR gate שער NOR **245**
normal equation משוואה נורמלית **350**
norm of a vector נורמה של וקטור **315-316**
NOT function (¬) פונקציית השלילה **245**
NOT gate שער NOT **245**
NP (complexity class) NP (מחלקת סיבוכיות) **489,** (תרגילים) **490**
NPC (complexity class) NPC (מחלקת סיבוכיות) **493-494**
NP-complete NP־שלם **493**
NP-completeness שלמות ב־NP **480-520**
of the circuit-satisfiability problem של בעיית ספיקות המעגלים **494-498**
of the clique problem של בעיית הקליקה **506-508**
of determining whether a boolean formula is a tautology
של הקביעה אם נוסחה בוליאנית היא טאוטולוגיה (תרגיל) **505**
of the formula-satisfiability problem של בעיית ספיקות הנוסחאות **500-502**
of the graph-coloring problem של בעיית צביעת הגרף (בעיה) **519**
of the hamiltonian-cycle problem של בעיית המעגל ההמילטוני **512-517**
of the hamiltonian-path problem של בעיית המסלול ההמילטוני (תרגיל) **518**
of the independent-set problem של בעיית הקבוצה הבלתי־תלויה (בעיה) **519**
of the longest-simple-cycle problem של בעיית המעגל הפשוט הארוך ביותר (תרגיל) **518**
proving, of a language הוכחה של, עבור שפה **499-500**
of the set-covering problem של בעיית כיסוי הקבוצה (תרגיל) **533**
of the set-partition problem של בעיית חלוקת הקבוצה (תרגיל) **518**
of the subgraph-isomorphism problem של בעיית האיזומורפיות של תת־גרף (תרגיל) **518**
of the subset-sum problem של בעיית סכום תת־קבוצה **510-512**
of the 3-CNF-satisfiability problem של בעיית הספיקות של 3-CNF **502-505**
of the traveling-salesman problem של בעיית הסוכן הנוסע **517-518**
of the vertex-cover problem של בעיית הכיסוי על־ידי קדקודים **508-509**
of the 0-1 integer-programming problem של גרסת 0-1 של בעיית התכנון בשלמים (תרגיל) **518**
NP-hard קשה־NP **493**
null vector וקטור מָאפֵס **317**
numerical instability אי־יציבות מספרית **331**

O

objective function פונקציית מטרה **142,** (תרגיל) **147**
occurrence of a pattern מופע של תבנית **426**
odd-even merging network רשת מיזוג זוג־פרד (בעיה) **242**

odd-even sorting network רשת מיון זוג־פרד (בעיה) **241**
off-line problem בעיה בלתי־מקוונת
least common ancestors אבות קדמונים משותפים עמוקים ביותר (בעיה) **74**
minimum מינימום (בעיה) **73**
$\bar{1}$ **170, 326**
one-pass method שיטת המעבר היחיד **75**
one-way hash function פונקציית גיבוב חד־כיוונית **410**
on-line convex-hull problem בעיית הקמור המקוונת (תרגיל) **473**
optimal substructure תת־מבנה אופטימלי
of shortest paths של מסלולים קצרים ביותר **124-125, 154, 160**
optimal vertex cover כיסוי אופטימלי על־ידי קדקודים **523**
optimization problem בעיית אופטימיזציה **482**
approximation algorithms for אלגוריתמי קירוב עבור **521-539**
OR function (∨) OR פונקציית **245, 327**
on a CRCW PRAM CRCW PRAM על (תרגיל) **292**
order of a group סדר של חבורה **395**
OR gate שער OR **245**
origin נקודת הראשית **455**
outer product מכפלה חיצונית **315**
output פלט
of a combinational circuit של מעגל צירופי **246**
of a logic gate של שער לוגי **245**
output sequence סדרת פלט **228**
output wire תיל פלט **228**
overdetermined system of linear equations מערכת משוואות לינאריות מוגדרת־יתר **331**
overflowing vertex קדקוד גולש **201**
overlapping-suffix lemma למת הסיפות החופפות **427**

