Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Computational aspects of Generating Catalan numbers

Bachelor Thesis

2020
Nikola Horníková

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Computational aspects of Generating Catalan numbers

Bachelor Thesis

Study Programme:	Applied informatics
Field of Study:	Applied informatics
Department:	Department of applied informatics
Supervisor:	doc. RNDr. Tatiana Jajcayová, PhD.

Bratislava, 2020
Nikola Horníková

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Nikola Horníková
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Computational Aspects of Generating Catalan Numbers
Výpočtové aspekty generovania Catalánových čísel

Anotácia:

Cieľ:

Vedúci: doc. RNDr. Tatiana Jajcayová, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 07.10.2019

Dátum schválenia: 08.10.2019 doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname: Nikola Horníková
Study programme: Applied Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study: Computer Science
Type of Thesis: Bachelor´s thesis
Language of Thesis: English
Secondary language: Slovak

Title: Computational Aspects of Generating Catalan Numbers

Annotation: Catalan Numbers appear in various contexts in Computer science. The nth Catalan number is given directly in terms of binomial coefficients, and the recursive definition given by a non-linear recurrence relation is also known. One of the mathematical objects to which the Catalan numbers are related are so called Hankel matrices, and those, in turn, are used with Hidden Markov models.

Aim:

The goal of the theses is to study this connection between the Catalan numbers and Hidden Markov models. Possible means to achieve this is to connect various different combinatorial problems in Informatics in which Catalan Numbers appear, employ some knowledge of statistics and probability, understand Hidden Markov models and design good tests and algorithms. This requires to use “intelligent" approach to programming as the sequence of Catalan Numbers grows very fast.

Supervisor: doc. RNDr. Tatiana Jajcayová, PhD.
Department: FMFI.KAI - Department of Applied Informatics
Head of department: prof. Ing. Igor Farkaš, Dr.

Assigned: 07.10.2019

Approved: 08.10.2019 doc. RNDr. Damas Gruska, PhD.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Acknowledgments: I would like to thank my supervisor doc. RNDr. Tatiana Jajcayová, PhD for her time, provided resources, consultations and valuable advice.

Abstrakt

Cieľom tejto práce je pokúsiť sa nájsť prepojenie medzi Katalánskymi číslami a Skrytými Markovovými modelmi. Tento cieľ sme dosiahli skúmaním a študovaním Katalánskych čísel, Hankelových matíc, Markovových reťazcov, Markovových modelov, až nakoniec Skrytých Markovových modelov. Rozhodli sme sa pre skúmanie tohto spojenia pomocou modelovania Markovových modelov použitím nami naprogramovaných algoritmov. Pomocou nich sme uskutočnili viacero experimentov a dostali sme rôzne výsledky. Z výsledkov sme odvodili a opísali spojenie, ktoré sme našli. Naša práca a jej výsledky sa dajú použiť ako vhodný odrazový mostík k ďalšiemu výskumu.

Kľúčové slová: Katalánske čísla, Hankelove matice, Skryté Markovove modely, programovací jazyk Rust

Abstract

The goal of the thesis was to study this connection between the Catalan number sand Hidden Markov models. We accomplished that by studying the Catalan numbers, Hankel matrices, Markov chains, Markov models and finally Hidden Markov models. To analyze the fields and explore the possible connection, we implemented algorithms related to Hidden Markov models. We executed multiple experiments in order to find the connection between the Catalan numbers and Hidden Markov models. The experiments resulted in both successful and unsuccessful conclusions. We were able to find and describe the found connection. We also provide a good starting point for further research of the connection.

Keywords: Catalan numbers, Hankel matrices, Hidden Markov models, Rust programming language

Contents

Introduction

Catalan numbers are a ubiquitous number sequence that arises in numerous surprising situation. Fascinating, yet not as popular as Fibonacci numbers, they have been studied for a long time and have appeared in various situations, related to various fields. We decided to study those numbers, pay attention to the similarities of their occurrences, and possibly find another one. One of the mathematical objects that the Catalan numbers appear in are matrices, specifically Hankel matrices. We studied the behaviour of Hankel matrices and its connection to the Catalan numbers.

During this study, we found out that the Hankel matrices are used with Hidden Markov models due to its rank property. We found this statistical model to be different from structures in examples of Catalan numbers we already crossed. Thus we decided to look for a connection between the Catalan numbers and Hidden Markov models. Hidden Markov models required another look into a distinct field, statistics, and probability. From Markov chains, through Markov models, we finally learnt about Hidden Markov models.

We implemented algorithms to build those models, in order to examine its capability to learn from data sets related to Catalan numbers. We chose Rust programming language as the implementation language for the algorithms because of its multiple benefits such as safety and speed. We implemented four algorithms related to Hidden Markov models and encountered some deficiencies in the used libraries. To counter them, we had to implement multiple functions mostly for matrix operations to be used in the algorithms. There were many possibilities to choose the data set from, so we decided to choose three examples of occurrences of the Catalan numbers to experiment with. We ended up with various results, both successful and failed. In one or the other, we studied the reasons behind them and provided an explanation.

Chapter 1

The origin of Catalan numbers

In this chapter, we will discuss Catalan numbers' history [11] [23], when and where they were noticed first and the different approaches of different people who led to their discovery. We will examine the increase of popularity in Catalan numbers throughout the past 200 years.

We will also talk about the process of how the Catalan numbers got their name and will point out various people of high importance in mathematics that contributed to either of the aforementioned events.

1.1 History

The Catalan numbers have appeared in many disguises since the 18th century. By being camouflaged, yet commonly present around, they arise in many different departments of mathematics and other fields. As a result, the Catalan numbers been rediscovered multiple times in various parts of the world.

1.1.1 China

Even though they were not known by the name we know them today, the first mention of the Catalan numbers goes back to the 18th century to China. It came to light that they were first used by a Mongolian mathematician Ming Antu, born in 1692. He published a book in 1730 called *Quick Methods for Accurate Values of Circle Segments*, in which he describes multiple trigonometric identities and power series, some of which are associated with the Catalan numbers. The book was examined by Luo Jianjin in 1988. Luo Jianjin noticed the occurrence of Catalan numbers in trigonometric expansions discussed in Ming Antu's book and pointed it out.

1.1.2 Europe

In the second half of the 18th century, around the year 1750, Euler was spending time looking for a number of ways to triangulate a polygon. He expressed his thoughts in a letter to Christian Goldbach in 1751. Goldbach, inspired by the idea, replied to Euler with a quadratic formula which according to him could be used to calculate the wanted numbers. Thanks to that, Euler derived the wanted formula from a binomial formula which he recalled from his earlier works. Euler thus composed a formula for calculating the number of ways to triangulate a polygon of n sides, which is one of the examples, where Catalan numbers arise.

Euler mentioned this problem to Johann Andreas von Segner too. Despite their rivalry, their cooperation resulted in Segner's work in which he derives and proves the recurrence relation. However, there was a mistake in this formula. Before publishing the work, Euler corrected it and published it as his own work.

1.1.3 The rest of the world

During the following 150 years, the Catalan numbers were discussed in various parts of the world. For example, Russians found ways to prove the Euler's formula differently. English provided other examples where Catalan number arise and studied the Euler's work.

1.2 Modern times

It had taken 80 years to prove the Euler formula and many more to find out and connect different examples to Catalan numbers. They are not so popular as some other number sequences such as the Fibonacci or Bernoulli ones, however, the interest in them is still growing. There are already numerous examples of them, which we will discuss in the following chapter, and they have still been appearing in more and more situations.

1.3 The name

1.3.1 Eugene Charles Catalan

Eugene Charles Catalan was born in 1814 in Belgium, however, he studied in Paris and contributed to the studies of the Catalan numbers in France. Catalan was interested in the work of his teachers who studied the number of triangulations of a polygon. Based on that, he obtained the standard formula for the nth Catalan number. He discovered many examples of where the Catalan numbers arise and found a one-to-one correspondence between them, for example, the number of different products of

n variables and the number of bracket sequences, discussed in chapter 2. Catalan also studied and published a work on divisibility of the Catalan numbers. He was also the first one to define the ballot numbers, a number of ways in which the votes could be counted so that the winner was always ahead in an election held between two candidates.

1.3.2 The first mention

As a result of their chaotic history, there are many theories of how the Catalan numbers got their name. They did not have a specific name for a long time and were addressed as *Euler numbers*, *Segner numbers*, and *Euler–Segner numbers*. However, these names did not catch on. In 1968 an American combinatorialist John Riordan called these numbers the *Catalan numbers* and with his monograph, the name for the numbers became popular [27].