P

P (complexity class) P (מחלקת סיבוכיות) **483, 485,** (תרגיל) **487**
package wrapping עטיפת חבילות **471**
page on a disk דף על דיסק **6,** (בעיה) **19**
pairwise relatively prime זרים בזוגות **383**
Pan's method of matrix multiplication שיטת פאן לכפל מטריצות (תרגיל) **325**
parallel algorithm אלגוריתם מקבילי **274-310**
see also arithmetic circuit, comparison network, CRCW, CREW, EREW, parallel random-access machine, sorting network
ראה גם: מעגל אריתמטי, רשת השוואה, CRCW, CREW, EREW, מכונת גישה אקראית מקבילית, רשת מיון

in finding a near optimal traveling-salesman four
במציאת מסלול כמעט אופטימלי עבור בעיית הסוכן הנוסע **525**
implemented with a Fibonacci heap מימוש עם ערימת פיבונצ'י **117**
implemented with a heap מימוש עם ערימה **117**
with integer edge weights עם משקלות קשתות שלמים (תרגיל) **117**
similarity to Dijkstra's algorithm דמיון לאלגוריתם של דייקסטרה **115, 135**
for sparse graphs עבור גרפים דלילים (בעיה) **118**
primality testing מבחן ראשוניות **411-418, 424-425**
Miller-Rabin test מבחן מילר־רבין **414-418**
pseudoprimality testing בדיקת פסאודו־ראשוניות **413**
primary memory זיכרון ראשי **6**
prime distribution function פונקציית ההתפלגות של הראשוניים **412**
prime number מספר ראשוני **380**
density of צפיפות של **412**
prime number theorem משפט המספרים הראשוניים **412**
primitive root of $\mathbf{Z}_n^*$ שורש פרימיטיבי של $\mathbf{Z}_n^*$ **403**
principal root of unity שורש ראשי של היחידה **363**
priority queue תור קדימויות
in Dijkstra's algorithm באלגוריתם של דייקסטרה **134**
in Prim's algorithm באלגוריתם של פרים **115, 116-117**
see also binary search tree, binomial heap, Fibonacci heap
ראה גם: עץ חיפוש בינרי, ערימה בינומית, ערימת פיבונצ'י
probabilistic analysis ניתוח הסתברותי
of boolean matrix multiplication של כפל מטריצות בוליאניות (בעיה) **352**
of carry propagation in a ripple-carry adder של קידום נשא בסוכם מגלגל־נשא (תרגיל) **259**
of convex hull over a sparse-hulled distribution של קמור מעל התפלגות בעלת קמור דליל (בעיה) **479**
of file comparison של השוואת קבצים (תרגיל) **434**
of Miller-Rabin primality test של מבחן הראשוניות של מילר ורבין **415-418**
of Pollard's rho heuristic של היוריסטיקת רו של פולארד **420-422,** (תרגילים) **422**
of Rabin-Karp algorithm של אלגוריתם רבין־קארפ **433**
of work-efficient parallel prefix algorithm
של אלגוריתם מקבילי יעיל־בעבודה לחישוב רישות **299-301,** (תרגילים) **301**
problem בעיה
abstract מופשטת **481**
concrete קונקרטית **482**
decision הכרעה **482**
intractable לא ניתנת לפתרון סביר **480**
optimization אופטימיזציה **482**

power חזקה
of an element modulo n n של איבר מודולו **402-406**
kth k־ית (תרגיל) **385**
nontrivial לא טריוויאלית (תרגיל) **385**
PRAM (parallel random-access machine) (מכונת גישה אקראית מקבילית) PRAM
274-277, 295, (תרגילים) **296**
see also CRCW, CREW, ERCW, EREW ראה גם:
predecessor קודם
in breadth-first trees בעצי חיפוש לרוחב **84**
in B-trees בעצי B־ (תרגיל) **15**
in shortest-paths trees בעצי מסלולים קצרים ביותר **122**
predecessor matrix מטריצת קודמים **153**
predecessor subgraph תת־גרף קודמים
in all-pairs shortest paths במסלולים קצרים ביותר בין כל הזוגות **153**
in breadth-first search בחיפוש לרוחב **88-89**
in depth-first search בחיפוש לעומק **91**
in single-source shortest paths במסלולים קצרים ביותר ממקור יחיד **122-123**
prefix of a string ($\prec$) רישא של מחרוזת **427**
prefix computation חישוב רישות
in carry-lookahead addition בחיבור צופה־נשא **252**
on an EREW PRAM EREW PRAM על **280-282,** (תרגיל) **296**
on a list על רשימה **280-281**
by a parallel prefix circuit על־ידי מעגל רישות מקבילי **253-255**
work-efficient version on an EREW PRAM גרסה יעילה בעבודה על EREW PRAM **296-301**
prefix function פונקציית הרישא **440-442**
prefix-function iteration lemma למת ההפעלה החוזרת של פונקציית הרישא **443**
preflow קדם־זרימה **201**
preflow-push algorithm אלגוריתם דחיפת־קדם־זרימה **200-218, 222**
basic operations in פעולות בסיסיות ב־ **202-203**
by discharging an overflowing vertex of maximum height
על־ידי פריקת קדקוד גולש שגובהו מקסימלי (תרגיל) **218**
to find a maximum bipartite matching למציאת זיווג מקסימלי בגרף דו־צדדי (תרגיל) **209**
generic algorithm אלגוריתם כללי **203-208**
lift-to-front algorithm אלגוריתם הרם־וקדם **209-218**
with a queue of overflowing vertices עם תור של קדקודים גולשים (תרגיל) **218**
presorting מיון מקדים **476**
Prim's algorithm האלגוריתם של פרים **115-117**
with an adjacency matrix עם מטריצת סמיכויות (תרגיל) **117**

solution to פתרון ל־ **482**

tractable ניתנת לפתרון סביר **480**

product מכפלה

cross וקטורית **455**

inner פנימית **315**

of matrices של מטריצות **315**

outer חיצונית **315**

partial חלקית **259**

of polynomials של פולינומים **357**

scalar flow סקלרית של זרימה (תרגיל) **184–185**

propagation delay זמן השהיה **246**

proper subgroup תת־חבורה ממש **394**

prune-and-search method שיטת גיזום־וחיפוש **465**

pruning a Fibonacci heap גיזום של ערימת פיבונצ'י (בעיה) **55**

pseudoinverse פסאודו־הופכי **350**

pseudoprime פסאודו־ראשוני **413**

public key מפתח ציבורי **406, 409**

public-key cryptosystem מערכת הצפנה מבוססת מפתח ציבורי **406–411**

push operation (in preflow-push algorithms) פעולת דחיפה (באלגוריתמי דחיפת־קדם־זרימה) **203**

Q

quadratic residue שארית ריבועית (בעיה) **423**

quadratic-sieve factorization algorithm אלגוריתם הנפה־הריבועית עבור פירוק לגורמים **425**

quasiring מעין־חוג **326–327**

queue תור

in breadth-first search בחיפוש לרוחב **84–85**

in preflow-push algorithms באלגוריתמי דחיפת־קדם־זרימה (תרגיל) **218**

quotient מנה **381**

R

Rabin-Karp algorithm אלגוריתם רבין־קארפ **430–434**

random-access machine מכונת גישה־אקראית

parallel מקבילית **274–276, 295,** (תרגילים) **296**

vs. comparison networks לעומת רשתות השוואה **227**

randomized algorithm אלגוריתם אקראי

for boolean matrix multiplication עבור כפל מטריצות בוליאניות (בעיה) **352**

coin flipping in הטלת מטבע ב־ **299**

Miller-Rabin primality test מבחן הראשוניות של מילר ורבין **414–418**

Pollard's rho heuristic היוריסטיקת רו של פולארד **419–422**

for work-efficient parallel prefix עבור חישוב רישות מקבילי יעיל־בעבודה **296–301**

rank דרגה

column של עמודות **316**

full מלאה **317**

of a matrix של מטריצה **316**

of a node in a disjoint-set forest של צומת ביער קבוצות־זרות **62, 68–69**

row של שורות **316**

ratio bound חסם יחס **521**

red-black tree עץ אדום־שחור

comparison with B-trees השוואה לעצי־B **9**

in determining whether any line segments intersect בקביעה אם ישנם קטעי ישרים נחתכים **461**