1.4 The formulas

There are many ways to calculate the Catalan numbers; recursively, directly, from the Pascal triangle [17], some other numbers sequences, and so on [11]. We implemented some of the approaches of how the Catalan numbers can be calculated in chapter 5 and described their complexity. In this section, we briefly provide the direct formula and the recursive formula, just to give the reader the idea of how the Catalan numbers look like. The first few terms of the sequence are:

$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845...$

Wee see that the Catalan numbers are a sequence with exponential growth [12]. The only prime Catalan numbers are $C_n = 2$ and $C_3 = 5$ [11]. The number C_n is proved [11] to be odd if and only if n is a Mersenne number. Mersenne numbers are a sequence defined as $M_n = 2^n - 1$ [21].

The recursive definition

$$C_n = C_0C_{n-1} + C_1C_{n-2} + ... + C_{n-2}C_1 + C_{n+1}C_0$$

which is equal to

$$\sum_{k=1}^{n} C_{k-1}C_{n-k}$$

where

$$C_0 = 1$$

.

The direct formula

The Catalan numbers can be computed directly as follows:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n!)}{(n+1)!n!} = \binom{2n}{n} - \binom{2n}{n+1}$$

The origin of this formula is derived in the first example provided in the following chapter 2.

Chapter 2

Various encounters with Catalan numbers

We collected some examples [11] [10] [5] of various situations where the Catalan numbers arise and we will present them to show the reader the diversity of unexpected occurrences of these numbers. We will not provide the proof for the given examples, those can be found in the literature listed in the bibliography. We will focus on emphasizing the presence of the Catalan numbers in different fields of mathematics and computer science and showing how they all connect.

Even though the Catalan numbers appear in many combinatorial problems, they also arise in many other surprising situations. Despite the variety of examples, they all end up with the same sequence of Catalan numbers. Therefore, there can be created a one-to-one correspondence between the enumerations considered in examples where the Catalan numbers arise, which might or might not be so clear at first sight. To help the reader see the connection between the unexpected occurrences, we order the examples among which a bijection is more transparent.

Lattice paths of n R's and n U's

Consider an $n \times n$ grid, on which you want to count the number of paths that start at $(0, 0)$ and end at (n, n) and never rise above the line $y = x$ (figure 2.1).

You are only allowed to go up, so to make an U move, or to go right, so to make an R move. One of the valid paths is for example the path shown in figure 2.2.

The RU paths are sequences of R's and U's. The sequence of the path in the figure 2.2 would be $RURURURURURU$. Such paths follow a rule that the number of U's never exceeds the number of R's (we never go more up than we go to the right, otherwise we would get above the line $y = x$). To calculate the number of these paths, we calculate the number of all paths and subtract the number of wrong ones, the paths that rise above the line $y = x$.

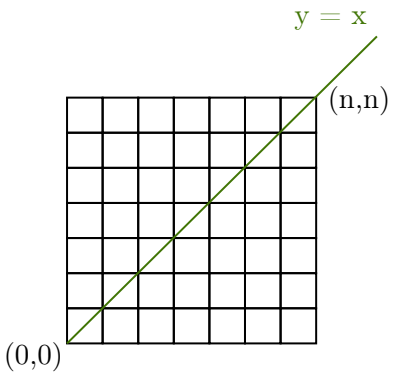

Figure 2.1: Line $y = x$ on $n \times n$ grid

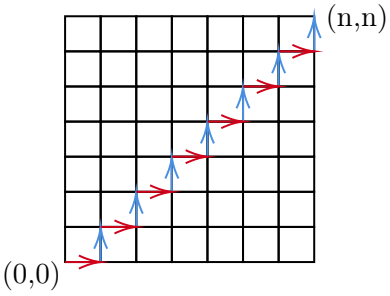

Figure 2.2: An example of a valid RU path

For an $n \times n$ grid, the number of ways to get from $(0, 0)$ to (n, n) is the number of arrangements of n R's and n U's, which is $\binom{2n}{n}$.

Now let us calculate the "wrong paths". To do so, take any path of n R's and n U's that goes above the line $y = x$, starts at $(0, 0)$ and ends at (n, n). Take a sequence of n R's and n U's such that the number of U's exceeds the number of R's at some point. Now, starting from the point where the path crossed the line $y = x$ for the first time, swap R's and U's. Doing so, you will always end up in the point $(n - 1, n + 1)$. By calculating the number of paths of $n-1$ R's and $n+1$ U's starting at $(0, 0)$ and ending at $(n - 1, n + 1)$, we get the number of paths consisting of n R's and n U's starting at $(0, 0)$ and ending at (n, n) that go above the line $y = x$, which is $\binom{2n}{n-1}$.

Therefore, the number of paths that start at $(0, 0)$ and ends at (n, n) and never rise above the line $y = x$ is $\binom{2n}{n} - \binom{2n}{n-1}$, which is one of the formulas to calculate the nth Catalan number already mentioned in chapter 1.

Strings of n 0's and n 1's

Let us look at words over a binary alphabet $\{0, 1\}$. We want to count the number of sequences, such that the number of 0's never exceeds the number of 1's. There is a clear bijection to the previous example by replacing the R's with 1's and U's with 0's.

Arrangement of n 1's and n -1's

In this example, we want to count the number of arrangements of 1's and -1's so that all partial sums are non-negative. Therefore we are counting the sequences of 1's and -1's in which the number of -1's never exceeds the number of 1's. Otherwise, some partial sum would be negative.

Ballot votes

As mentioned in the first chapter, the ballot numbers are numbers that describe how votes could be counted so that the winner is always ahead in an election held between two candidates. This idea can be used to enumerate many other examples of a situation of competition of two figures, such that one is always in the lead.

Balanced strings of n left and n right parentheses

This problem is once again related to creating words of 2 symbols, so we are starting to see a pattern here. We have the same number of left and right parentheses, so we know that each opening parentheses will have a closing one if it comes after it. Therefore, if a string has a balanced number of parentheses, the number of left parentheses never exceeds the number of right ones (because the closing parenthesis would not match any opening one).

The arrangement of integers

Consider an arrangement of integers such as $1, 2, 3, ...2n-1, 2n$ so that:

- the odd integers occur in increasing order,
- and the even integers occur in increasing order,
- and $2k-1$ appears before $2k$ for all $1 \leq k \leq n$

By replacing the odd numbers with 1's and the even numbers with -1's, we create a bijection to one of the previous problems.

Dyck paths

Dyck paths are paths on an $n \times n$ grid such that we can only move on diagonals, either from bottom left to the upper right or from the upper left to the bottom right, the path has a length of $2 \times n$ and it never drops below the x-axis. Dyck paths are the simplest occurrence of the Catalan numbers in examples with paths. There are many types of paths on a grid that under some conditions result in the Catalan numbers. For example *Motzkin paths* that never drop below the x-axis, Dyck or Motzkin paths with *peaks* or *valleys*. They, however, discover many more occurrences of the Catalan numbers, such as examples with *Pairwise disjoint collections* which are worth mentioning.

Young Tableaux

Young tableau is an object used to represent algebraic structures in a linear way. We will focus on *Young diagram*, which is $2 \times n$ cells divided into n columns and 2 rows.

We will be placing integers from 1 to n so that the entries in each row are in ascending order and for each of the n columns the entry in the first row is smaller than the entry in the second row. The number of ways to do so are the Catalan numbers, even though this example might seem distinct from the previous one. However, there is a simple one-to-one correspondence to the Dyck paths.

Compositions of an integer i

Consider having a pair of two compositions of the integer i: $a_1 + a_2 + ... + a_n$ and $b_1 + b_2 + ... + b_n$ so that each partial consequent sum of m a's is greater or equal to a partial consequent sum of m b's, where $0 < m \leq n$. To create a bijection to Dyck paths, we draw a_1 D steps, b_1 D* steps ... a_n D steps, b_n D* steps.

Arcs

Let us have n vertices lying on a horizontal line and being connected by arcs so that:

- each arc connect two vertices above the horizontal line,
- and each vertex is part of just one arc,
- and no two of the arcs intersect.

To create a bijection, each time we encounter the start of an arc we write down 1 and when we encounter it again we write -1. This way we end up with a sequences of 1's and -1's, in which the number of -1's never exceeds the number of 1's (because the starting point of an arc must come before its' ending point). This example was already seen.