and 2-3-4 trees ועצי 2-3-4 (תרגיל) **10**

reducibility רדוקציה **492–493**

reduction algorithm אלגוריתם רדוקציה **492**

reduction function פונקציית רדוקציה **492**

register אוגר **266**

rejection דחייה

by an algorithm על־ידי אלגוריתם **485**

by a finite automaton על־ידי אוטומט סופי **435**

relative error שגיאה יחסית **521**

relative error bound חסם שגיאה יחסית **522**

relatively prime זרים זה לזה **383**

relaxation הקלה **125–131**

relaxed heap ערימה נינוחה **55**

remainder שארית **381**

repeated exponentiation העלאה חוזרת בחזקה **65–66**

repeated squaring העלאה חוזרת בריבוע

for all-pairs shortest paths עבור מסלולים קצרים ביותר בין כל הזוגות **157–158**

for raising a number to a power עבור העלאת מספר בחזקה **404**

repetition factor of a string גורם חזרה במחרוזת (בעיה) **452**

representative of a set נציג של קבוצה **56**

residual capacity קיבול שיורי **186, 188**

residual edge קשת שיורית **186**

residual network רשת שיורית **186–188**

residue שארית **381,** (בעיה) **423**

responsible processor מעבד אחראי **278**

reweighting שקלול מחדש

searching חיפוש
in B-trees בעצי־B **10-11**
search tree, *see* balanced search tree, binary search tree, B-tree, red-black tree, splay tree, 2-3 tree, 2-3-4 tree
עץ חיפוש, ראה: עץ חיפוש מאוזן, עץ חיפוש בינרי, עץ־B, עץ אדום־שחור, עץ splay, עץ 2-3, עץ 2-3-4
secondary storage התקן אחסון משני
search tree for עץ חיפוש עבור **5-20**
stacks on מחסניות על (בעיה) **19**
second-best minimum spanning tree עץ פורש מינימלי שני הכי טוב (בעיה) **118**
secret key מפתח סודי **406, 409**
segment, *see* directed segment, line segment קטע, ראה: קטע מכוון, קטע ישר
segmented parallel prefix חישוב רישות מקבילי מקוטע (בעיה) **307**
semiconnected graph גרף קשיר־למחצה (תרגיל) **105**
sequence ⟨⟩ סדרה **228, 234**
set {} קבוצה
independent בלתי־תלויה **302,** (בעיה) **519**
ruling שליטה (תרגיל) **307**
set-covering problem בעיית כיסוי הקבוצה **529-533**
weighted המשוקללת (בעיה) **538**
set-partition problem בעיית חלוקת הקבוצה (תרגיל) **518**
settling of circuit values התייצבות ערכים במעגל **246**
shadow of a point צל של נקודה (תרגיל) **473**
Shamir's boolean matrix multiplication algorithm האלגוריתם של שמיר לכפל מטריצות בוליאניות (בעיה) **352**
shifter מעגל הזזה (תרגיל) **265**
shift in string matching היסט בהתאמת מחרוזות **426**
shortest paths מסלולים קצרים ביותר **120-177**
all-pairs בין כל הזוגות **152-177**
Bellman-Ford algorithm אלגוריתם בלמן־פורד **136-139**
and breadth-first search וחיפוש לרוחב **86-88, 120**
and difference constraints ואילוצי הפרשים **142-147**
Dijkstra's algorithm האלגוריתם של דייקסטרה **131-135**
on a directed acyclic graph על גרף מכוון ללא מעגלים **139-141**
in ϵ-dense graphs בגרפים ϵ־צפופים (בעיה) **176**
estimate of אומדן של **125**
Floyd-Warshall algorithm אלגוריתם פלויד־וורשאל **160-163**
Gabow's scaling algorithm האלגוריתם המדורג של גאבו (בעיה) **148**
Johnson's algorithm האלגוריתם של ג'ונסון **166-169**
by matrix multiplication באמצעות כפל מטריצות **154-159**
and negative-weight cycles ומעגלים בעלי משקל שלילי **121**
in all-pairs shortest paths במסלולים קצרים ביותר בין כל הזוגות **166**
in single-source shortest paths במסלולים קצרים ביותר ממקור יחיד (בעיה) **148**
rho heuristic היוריסטיקת רו **419-422**
ring חוג **327-328**
ripple-carry addition חיבור מגלגל־נשא **249-250**
compared to bit-serial addition בהשוואה לחיבור סיבית סדרתי **267**
root שורש
of unity של היחידה **363**
of $\mathbf{Z}_n^*$ של $\mathbf{Z}_n^*$ **403**
root list רשימת שורשים
of a binomial heap של ערימה בינומית **25**
of a Fibonacci heap של ערימת פיבונצ'י **40**
rotational sweep סריקה סיבובית **465-472**
row rank דרגת שורות **316**
row vector וקטור שורה **312**
RSA public-key cryptosystem מערכת ההצפנה RSA מבוססת מפתח ציבורי **406-411**
ruling set קבוצה שליטה (תרגיל) **307**
running time זמן ריצה
of a combinational circuit של מעגל צירופי **248**
of a comparison network של רשת השוואה **229**
of a graph algorithm של אלגוריתם על גרף **79**
Russian peasant's algorithm אלגוריתם האיכר הרוסי **268**