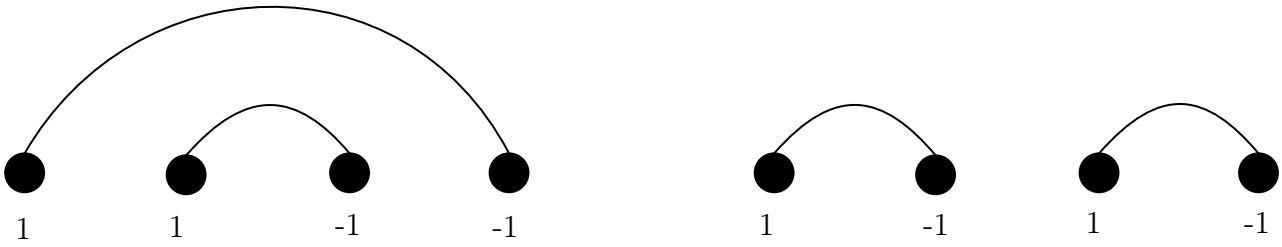

Figure 2.3: Two ways to connect 4 vertices with 2 arcs

We listed the 2 ways (C_2) to connect 4 vertices with 2 arcs respecting the aforementioned conditions in the figure 2.3. The resulting sequences would be the labels of the vertices from left to right. These are the sequences of 1's and -1's of length 4, such

that the number of -1's never exceeds the number of 1's.

There are many modifications of this arc example for instance:

- When we have $n-2$ vertices some of which may be isolated but no isolated vertex may lie under an arc.
- When we have $n-2$ vertices some of which may be isolated and they may lie under an arc.
- When we have $2n-2$ vertices on a horizontal line and we label them consecutively. Then, for each arc that appears, the left endpoint is at an odd-numbered vertex while the right one is at an even-numbered vertex.

Diagonalization of the polygon

This problem is closely related to triangulation of a polygon which is often associated with computer graphics. Now we will talk about the number of ways to draw $n-1$ diagonal in a polygon P of $n+2$ sides within its interior so that:

- no two diagonals intersect within the interior of P,
- and the diagonal partition of the interior of P into n triangles.

To create a bijection, we label the diagonals with the names of the sides of the polygon and parenthesize them if necessary.

Non-decreasing sequences

Consider a non-decreasing sequence $a_1, a_2, ..., a_n$ of positive integers where $a_i \leq i$ for all $1 \leq i \leq n$. To create a bijection between these sequences and the aforementioned arrangements of 1's and -1's, we create a sequence of n 1's and n -1's and for each 1, we write down the number of -1's on the left. We end up with a new sequence and by adding 1 to each element of the sequence, we create all wanted non-decreasing sequences.

Complete rooted binary trees on $2n+1$ vertices

Trees are frequently used data structures in computer science. A rooted tree is a tree in which a node is singled out, called the root of the tree. A binary tree is a tree in which each node has at most two children. There are many different kinds of trees and their traversals. In this example, we have a complete binary tree and we traverse it by going left (L) and then right (R) if the current node has children. The tree is complete, so the number of L's and R's must be equal. We start with (L) and each node that

has a left child must have a right child because the given tree is complete. Therefore the number of R's never exceeds the number of L's.

Rooted ordered trees on $n+1$ vertices

A rooted ordered tree is a tree in which the nodes are somehow ordered. By a postorder traversal of the tree, we write down 1 when we encounter a vertex for the first time and we write -1 when we encounter it on the way back.

For instance, the tree in figure 2.4 would be traversed in postorder as follows: $0, 1, 2, 3, 4, 5, 6, 7, 8$. The constructed sequence of 1's and -1 would be $1(8), 1(5), 1(2), 1(1), 1(0), -1(0), -1(1), -1(2), 1(4), 1(3), -1(3), -1(4), -1(5), 1(7), 1(6), -1(6), -1(7), -1(8)$ (the numbers in brackets corresponds to the labels of the vertices).

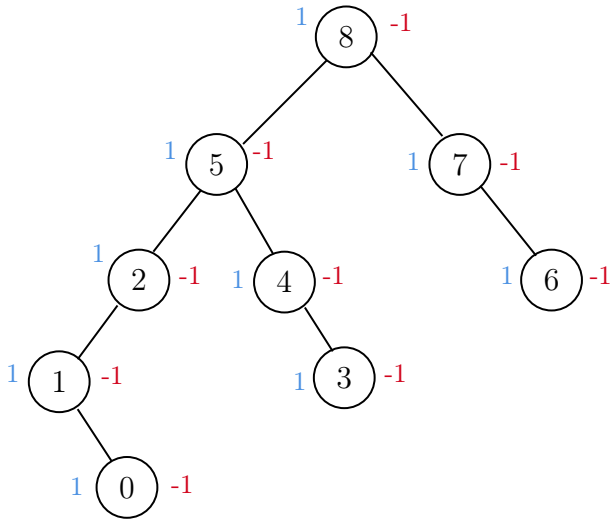

Figure 2.4: An example of a postorder traversal of a binary tree

Rooted ordered binary trees on n vertices

We want to calculate the number of different binary trees with n vertices. We select one of the n vertices as the root. The substructures on left and right are then rooted ordered binary trees on $n-1$ vertices. We consider the division of the $n-1$ vertices between the two subtrees:

- 0 vertices on left, $n-1$ vertices on right, so $t_0 \times t_{n-1}$ possibilities
- 1 vertex on left, $n-2$ vertices on right, so $t_1 \times t_{n-2}$ possibilities
- i vertices on left, $n-1-i$ vertices on right, so $t_i \times t_{n-1-i}$ possibilities
- $n-1$ vertices on left, 0 vertices on right, so $t_{n-1} \times t_0$ possibilities

There is a simple bijection to the *Rooted ordered trees* mentioned before by removing the leaves from the trees from the former one.

Special rooted trees

Now we will look into the rooted trees in which every node to the right has a greater number of children than his sibling on the left. We want to count the number of vertices at level $n-1$ in such tree. This occurrence of Catalan numbers and it's bijection to another example might appear a little bit forced, however, it is crucial as it deduces other examples in which the Catalan numbers arise.

The bijection is done by creating a sequence from root to a leaf from the number of children the node has. The sequences are the aforementioned non-decreasing sequences.

If we look into the paths from the root to leaves, we get sequences of numbers which again occurs in results of other problems which therefore also results in the Catalan numbers, for example, *The tennis ball problem*, or when we examine n nested for loops.

Cliques

Another popular data structure also used in computer science is a graph. A clique is induced by a subset W of set of vertices V, where for all $(a, b) \in W$ the edge $(b, a) \in W$. A maximal clique is a clique induced by the longest of such subsets of vertices. Now we want to calculate the number of graphs with vertex set $V = \{1, 2, ..., n\}$ where the vertex set for each maximal clique is made up of a set of consecutive integers. We create recursively all graphs as sequences so that with $n = 1$ we have only one option (1), a graph of one vertex. Then, for $n = 2$, we either do only one step, add a vertex (1,1), or we do two steps, add a vertex, and connect them (1,2). We continue and create all trees recursively. The trees created are the same as the trees in the previous example.

Stack

This data type serves as a collection of elements which follows a particular order in which the operations *push* and *pop* are performed. The ways of reordering a sequence with stack are also Catalan numbers. Each push operation is 1, each pop operation is 0. The number of 0's never exceeds the number of 1's, because we cannot remove elements from an empty stack.

Chapter 3

Hankel matrices and Catalan numbers

As seen in the previous chapter, the Catalan numbers arise in many different situations. Another fascinating aspect of the Catalan numbers is their interesting behaviour in mathematical structures and objects. In this chapter, we provide a brief introduction to Hankel matrices and how they relate to Catalan numbers.

3.1 Hankel matrices

Hankel matrix, also known as persymmetric matrix or catalecticant matrix, is a square matrix in which each ascending skew-diagonal from left to right is constant. Alternatively, a Hankel matrix

$$A = \begin{pmatrix} c_0 & c_1 & c_2 & ... & c_{n-1} \\ c_1 & c_2 & c_3 & ... & c_n \\ c_2 & c_3 & c_4 & ... & c_{n+1} \\ ... & ... & ... & ... & ... \\ c_{n-1} & c_n & c_{n+1} & ... & c_{2n-2} \end{pmatrix} \tag{3.1}$$

is a matrix $(a_{i,j})$ in which for every r the entries on the diagonal $i + j = r$ are the same, for example $a_{i,r-i} = c \times r$ for some c [28] as seen in 3.3.

Given an infinite sequence $\{a_n\}_{n=0}^{\infty}$, the Hankel matrix of this sequence is the infinite matrix H, whose entry (i, j) is a_{i+j} [8]. Using zero-based numbering, the described matrix looks as follows:

$$H = \begin{pmatrix} a_0 & a_1 & a_2 & ... \\ a_1 & a_2 & a_3 & ... \\ a_2 & a_3 & a_4 & ... \\ ... & ... & ... & ... \end{pmatrix} \tag{3.2}$$

Then, generally, we will use the notation $H(n)$, where $n \geq 0$ is the upper-left $(n+1) \times (n+1)$ submatrix of H.

For example:

$$H(0) = \begin{pmatrix} a_0 \end{pmatrix} \qquad H(1) = \begin{pmatrix} a_0 & a_1 \\ a_1 & a_2 \end{pmatrix}$$

The aforementioned matrix A in equation 3.1 would therefore be a Hankel matrix $H(n-1)$ of size $n \times n$.

3.2 Hankel transform

We will denote h_n to be the the determinant of the matrix $H(n)$, so $h_n = det(H(n))$. The determinant of a matrix is a number that is defined only for square matrices, which the Hankel matrix is. Determinants are mathematical objects that are often used in the analysis of systems of linear equations.

The Hankel transform is a sequence of determinants of hankel matrices of a sequence. Formally, the sequence $\{h_n\}_{n=0}^{\infty}$ is a Hankel transform of a sequence $\{a_n\}_{n=0}^{\infty}$.

3.3 Catalan numbers step in

The Catalan numbers and their behaviour in Hankel matrices with applied Hankel transform is already a subject of many papers, the results of which we describe in this section. Not only were studied the Hankel transforms of the whole sequence, but also the Hankel transforms of the sequence of the Catalan numbers without the first n terms.

The Hankel matrix of the sequence of the Catalan numbers looks as follows:

$$A = \begin{pmatrix} 1 & 1 & 2 & 5 & 14 & ... \\ 1 & 2 & 5 & 14 & 42 & ... \\ 2 & 5 & 14 & 42 & 132 & ... \\ 5 & 14 & 42 & 132 & 429 & ... \\ 14 & 42 & 132 & 429 & 1430 & ... \\ ... & ... & ... & ... & ... & ... \end{pmatrix} \tag{3.3}$$

Desainte-Catherine and Viennot found the determinant of Hankel matrices of the Catalan numbers to be $\prod_{1 \leq i \leq j \leq k} \frac{i+j+2\times n}{i+j}$ [18].

As proved by Mays and Wojciechowski [20], the Hankel transform of the sequence of Catalan numbers obtained by removing the first element of the sequence, number 1, is a sequence of 1's. Moreover, the application of Hankel transform to the sequence with two terms emitted, 1 and 1, results in the sequence $\{n+2\}_{n=0}^{\infty}$. They also proved that by removing the first three elements, namely 1, 1 and 2, we get the sequence

$$\{\frac{(n+2)(n+3)(2\times n+5)}{6}\}_{n=0}^{\infty}.$$

Another amusing proof [4], written by Cvetković, Rajković, and Ivković, is that the Hankel transform of the sequence of the sum of two consecutive Catalan numbers, $\{c_n + c_{n+1}\}$ is a sequence of every other Fibonacci number.

3.3.1 Hankel transform of Catalan numbers in relation with other specific matrices

Consider a Hankel matrix of the matrix of the Catalan numbers $C\{k\} = \{c_{i+j+k}\}$, with the first k elements of the sequence emitted. Let us have LU decomposition of a matrix A (3.3).

The LU decomposition of a matrix A is the LUP decomposition of a non-singular square matrix, where $A = L\times U\times P$ (matrix product), in which the matrix L represents the *lower triangular matrix* of the matrix A, U represents the *upper triangular matrix* of the matrix A and P represents a special kind of *permutation matrix*, I_n, called *identity matrix*. Hence, P is absent and $A = L \times U$ [3].

Let us define the lower triangle of the matrix A as $L_{i,j} = (-1)^{i+j} \times \binom{i+j}{i-j}$. Then, because $U = L^T$, $U_{i,j} = (-1)^{i+j} \times \binom{i+j}{j-i}$. As proved by Dougherty, French, Saderholm, and Qian [8], the matrix product $LC\{0\}U$ is an identity matrix. Moreover, $LC\{k\}U_{i,j} = \binom{2k}{k+i-j}$, where $i + j \geq k \geq 0$.

Chapter 4

Markov models

Even though the Catalan numbers do not have a direct connection with Markov models, they both have a common mathematical structure, the Hankel matrix introduced in chapter 3. In order to understand the connection between them, we provide a brief introduction to Markov chains and more specifically the Hidden Markov models.

4.1 Markov chain

Markov chain is a *stochastic model* describing a sequence of states that evolves randomly in time and remembers its past state only by its most recent value [22]. Markov chain a discrete-time process and its continuous-time version is called *Markov process*. Markov process is a *stochastic process* that satisfies the *Markov property*. A stochastic process is said to hold Markov property when the past has no bearing on the future, so it is *memoryless*.

4.1.1 Regular Markov chain

A Markov chain is regular, if its transition matrix is *regular*, which means that some power of the matrix has only positive entries [31]. Let us describe the vocabulary used with Markov chains in this simple made-up example:

Imagine a coffee company called A, and all other coffee companies represented by A'. Now consider an advertising plan for company A, whose goal is to have as many customers as possible. Their advertising plan is supposed to be really successful and its predicted results are given by how the percentage of customers of either of the companies changes after a week. The probability that a customer who buys coffee from company A will continue purchasing it from this company is 0.8. Then, because the actions are mutually exclusive, the probability that a customer who buys coffee from company A will start purchasing coffee from company A' is 0.2. The probability that a

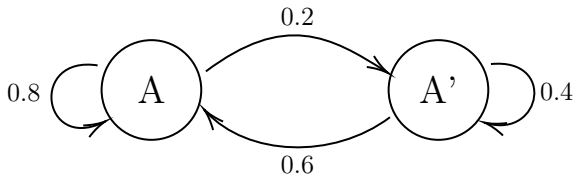

Figure 4.1: Transition diagram

customer continues buying coffee from company A' when he previously bought it from A' is 0.4. Then the probability that a customer buying coffee from company A' starts purchasing it from company A is 0.6. This situation is drawn in a *transition diagram* in the figure 4.1:

The same information could be shown by a *transition probability matrix* (4.1), which is a *row-stochastic* matrix with labels corresponding to the coffee companies A and A'. The labels at the top represent the next state, and the right labels represent the current state:

$$P = \begin{matrix} \\ A \\ A' \end{matrix} \begin{matrix} \begin{matrix} A & A' \end{matrix} \\ \begin{pmatrix} 0.8 & 0.2 \\ 0.6 & 0.4 \end{pmatrix} \end{matrix} \tag{4.1}$$

In order for the chain to begin, we need an *initial state distribution matrix* (4.2), that shows the probability distribution among states. In our example, it means how many people buy coffee from company A and A':

$$S_0 = \begin{matrix} \begin{matrix} A & A' \end{matrix} \\ \begin{pmatrix} 0.1 & 0.9 \end{pmatrix} \end{matrix} \tag{4.2}$$

To calculate how the advertising plan will affect the customers after one week, we multiply the initial state distribution matrix by the state probability matrix and we get the matrix S_1 (4.3), representing how many customers buys coffee from company A and A' after one week:

$$S_1 = \begin{matrix} \begin{matrix} A & A' \end{matrix} \\ \begin{pmatrix} 0.62 & 0.38 \end{pmatrix} \end{matrix} \tag{4.3}$$

We see that the advertisement plan really worked out for the company A. However, if we multiplied the matrix S_0 (4.2) by the probability transition matrix P (4.1) enough

times, we would eventually get a *stationary* matrix (4.4):

$$S_n = \begin{matrix} A & A' \\ \begin{pmatrix} 0.75 & 0.25 \end{pmatrix} \end{matrix} \tag{4.4}$$

The matrix S_n (4.4) is called a stationary, because it will stay the same if we multiply it by the probability transition matrix any number of times. In our example it means that after any number of weeks, the percentage of customers buying coffee from the company A will be 0.75 at most.

4.1.2 Absorbing Markov chain

A Markov chain is regular, if it has at least one *absorbing state* and it is possible to get from each non-absorbing state to at least one absorbing state in a finite number of steps [8]. A state is absorbing if once the state is entered, it is impossible to leave.