S

safe edge קשת בטוחה **109**
SAT SAT **500**
satisfiability ספיקות **494, 500-502**
satisfiable formula נוסחה ספיקה **500**
satisfying assignment השמה מספקת **494, 500**
saturated edge קשת רוויה **203**
saturating push דחיפה מרווה **203, 207**
scalar flow product מכפלה סקלרית של זרימה (תרגיל) **184-185**
scalar multiple כפולה סקלרית **314**
scaling דירוג
in maximum flow בזרימה מקסימלית (בעיה) **200**
in single-source shortest paths במסלולים קצרים ביותר ממקור יחיד (בעיה) **148**
Schur complement משלים־שור **336, 347**
Schur complement lemma למת משלים־שור **348**

with negative-weight edges עם קשתות בעלות משקל שלילי **121-122**
optimal substructure of תת־מבנה אופטימלי של **124-125**
problem variants גרסאות של הבעיה **121**
and relaxation והקלה **125-128**
by repeated squaring באמצעות העלאה חוזרת בריבוע **157-158**
single-source ממקור יחיד **120-151**
tree of עץ של **123, 128-130**
in an unweighted graph בגרף בלתי משוקלל **86**
in a weighted graph בגרף משוקלל **120**
similar (~) דומה **450**
simple polygon מצולע פשוט (תרגיל) **464**
simplex סימפלקס (הערה) **142**
simplex algorithm אלגוריתם הסימפלקס **142, 151**
simulation, work-efficient הדמיה, יעילה בעבודה **295**
single-destination shortest paths מסלולים קצרים ביותר ליעד יחיד **121**
single-pair shortest path מסלול קצר ביותר בין זוג קדקודים אחד **121,** (תרגיל) **146**
single-source shortest paths מסלולים קצרים ביותר ממקור יחיד **120-151**
Bellman-Ford algorithm אלגוריתם בלמן־פורד **136-139**
and difference constraints ואילוצי הפרשים **142-147**
Dijkstra's algorithm האלגוריתם של דייקסטרה **131-135**
on a directed acyclic graph על גרף מכוון ללא מעגלים **139-141**
in ϵ-dense graphs בגרפים ϵ־צפופים (בעיה) **176**
Gabow's scaling algorithm האלגוריתם המדורג של גאבו (בעיה) **148**
singly connected graph גרף חד־קשיר (תרגיל) **97**
singular matrix מטריצה סינגולרית **316**
sink בור (תרגיל) **83, 179, 182-183**
size גודל
of an algorithm's input של קלט לאלגוריתם **379-380, 482-484**
of a binomial tree של עץ בינומי **23**
of a clique של קליקה **506**
of a combinational circuit של מעגל צירופי **248, 293**
of a disjoint-set tree של עץ קבוצות־זרות **68**
of a subtree in a Fibonacci heap של תת־עץ בערימת פיבונצ'י **52**
of a vertex cover של כיסוי על־ידי קדקודים **508, 523**
skew symmetry סימטריה נגדית **179**
solution פתרון
to an abstract problem לבעיה מופשטת **481**
to a concrete problem לבעיה קונקרטית **482**

feasible אפשרי **142**
to a system of linear equations למערכת משוואות לינאריות **331**
sorting מיון
on an EREW PRAM על EREW PRAM **295**
of points by polar angle של נקודות על־פי זוויות קוטביות (תרגיל) **458**
using networks, *see* sorting network באמצעות רשתות, ראה: רשת מיון
sorting network רשת מיון **230,** (תרגילים) **230**
AKS AKS **243**
based on insertion sort המבוססת על מיון־הכנסה (תרגיל) **230**
based on merge sort המבוססת על מיון־מיזוג **239-241**
bitonic ביטונית **234-237**
odd-even זוג־פרד (בעיה) **241**
source מקור **83, 121, 179, 182-183**
spanning tree עץ פורש **108**
sparse graph גרף דליל **80**
sparse-hulled distribution התפלגות בעלת קמור דליל (בעיה) **479**
splay tree splay עץ **4**
spline פונקציית ספליין (בעיה) **353**
splitting פיצול
of B-tree nodes של צמתים בעץ־B **11-13**
of 2-3-4 trees של עצי 2-3-4 (בעיה) **19**
spurious hit פגיעה מדומה **431**
square matrix מטריצה ריבועית **312**
square of a directed graph ריבוע של גרף מכוון (תרגיל) **83**
square root, modulo a prime שורש ריבועי, מודולו מספר ראשוני (בעיה) **423**
squaring, repeated העלאה חוזרת בריבוע
for all-pairs shortest paths עבור מסלולים קצרים ביותר בין כל הזוגות **157-158**
for raising a number to a power עבור העלאת מספר בחזקה **404**
stability of circuit values יציבות של ערכים במעגל **246**
stack מחסנית
in Graham's scan בסריקה של גראהם **466**
on secondary storage על התקן אחסון משני (בעיה) **19**
star-shaped polygon מצולע בעל צורת כוכב (תרגיל) **473**
star tree עץ כוכב (בעיה) **308**
start state מצב התחלתי **435**
state of a finite automaton מצב של אוטומט סופי **435**
Strassen's algorithm האלגוריתם של שטראסן **320-326**
and boolean matrix multiplication וכפל מטריצות בוליאניות **328-329,** (בעיה) **352**