Let us introduce an exemplary situation where you are standing on a street S and deciding whether you go to a restaurant A or B. If you enter the chosen restaurant, you do not leave for the other one, neither you eat at both. This situation is displayed in a following transition diagram in the figure 4.2:

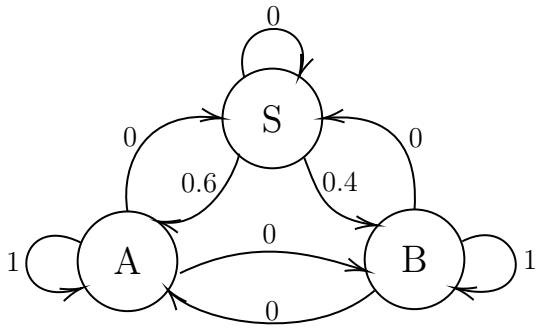

Figure 4.2: Transition diagram

The states A and B are the absorbing states. As you can see, there is a probability of 1 that you will stay in them and 0 that you will go to any other state.

4.2 Hidden Markov models

Unlike Markov Chain, a Hidden Markov Model (HMM) is a Markov model in which the state is not fully observable, rather it is only observed indirectly by noisy observations. It is modeled for a process that has the Markov property. If the current state depends

only on the previous state, we talk about *first order* HMM. Similarly, if the current state depends on n previous states, it is a HMM with order n [9].

Let us provide an example in which we will explain the terms used with Hidden Markov models:

Imagine yourself living on a continent A and having a friend on a different continent B. Suppose there is no internet or news, so you cannot obtain information about weather on the continent B directly. However, you and your friend are really close and you call each other every day and talk about what you did during that day. You know that all possible weather states that can be at the continent B are "sunny" (S) and "rainy" (R). The transition from one state to another is a first order Markov process. You also know the probabilities of transitions between the weather states, which can be represented by a transition probability matrix discussed in previous sections of this chapter.

$$A = \begin{matrix} \\ S \\ R \end{matrix} \begin{matrix} \begin{matrix} S & R \end{matrix} \\ \begin{pmatrix} 0.6 & 0.4 \\ 0.45 & 0.55 \end{pmatrix} \end{matrix} \tag{4.5}$$

Your current observation of your friend behaviour indicates a correlation between the weather and his activities. You know that when it is "rainy", he usually stays at "home" (H) and when it is "sunny", he usually goes "out" (O). You know these activities happen with certain probabilities. These observations can be put in an *observation matrix*, sometimes called *emission probability matrix* (4.6).

$$B = \begin{matrix} \\ S \\ R \end{matrix} \begin{matrix} \begin{matrix} H & O \end{matrix} \\ \begin{pmatrix} 0.3 & 0.7 \\ 0.7 & 0.3 \end{pmatrix} \end{matrix} \tag{4.6}$$

The *hidden states* in this model are the states of the weather: "sunny" and "rainy". You cannot observe them directly, but you monitor them obliquely.

You have monitored your friend for 4 days and you observed his behaviour as followed: On the first two days, he stayed at home, then he went out and on the last day, he stayed at home. To start with the computations, we need to know, in which state we begin. You know that the continent has some probabilities of having *sunny* or *rainy* weather, let us say there is a 50% probability of *sunny* and 50% probability

of *rainy*. This information is shown in the initial state distribution matrix (4.7):

$$\pi = \begin{matrix} S & R \end{matrix} \\ \begin{pmatrix} 0.5 & 0.5 \end{pmatrix} \tag{4.7}$$

In addition to the introduced matrices A(4.5), B(4.6) and π(4.7), the following notation is largely used for terms associated with Hidden Markov models [26]:
T is the length of the observation sequence
N is the number of states in the model
M is the number of observation symbols
$Q = \{q_0, q_1, ..., q_{N-1}\}$ is a distinct states of the Markov process
$V = \{0, 1, ..., M-1\}$ is a set of possible observations
$O = (O_0, O_1, ..., O_{T-1})$ is an observation sequence
$A = \{a_{ij}\}$ is a square matrix: $a_{ij} = P(\text{state } q_j \text{ at } t+1 | \text{ state } q_i \text{ at } t)$
$B = \{b_j(k)\}$ is a $N \times M$ matrix: $b_j(k) = P(\text{observation } k \text{ at } t | \text{ state } q_j \text{ at } t)$

Generally, a Hidden Markov model is defined by the matrices A, B and π and denoted as λ, such as $\lambda = (A, B, \pi)$.

For our given example, we have:
$T = 4$
$N = 2$
$M = 2$
$Q = \{S, R\}$
$V = \{H, O\}$
.

Assume a generic state sequence of length 4: $X = (x_0, x_1, x_2, x_3)$ with associated observations: $O = (O_0, O_1, O_2, O_3)$. A stated in the definition, π_0 is therefore the probability of beginning in the state x_0. Moreover, the probability of firstly observing O_0 is $b_{x_0}(O_0)$. Finally, a_{x_0,x_1} is the probability of transitioning to the state x_1 from the state x_0. Expanding these conclusions, the probability of the state sequence X is:

$$P(X, O) = \pi_{x_0} b_{x_0}(O_0) a_{x_0,x_1} b_{x_1}(O_1) a_{x_1,x_2} b_{x_2}(O_2) a_{x_2,x_3} b_{x_3}(O_3)$$

To find out the most probable state sequence based on our observation sequence, we compute the normalized probability (so that they all sum up to 1) of all state sequences of length 4. There is 2^4 such sequences given that we have two states: H and O. We listed them and the probability of each of them in the table 4.1.

state sequence	probability
HHHH	0.06332022499999998
HHHO	0.06274477499999999
HHOH	0.059754524999999996
HOHH	0.05943952499999998
OHHH	0.059867525
HHOO	0.065180475
HOHO	0.05964547499999996
OHHO	0.059417475
HOOH	0.06175822499999998
OOHH	0.06187522499999997
OHOH	0.05721022499999996
HOOO	0.06815677499999996
OHOO	0.06250477499999996
OOHO	0.062189774999999954
OOOH	0.06504952499999997
OOOO	0.07188547499999998

Table 4.1: Table of probabilities of each sequence

We have 4 positions and for each we summed the probabilities in the position of the states H and O in the table 4.2.

	position			
	0	1	2	3
P(H)	0.4999999999999999	0.4899999999999999	0.4884999999999999	0.4882749999999998
P(O)	0.4999999999999998	0.5099999999999998	0.5114999999999997	0.5117249999999998

Table 4.2: Table of probabilities of each state in every position

For each position, the HMM chooses the state with higher probability. We see that the state sequence with the highest probability in a HMM sense is: $HOOO$.

Why did we say *in a HMM sense*?

Another approach, to find out the sequence with the highest probability from the given table could be *dynamic programming* (DP) . In the sense of dynamic programming, we would choose a sequence of length 4 with the highest probability, which as we can see from the table is the sequence $OOOO$. The optimal results for the HMM and DP might differ, as they did in our case.

4.2.1 Algorithms used with HMM

HMM being a statistical model, it can be build a trained. There are various fields it is used in and different methods it is built with. In general, there are three central issues with Hidden Markov Model [13]:

1. **The Evaluation Problem**
   Given the model and the sequence of visible and observable symbols, we are looking for the probability that a particular sequence of states was generated by the model [25]. This is solved by the Forward and Backward algorithms, discussed later in this chapter.

2. **The Learning Problem**
   Estimate the high level structure of the model (number of hidden states and visible states). Once they are defined, estimate the transmission probability matrix and emission probability matrix using the training sequence [25]. This is done by the Forward-Backward Algorithm.

3. **The Decoding Problem**
   Lastly, we can use the model to predict the hidden states which generated the visible sequence [25]. This problem can be solved by Viterbi Algorithm [35].

In our case, we want to built a HMM for an existing process which will be introduced later. We provide a description of the algorithms used for solving all of the above problems.

Forward algorithm

Given the observation sequence O and the model $\lambda = (A, B, \pi)$, find $P(O|\lambda)$.

The first approach could be calculating all the probabilities of all the possibilities of the non-observable states given the observation sequence O. However, as you can imagine, that would be a tedious process. For a matrix with N hidden states and T observations, there are N^T operations needed. That would result in an exponential algorithm with a time complexity $O(N^T)$, in the *Big O Notation*. Instead of this brute force approach, HMM uses an ideology similar to dynamic programming, called forward algorithm, or α-pass.