string מחרוזת **426**

string matching התאמת מחרוזות **426–453**

based on repetition factors המבוססת על גורמי חזרה (בעיה) **452**

Boyer-Moore algorithm אלגוריתם בויאר־מור **447–452**

on a CREW PRAM CREW PRAM על (בעיה) **453**

by finite automata באמצעות אוטומטים סופיים **434–440**

with gap characters עם תווי "חלון" (תרגיל) **429**, (תרגיל) **440**

Knuth-Morris-Pratt algorithm אלגוריתם קנות'־מוריס־פראט **440–446**

naive algorithm האלגוריתם הנאיבי **428–430**

Rabin-Karp algorithm אלגוריתם רבין־קארפ **430–434**

string-matching automaton אוטומט התאמת־מחרוזות **435–439**, (תרגילים) **440**

strongly connected components, decomposition into רכיבים קשירים היטב, פירוק ל־ **100–105**

subgraph-isomorphisms, problem בעיית האיזומורפיות של תת־גרפים (תרגיל) **518**

subgroup תת־חבורה **394–395**

subset-sum problem בעיית סכום תת־קבוצה

approximation algorithm for אלגוריתם קירוב עבור **533–537**

NP-completeness of שלמות ב־NP של **510–512**

with unary target עם מטרה בייצוג אונרי (תרגיל) **518**

suffix (>) סיפא **427**

suffix function פונקציית הסיפא **436**

suffix-function inequality אי־שוויון ערכי פונקציית הסיפא **437**

suffix-function recursion lemma למת הרקורסיה של פונקציית הסיפא **438**

sum סכום

Cartesian קרטזי (תרגיל) **363**

flow זרימת (תרגיל) **184**

of matrices מטריצות **314**

of polynomials פולינומים **356**

sum bit סיבית הסכום **250**

summary of path labels סיכום של תוויות מסלולים **171**

summary operator (⊕) אופרטור סיכום **170**

summation lemma למת הסכום **365**

superpolynomial time זמן על־פולינומיאלי **480**

supersink בור־על **183**

supersource מקור־על **183**

sweeping סריקה **459–464**, (בעיה) **478**

sweep line קו סריקה **459**

sweep-line status מצב קו סריקה **461**

symmetric function פונקציה סימטרית (בעיה) **272**

symmetric matrix מטריצה סימטרית **314**, (תרגילים) **319**

symmetric positive-definite matrix מטריצה סימטרית חיובית־לחלוטין **346–348**

symmetry breaking שבירת סימטריה **301–307**

system of difference constraints מערכת אילוצי הפרשים **142–147**

system of linear equations מערכת משוואות לינאריות **330–342**, **353**

T

tally circuit מעגל מנייה (תרגיל) **258**

Tarjan's off-line least-common-ancestors algorithm

האלגוריתם של Tarjan לבעיה הלא־מקוונת של האבות הקדמונים המשותפים העמוקים ביותר (בעיה) **74**

tautology טאוטולוגיה (תרגיל) **491**, (תרגיל) **505**

text in string matching טקסט בהתאמת מחרוזות **426**

3-CNF **502**

3-CNF-SAT **502**

3-CNF satisfiability ספיקות נוסחאות 3-CNF **502–505**

3-COLOR (בעיה) **519**

3-conjunctive normal form צורה תלת־קוניונקטיבית נורמלית **502**

tick of a clock פעימת שעון **266**

timestamp חותמת זמן **91**, (תרגילים) **96–97**

Toeplitz matrix מטריצת טפליץ (בעיה) **375**

topological sort מיון טופולוגי **97–99**

in computing single-source shortest paths on a dag בחישוב מסלולים קצרים ביותר ממקור יחיד בגמ"ל **139**