For $t = 0, 1, ...T-1$ and $i = 0, 1, ...N-1$ define $\alpha_t(i) = P(O_0, O_1, ...O_t, x_t = q_i|\lambda)$ as a probability of the partial observation sequence up to time t, where the underlying Markov process is in the state q_i at time t.

We can compute a_t recursively:

1. Let $\alpha_0(i) = \pi_i b_i(O_0)$ for $i = 0, 1, ...N-1$.
2. Compute $\alpha_t(i) = \sum_{j=0}^{N-1}(\alpha_{t-1} j a_{ji}) b_i(O_t)$ for $t = 1, 2, ...T-1$ and $= 0, 1, ...N-1$.
3. Then from the definition of $\alpha_t(i)$ we see that $P(O|\lambda) = \sum_{i=1}^{N-1} \alpha_{T-1}(i)$.

This approach results in a way more efficient algorithm. α-pass has $O(N^2T)$ complexity, which is way better than $O(N^T)$.

Backward algorithm

Given the model $\lambda = (A, B, \pi)$ and an observation sequence O, find an optimal state sequence for the underlying Markov process.

To avoid algorithms with extreme complexity, HMM uses the Backward algorithm, or β-pass, to solve this problem. The Backward algorithm is analogous to the Forward algorithm, but as the name suggests, it goes backwards.

Define $\beta_t(i) = P(O_{t+1}, O_{t+2}, ...O_{T-1}|x_t = q_i, \lambda)$ recursively for $t = 0, 1, ...T-1$ and $i = 0, 1, ...N-1$:

1. Let $\beta_{T_1}(i) = 1$ for all $i = 0, 1, ...N-1$.
2. Compute $\beta_t(i) = \sum_{j=0}^{N-1} a_{ij} b_j(O_{t+1}) \beta_{t+1} j$ for $t = T-2, T-3, ...0$, $i = 0, 1, ...N-1$.
3. Define $\gamma_t(i) = P(x_t = q_i|O, \lambda)$ for $t = 0, 1, ...T-1$ and $i = 0, 1, ..N-1$.

From the previous algorithm, the α-pass, we know that $\alpha_t(i)$ calculates the probability up to time t. $\beta_t(i)$ measures the probability after time t. Then, $\gamma_t(i) = \frac{\alpha_t(i) \times \beta_t(i)}{P(O|\lambda)}$. Then, it follows that q_i is the state at the time t with the highest probability. $\gamma_t(i)$ is the maximum for q_i taken over the index i.

Forward-Backward algorithm

Given an observation sequence O and the dimensions N and M, find the model $\lambda = (A, B, \pi)$. This problem is basically training the model to best fit the observed data.

Let us define *di-gammas* as $\gamma_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1} j}{P(O|\lambda)}$ for $t = 0, 1, ..T-2$ and $i, j \in \{0, 1, ...N-1\}$. The $\gamma_t(i)$ and $\gamma_t(i, j)$ are related by $\gamma_t(i) = \sum_{j=0}^{N-1} \gamma_t(i, j)$ for $t = 0, 1, ...T-2$.

Given γ and di-gamma, we can verify that the model λ can be reestimated as:

1. Let $\pi_i = \gamma_0(i)$ for $i = 0, 1, ..N-1$.

2. Compute $a_{ij} = \frac{\sum_{t=0}^{T-2} \gamma_t(i,j)}{\sum_{t=0}^{T-2} \gamma_t(i)}$ for $i, j \in \{0, 1, ..N-1\}$.
   The numerator of this fraction gives the expected number of transitions from state q_i to state q_j, while the denominator gives the expected number of transitions from state q_i to any state. This calculates the probabilities of the desired values.

3. Compute $b_j(k) = \frac{\sum_{t \in \{0,1,..T-1\} O_t = k} \gamma_t(j)}{\sum_{t=0}^{T-1} \gamma_t(j)}$.

   We are measuring the probability of the wanted value again. The nominator gives the expected number of times the model λ is in the state q_j with the observation k. The denominator simply gives the number of times the model λ us in the state q_j with any observation.

The Forward-Backward algorithm, also known as the $\alpha - \beta$-pass, works as follows:

1. Initialize the model $\lambda = (A, B, \pi)$ by a guess or with some random values.

2. Compute: $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i)$, $\gamma_t(i,j)$.

3. Reestimate the model λ.

4. While $P(O|\lambda)$ increases, repeat from the computational part (step 2.).
   Whilst $P(O|\lambda)$ could increase indefinitely by some negligible small amount, it is recommended to set a minimal value of the differences between $P(O|\lambda)$ beetween the iteration, or to set a maximum number of iterations.

Viterbi algorithm

Given an observation sequence O and the model $\lambda = (A, B, \pi)$, find the most probable sequence of hidden states Q.

As finding all the possible scenarios of hidden states for the visible states and calculating their probabilities would result in a problem with complexity $O(N^T T)$ a much more efficient algorithm, Viterbi algorithm, is used with the complexity $O(TM)$ [35].

To begin with, we repeat these two steps for all given observations:

1. At first, we need to calculate the highest probability along a single path for first t observations which ends at state i.
   $\omega_i(t) = max_{q_1, \cdots, q_{T-1}} p(q_1, q_2, \cdots, q_{T=i}, o_1, o_2, \cdots, o_T|\lambda)$

2. At each step t, we need to find the state with maximal $\omega_i(t)$ in order to find the sequence of the hidden states.
   $\omega_i(t+1) = max_i\Big(\omega_i(t)a_{ij}b_{jkO(t+1)}\Big)$

Once we are done, we need to:

1. Firstly, we will identify the last hidden state by maximum likelihood.
2. Lastly, we will backtrack the process to find the sequence of hidden symbols with the highest probability

4.2.2 Hidden Markov models and Hankel matrices

It is known that a necessary condition for a stochastic process to have a HMM is that an associated Hankel matrix should have *finite rank* [34]. Even though this condition is necessary, it is not sufficient, which is proved in [6]. Therefore, we are not always able to construct a HMM of a process that has a Hankel matrix of a finite rank. However, it is proved [34] that for the processes whose Hankel rank is finite, it is always possible to construct a *quasi-realization* of such a process.

Chapter 5

Implementation

This chapter contains the implementation part of our work. In the first section, we introduce the technologies we chose to work with. The second section includes the problems we dealt with and our specific solutions. The last section sums up results regarding implementation.

5.1 Technologies

In this section, we discuss the technologies used in the development. We emphasize the reasons why we decided to use these technologies and what their benefits are.

5.1.1 The Rust programming language

The main reason we chose Rust as the programming language for this work is summed up at the beginning of The Book [16], a nicknamed book about Rust. It says that *The Rust programming language helps you write faster, more reliable software* [16], which is something that every developer wants. We find this programming language really amusing, as it balances out the oftentimes frustrating challenges it produces with the excitement it provides. Another benefit the language has is that *the Rust programming language is fundamentally about empowerment: no matter what kind of code you are writing now, Rust empowers you to reach farther, to program with confidence in a wider variety of domains than you did before* [16].

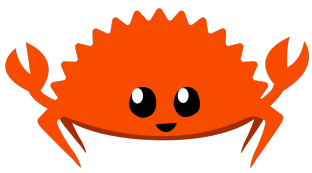

Figure 5.1: Ferris the Rustacean, unofficial mascot of Rust [33]

By choosing Rust as our programming language, we followed its strict ownership discipline and we avoided accidents with unsafe code. Rust provides easy expressing common C++ idioms in a safe way. We also did not have to deal with pointer invalidation as any related problems are solved by default. [15] Rust is also considered to be a green programming language, meaning it has low energy consumption. Compared to other programming languages, Rust is also blazing fast [1]. In an analysis of how energy memory, time relates rust performed very well in the executed tests. The results are shown in figure 5.2. Its type system and runtime guarantee the absence of data races, buffer overflows, stack overflows, and accesses to uninitialized or deallocated memory [19].

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Figure 5.2: Normalized global results for Energy, Time, and Memory [24]

5.1.2 Library ndarray

In our work, we need to operate with matrices in multiple ways. This library provides an n-dimensional general container for elements [32]. It also implements operations on them. There are many other libraries that provide the implementation of n-dimensional matrices and operations on them, such as `numpy`, which provides Rust interfaces for NumPy C APIs [30]. We chose `ndarray` because it was recommended in the Rust

Cookbook [7], which is *a collection of simple examples that demonstrate good practices to accomplish common programming tasks, using the crates of the Rust ecosystem* [7].