tour מסלול

Euler אוילר **283**

of a graph של גרף **517**

tractable ניתן לפתרון סביר **480**

transition function פונקציית מעברים **435**, **439**, (תרגיל) **446**

transitive closure סגור טרנזיטיבי **163–164**, (תרגילים) **330**

and boolean matrix multiplication וכפל מטריצות בוליאניות (תרגיל) **330**, (תרגיל) **346**

of dynamic graphs של גרפים דינמיים (בעיה) **176**

transpose מוחלף

conjugate צמוד (תרגיל) **346**

of a directed graph של גרף מכוון (תרגיל) **82**

of a matrix של מטריצה **81**, **312**

transposing a raster image חילוף תמונת סריקה (בעיה) **309**

transposition network רשת חילוף (בעיה) **241**

traveling-salesman problem בעיית הסוכן הנוסע

approximation algorithm for אלגוריתם קירוב עבור **525–529**

bottleneck צוואר הבקבוק (תרגיל) **529**
NP-completeness of NP־שלמות ב־ **517-518**
with triangle inequality עם אי־שוויון המשולש **525-527**
without triangle inequality בלי אי־שוויון המשולש **528**
tree עץ
binomial בינומי **22-24, 41**
breadth-first רוחב **83, 89**
depth-first עומק **91**
diameter of קוטר של (תרגיל) **90**
dynamic דינמי **4**
finding the root of, on a CREW PRAM מציאת השורש של, על CREW PRAM **291**
full walk of סריקה מלאה של **527**
minimum spanning פורש מינימלי **108-119**
optimal vertex cover of כיסוי אופטימלי על־ידי קדקודים של (תרגיל) **525**
parse פיסוק **503**
pointer מצביעים (בעיה) **308**
shortest-paths מסלולים קצרים ביותר **123, 128-130**
spanning פורש **108**
splay ספליי **4**
star כוכב (בעיה) **308**
2-3 2-3 **20**
2-3-4 2-3-4 **8**
tree edge קשת עץ **89, 91, 95**
trial division ניסיון חילוק **412**
triangle משולש
in an undirected graph בגרף בלתי־מכוון (תרגיל) **330**
triangle inequality אי־שוויון המשולש **525**
triangular matrix מטריצה משולשית **313,** (תרגיל) **318**
tridiagonal linear system מערכת לינארית תלת־אלכסונית (בעיה) **353**
tridiagonal matrix מטריצה תלת־אלכסונית **313**
trimming of a list גיזום רשימה **534**
trivial divisor מחלק טריוויאלי **380**
truth assignment השמת ערכי אמת **494, 500**
truth table טבלת אמת **245**
TSP **517**
2-CNF-SAT (תרגיל) **505**
2-CNF satisfiability 2-CNF ספיקות נוסחאות (תרגיל) **505**
two-pass method שיטת שני המעברים **64**

2-3-4 heap 2-3-4 ערימת (בעיה) **36**
2-3-4 tree 2-3-4 עץ **8**
joining איחוד (בעיה) **19**
and red black trees ועצים אדומים־שחורים (תרגיל) **10**
splitting פיצול (בעיה) **19**
2-3 tree 2-3 עץ **20**

U
unary אונרי **483**
undetermined system of linear equations מערכת משוואות לינאריות תת־מוגדרת **331**
undirected graph גרף בלתי מכוון
articulation point of נקודת הפרדה של (בעיה) **105**
biconnected component of רכיב דו־קשיר של (בעיה) **105**
bridge of גשר של (בעיה) **105**
clique in קליקה בתוך **506**
coloring of צביעה של **302,** (בעיה) **519**
computing a minimum spanning tree in חישוב עץ פורש מינימלי עבור **108-119**
converting to, from a multigraph המרה ל־, ממולטי־גרף (תרגיל) **83**
d-regular *d*־רגולרי (תרגיל) **200**
grid שריג (בעיה) **218**
independent set of קבוצה בלתי תלויה של **302,** (בעיה) **519**
matching of זיווג של **197**
vertex cover of כיסוי על־ידי קדקודים של **508, 523**
union איחוד
of languages של שפות **484**
disjoint-set-forest implementation of מימוש של UNION על־ידי יער קבוצות זרות **63**
linked-list implementation of מימוש של UNION על־ידי רשימות מקושרות **59-61,** (תרגיל) **61**
union by rank איחוד על פי דרגה **62**
unique factorization of integers פירוק יחיד לגורמים של מספרים שלמים **384**
uniting איחוד
of binomial heaps של ערימות בינומיות **27-32**
of Fibonacci heaps של ערימות פיבונאצ'י **43**
of heaps של ערימות **21**
of 2-3-4 heaps 2-3-4 של ערימות (בעיה) **36-37**
unit lower-triangular matrix מטריצת יחידה משולשית תחתונה **314**
unit upper-triangular matrix מטריצת יחידה משולשית עליונה **313**
unit vector וקטור יחידה **312**
unmatched vertex קדקוד בלתי־מזווג **197**