5.2 Generating Catalan numbers

As already mentioned in chapter 1, Catalan numbers can be generated in multiple ways. We implemented four of them and analyzed their complexities and expressed them in the *Big O Notation*.

5.2.1 From recurrent formula

The following function is a recursive function where the argument n decreases by 1. Therefore the function is called n times and the complexity is $O(n)$.

```
fn catalan_numbers_recursive(
    n: i64
) -> i64 {
    match n {
        0 => 1,
        _ => catalan_numbers_recursive(n - 1) * 2 * (2*n - 1) / (n + 1)
    }
}
```

5.2.2 From Pascal triangle

To evaluate the Catalan numbers from the Pascal triangle, we first need to build the Pascal triangle. We see that we call the function `fill_triangle` in the loop twice. Therefore the complexity of this function is $O(n \times 2n)$, which is $O(n^2)$.

```
fn from_pascal_triangle(n: i64) {
    let mut pct = vec![1; n:usize + 2];
    for i in 1..(n + 1) {
        fill_triangle(&mut pct, i);
        pct[i + 1] = pct[i];
        fill_triangle(&mut pct, i + 1);
        println!("{}", pct[i + 1] - pct[i]);
    }
}
```

```
fn fill_triangle(
    pct: &mut Vec<i64>,
    mut i: i64) {
    loop {
        if i == 1 { break; }
        pct[i] = pct[i] + pct[i - 1];
        i -= 1;
    }
}
```

5.2.3 Using an iterator

To get the *nth* term, we iterate from 1 to n. Therefore the complexity is $O(n)$.

```
struct Catalan {
    curr: i64,
    n: i64,
}

impl Iterator for Catalan {
    type Item = i64;

    fn next(&mut self) -> Option<i64> {
        self.curr = self.curr * 2 * (2 * self.n - 1) / (self.n + 1);
        self.n += 1;
        Some(self.curr)
    }
}
```

5.2.4 From the direct formula

Even though calculating the *nth* Catalan number from the direct formula may seem to be $O(1)$, we have to loop over n because of the factorial in it. Therefore the complexity is $O(n)$.

```
fn cn_nonrecursive(
    n: i64
) -> i64 {
    let mut cn = 1;
    for i in 0..n {
        cn *= 2 * n - i;
        cn /= i + 1;
    }
    return cn / (n + 1);
}
```

5.3 Matrix Operations

Even though the library `ndarray` provides most of the functionalities we need, to simplify some operations, we had to implement the following two functionalities by ourselves to fit our needs.

5.3.1 Horizontal stacking

Horizontal stacking is stacking arrays column wise. Acoording to scipy documentation, it *is equivalent to concatenation along the second axis, except for 1-D arrays where it concatenates along the first axis* [2]. The library `ndarray` provides the following function which stacks given arrays along the given axis [32].

```
stack<'a, A, D>(
    axis: Axis,
    arrays: &[ArrayView<'a, A, D>]
) -> Result<Array<A, D>, ShapeError>
```

Therefore, we used it for horizontal stacking accordingly. However, the given `arrays` might not have compatible shape, thus we need to return a `Result`.

```
fn hstack_from_stack<A, D>(
    arrays: &[ArrayView<A, D>]
) -> Result<Array<A, D>, ShapeError>
    where
        A: Copy,
        D: RemoveAxis,
{
    if arrays[0].ndim() == 1 {
        stack(Axis(0), arrays)
    } else {
        stack(Axis(1), arrays)
    }
}
```

5.3.2 Boolean masking

Boolean masking is a way of quantifying a sub-collection of a collection. It is an array of Boolean values based on some Boolean condition. In the Forward-Backward algorithm, in which we will use the Boolean masking, we want to create a hard mask, which is an array with the same dimensions as the one we are masking. We will only be masking one-dimensional arrays, therefore we used the *ndarray* `mapv` function.

```
fn create_hard_mask_for_1d(
    arr: &Array1<i32>,
    i: i32
) -> Array1<bool> {
    arr.mapv(|x| x == i)
}
```

Then, we needed to apply the mask on an array and return a new array. We will be masking a two-dimensional array with a vector. That means we want to apply the given mask on each element of the given two-dimensional array. Same as before, the given `mask` might not be compatible with the inner elements of the two-dimensional array `arr`. Therefore we return a `Result` again.

```
fn apply_mask_for_2d(
    arr: &Array2<f64>,
    mask: &Array1<bool>
) -> Result<Array2<f64>, ShapeError> {
    let res_columns = mask.iter().filter(|x| **x == true).count();
    let mut b = Array2::from_elem((arr.nrows(), res_columns), 0.0);
    let mut k = 0;

    for i in 0..arr.nrows() {
        for j in 0..arr.ncols() {
            if mask[j] {
                b[[i, k]] = arr[[i, j]] as f64;
                k += 1;
            }
        }
        k = 0;
    }
    Ok(b)
}
```

5.3.3 Applying element-wise operation

This functions is not part of the algorithm, but it helps us to eliminate values that would cause and *underflow error* [14]. We scaled those values using natural logarithm. The natural logarithm function `ln()` can be used in Rust on floats and reals. The parameter `arr` is an `ArrayView` of `f64` with the dimension `D`. We called the *ndarray* function `mapv` to apply the function `ln()` on each element of `arr`.

```
fn element_wise_logn<A, D>(
    arr: &ArrayView<A, D>
) -> Array<A, D>
    where
        A: Copy+Float,
        D: RemoveAxis,
{
    arr.mapv(|x| {x.ln()})
}
```

5.4 Hidden Markov Model algorithms

For building the model, we implemented *Forward Algorithm*, *Backward Algorithm* and *Forward-Backward Algorithm*. We also implemented *Viterbi Algorithm* for predicting the sequence of hidden states, given the observation sequence. All these algorithms were described in chapter 4 and can be found in the appendix.

Chapter 6

Experiments

In this chapter, we focus on working with data related to Catalan numbers and using it with Hidden Markov Models. We deliver the description of the data we chose to experiment with, the process and its results. Each section of this chapter is therefore dedicated to one kind of data we decided to work with.

The first step was to choose the data to build a Hidden Markov Model from. We chose sequences of symbols that are mentioned in chapter 1 in examples with Catalan numbers. These sequences will be our training data. Those are mostly sequences of two symbols, that are somehow (explained in chapter 1) enumerated by Catalan numbers.

The next step was to generate several of the possible sequences for a chosen number n. This way we get sequences of the same length ($2n$) and with the same characteristics (f.e. number of 0's never exceeds number of 1's). This will be the observation sequence for our Hidden Markov Model.

Then we create a sequence of hidden states for this sequence. We refer to it as to *the original sequence of hidden states*, because it is based on the observed data. These hidden states and their meaning will be further explained in each subsection. For each kind of data, we might experiment with different hidden states.

Using the *Forward-Backward Algorithm* described in chapter 4, we build a new Hidden Markov Model from each of the sequence in each section. There will be as many models as there are sequences.

In the next step, we use *Viterbi Algorithm* with each of the models to predict the sequence of the hidden states. This way we get a predicted sequence of hidden states for each of the Hidden Markov Models built with the observation sequence.

Each section is then summarized and the results are discussed in the last part of it. We also look at the behaviour of the models and examine possible similarities between the predicted hidden states that may arise. We repeated the experiment for different $n's$.

6.1 Lattice paths of n R's and n U's

This example and even its proof was already discussed in depth in chapter 2. An illustration of an *RU* path is shown in figures 2.1 and 2.2.

6.1.1 Choosing the data to study the lattice paths

To begin with, we pick n to be 100. We will generate various sequences of n R's and n U's such that the number of U's never exceeds the number of U's.

The total number of such sequences of length $2n$ is C_n. Out of curiosity, we used one of our algorithms described in chapter 5 to obtain the value of C_{100}. We found out that

$$C_{100} = 896519947090131496687170070074100632420837521538745909320.$$

As this number is really high, it would be difficult to generate all C_n sequences of length $2n$, build a Hidden Markov Model for each of them and study the data. Working with large amount of data might bring more results, however, it requires a different approach. We study smaller examples and rather examine possible connection of Hidden Markov Models and Catalan numbers. Any further study may be discussed in subsequent work.

We decided to define the hidden states for the observable states as follows. First, we label an observed symbol with 1 if it is a state in which the number of R's and U's is the same. This mean that the *RU* path touched the diagonal $y = x$. Otherwise we will mark the observed symbol as 0.