unordered binomial tree עץ בינומי לא־סדור **41**
upper-triangular matrix מטריצה משולשית עליונה **313**

V

valid shift היסט תקף **426**
value of a flow ערך של זרימה **179–180**
Vandermonde matrix מטריצת ואנדרמונדה (תרגיל) **319**
van Emde Boas data structure van Emde Boas מבנה נתונים **4**
vector וקטור **312, 315–317**
  convolution of קונבולוציה של **358**
  cross product of מכפלה וקטורית של **455**
  in the plane במישור **455**
verification אימות **487–491**
vertex קדקוד
  articulation point נקודת הפרדה (בעיה) **105**
  intermediate ביניים **160**
vertex cover כיסוי על־ידי קדקודים **508, 523**
  for a tree עבור עץ (תרגיל) **525**
vertex-cover problem בעיית הכיסוי על־ידי קדקודים
  approximation algorithm for אלגוריתם קירוב עבור **523–525**
  and clique problem ובעיית הקליקה (תרגיל) **525**
  NP-completeness of שלמות ב־NP של **508–509**

W

Wallace-tree multiplier כופל עץ־וואלאס **262–264**
weight משקל
  of an edge של קשת **81**
  mean ממוצע (בעיה) **149**
  minimax מינימקסימלי (בעיה) **177**
  of a path של מסלול **120**
weighted bipartite matching זיווג בגרף דו־צדדי משוקלל **55**
weighted set-covering problem הגרסה המשוקללת של כיסוי קבוצה (בעיה) **538**
weighted-union heuristic היוריסטיקת האיחוד המשוקלל **60**
weight function for a graph פונקציית משקל עבור גרף **81**
white-path theorem משפט המסלול הלבן **94**
white vertex קדקוד לבן **84, 91**
widget רכיב בנייה **512**
wire תיל **228, 246**

witness to the compositeness of a number עד לפריקותו של מספר **414**
work עבודה **280,** (תרגיל) **296**
work-efficient PRAM algorithm אלגוריתם PRAM יעיל בעבודה **280**
  matrix multiplication on a CRCW PRAM כפל מטריצות על CRCW PRAM (תרגיל) **296**
  matrix-vector multiplication on an EREW PRAM כפל מטריצה בווקטור על EREW PRAM (תרגיל) **296**
  parallel prefix computation חישוב רישות מקבילי **296–301**
work-efficient simulation הדמיה יעילה בעבודה **294–295**

X

XOR (exclusive-or) function (⊕) פונקציית XOR ("או־מוציא") **245**
XOR gate שער XOR **245**

Y

Yen's improvement to the Bellman-Ford algorithm השיפור של ין לאלגוריתם בלמן־פורד (בעיה) **147**

Z

$\mathbf{Z}_n$ (equivalence classes modulo n) ($\mathbf{Z}_n$ מחלקות שקילות מודולו n) **381**
$\mathbf{Z}_n^*$ (elements of multiplicative group modulo n) ($\mathbf{Z}_n^*$ איברי החבורה הכפלית מודולו n) **392**
$\mathbf{Z}_n^+$ (nonzero elements of $\mathbf{Z}_n^*$) ($\mathbf{Z}_n^+$ איברים שונים מאפס של $\mathbf{Z}_n^*$) **413**
$\bar{0}$ **170, 326**
zero-knowledge encryption הצפנת אפס־ידע **425**
zero matrix מטריצת אפס **312**
zero of a polynomial modulo a prime אפס של פולינום מודולו מספר ראשוני (תרגיל) **399**
0-1 integer-programming problem גרסת 0-1 של בעיית התכנון בשלמים (תרגיל) **518**
0-1 knapsack problem גרסת 0-1 של בעיית התרמיל (תרגיל) **487**
zero-one principle עקרון ה־0-1 **231–233,** (תרגיל) **237,** (תרגיל) **238**