6.1.2 Experimenting with the lattice paths

First, we randomly generated an *RU* paths of lengths 100 and labeled the states with 1 when the number of R's and U's was the same and 0 otherwise. The generated observation sequences and their hidden states are shown in figures 6.1 and 6.2 for $n = 100$.

```
['R' 'U' 'R' 'R' 'U' 'U' 'R' 'U' 'U' 'R' 'U' 'R' 'U' 'R' 'U' 'R' 'U'
 'R' 'R' 'U' 'R' 'R' 'R' 'U' 'U' 'R' 'U' 'U' 'R' 'U' 'R' 'U' 'R' 'U'
 'U' 'R' 'U' 'R' 'R' 'R' 'R' 'U' 'R' 'U' 'U' 'U' 'U' 'R' 'R' 'R' 'U'
 'U' 'R' 'U' 'U' 'U' 'R' 'R' 'U' 'R' 'R' 'U' 'R' 'R' 'R' 'R' 'R' 'R'
 'U' 'R' 'R' 'R' 'R' 'U' 'U' 'R' 'R' 'R' 'U' 'R' 'U' 'U' 'U' 'U' 'U'
 'U' 'R' 'U' 'U' 'R' 'U' 'U' 'R' 'R' 'R' 'U' 'U' 'U' 'R' 'U' 'U' 'R'
 'R' 'U' 'R' 'R' 'R' 'R' 'R' 'U' 'U' 'U' 'U' 'R' 'U' 'U' 'R' 'R' 'U'
 'U' 'R' 'R' 'U' 'U' 'U' 'R' 'R' 'U' 'U' 'U' 'U' 'U' 'U' 'R' 'U' 'R'
 'R' 'U' 'R' 'R' 'U' 'U' 'U' 'U' 'U' 'R' 'R' 'R' 'R' 'R' 'R' 'U' 'R'
 'U' 'R' 'U' 'U' 'R' 'R' 'U' 'U' 'U' 'R' 'U' 'R' 'U' 'R' 'R' 'R' 'R'
 'R' 'R' 'R' 'U' 'R' 'R' 'U' 'U' 'R' 'U' 'R' 'R' 'U' 'U' 'R' 'R' 'U'
 'R' 'R' 'U' 'R' 'R' 'U' 'U' 'U' 'U' 'U' 'U' 'U' 'U']
```

Figure 6.1: Observed symbols (RU path) for $n = 100$

```
[0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 1
 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
```

Figure 6.2: Hidden states

We used the data to build a HMM. First, we read the observation sequence, and randomly initialized transition probability matrix and emission probability matrix. As we have only two states, we set the initial probability to be 0.5 for each. We build the HMM using the `forward_backward_algorithm` function with 10000 iterations. The higher the number of iterations is, the more similar will the resulting models be with differently initialized properties.

Before checking the actual results, we will describe what the model should look like based on the data we have. The output of the `forward_backward_algorithm` function is a pair of two matrices: the transition probability matrix and the emission probability matrix. As we have two observable states and two hidden states, their size will be 2×2 each. First, consider the transition probability matrix. We have two hidden states, 0 and 1, to be at at some step t. If we are in state 0 at the step t, it means that the number of R's and U's is not equal. What is the probability that at the step $t + 1$, we will be in the state 1, which means the number of R's and U's is be equal? As we see in the data in figure 6.2, there is not many 1's. Why is that so? When we create

a random sequence of n R's and n U's, the higher the n, the less often we will hit the point where the number of R's is the same as U's. Assume that the sequence has a point at which the number of R's is the same as the number of U's except for the last element. We split the sequence at the point where the number of R's is equal to the number of U's, we will call it m. This way, we would split a sequence of n R's and n U's of length $2n$ into two sequences of length m and $2n - m$. We know that the first sequence, of length m has the same number of R's and U's and the second sequence of length $2n - m$ has also the same number of R's and U's. As the first sequence of length m is a sub-sequence of the original sequence of the length $2n$, it holds the same properties and is basically an RU path of $\frac{m}{2}$ R's and $\frac{m}{2}$ U's. So there is $C_{\frac{m}{2}}$ ways to reorder the sequence of length m to preserve the property of having the same number of R's and U's, because every RU path has the same number of R's and U's. We also know that there is C_n RU paths of the original length $2n$. Catalan numbers grow exponentially even though the length of the grid does not. So the difference between the number of RU paths that touch the diagonal and those that do not (except for the ending point) will also increase exponentially.

So, now we know that there is less RU paths of length $2n$ that touched the diagonal at some point than those that did not touch it. Thus if we are at the state 0 in the step t (the number of R's is different that the number of U's), there is a higher probability that they will not be equal in the step $t + 1$ too. If we are at the state 1 in the step t (the number of R's is the same as the number of U's) then by adding either R or U we will get a sequence of uneven number of R's and U's. Therefore the probability of transitioning from state 1 to state 1 is 0.

For the emission probability matrix, when we are emitting 1 (the number of R's is equal to number of U's), there will be a 100% probability for that symbol to be U. The reason is that the RU path always starts with R (otherwise it would cross the diagonal) and at every point, the number of U's must not exceed the number of R's. If we were emitting 0 (the number of R's is not equal to number of U's), there should be a slightly higher probability for that symbol to be R. The reason is based again on the property of the sequence. Only by adding the symbol U to the sequence can make the sequence have equal number of R's and U's, leaving us with fewer U's to create a sequence where the number of R's and U's would not be equal.

We can see the resulting HMM defined by a transition probability matrix and emission probability matrix in the figure 6.3. We see that our predictions about the matrices were correct.

```
[[0.9538923352396265, 0.04610766476037175],
 [0.9999999999999997, 0.0000...]],
 shape=[2, 2], strides=[2, 1], layout=C (0x1), const ndim=2

[[0.5247381328434142, 0.4752618671565857],
 [0.0000..., 0.9999999999999998]],
 shape=[2, 2], strides=[2, 1], layout=C (0x1), const ndim=2
```

Figure 6.3: HMM

To continue our experiment, we will use the matrices from figure 6.3 as an input to our `viterbi_algorithm` function to obtain the predicted sequence of hidden states using the *Viterbi algorithm* from chapter 5. In the figure 6.4 we see the returned sequence.

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Figure 6.4: Predicted sequence of hidden states

6.1.3 The result of experimenting with the lattice paths

Our idea to create the hidden states based on the *RU* path touching the diagonal did not work well. There was probably too little number of those "touches" and they did not appear in any specific pattern. Another problem is that at each point, we knew the actual number of R's and U's based on all previous elements in the sequence. However, we created a Hidden Markov model of a constant order, which is not able to look at all the previous states. This way our data were labeled all with the same hidden state, 0. As we mentioned before, the higher the number n is, the bigger the difference between the number of states marked as 0 (the number of R's and U's is not the same) and the number of states marked as 1 (the number of R's and U's is equal). Therefore this way of labeling the data is not a good idea.

6.2 Arrangements of integers

In this section, we will work with arrangements of integers $1, 2, ...2n$ in which the odd integers occur in increasing order, the even integers occur in increasing order and for all k, where $1 \leq k \leq n$ $2k-1$ appears before $2k$. For example, for $n = 3$, there is $C_3 = 5$ possibilities to reorder a sequence $1, 2, 3, 4, 5, 6$:

- $1, 2, 3, 4, 5, 6$
- $1, 3, 5, 2, 4, 6$
- $1, 3, 2, 5, 4, 6$
- $1, 2, 3, 5, 4, 6$
- $1, 3, 2, 4, 5, 6$

6.2.1 Choosing the data to study the arrangements of integers

We generated one of the possible C_n arrangements of length $2n$ and considered the following hidden states. We labeled an item of the sequence as 1 if it was larger than the previous item or 0 otherwise.

If we build our model based on this data, it would have $2n$ states, since each number appears in the sequence only once. The probability of transitioning from k-th element to $k+1$ where $k \in \{1, 2, ..., 2n\}$ would be 100%. In a book by Ralph P. Grimaldi [11], a bijection from this occurrence of Catalan numbers to another one is shown by replacing even numbers with 1 and odd numbers with -1. To simplify our data, we also applied this replacement, however, we preserved the meanings of the hidden states.

6.2.2 Experimenting with the arrangements of integers

To begin with, we generated a sequence from 1 to $2n$ for $n = 100$. The generated data is displayed in figure 6.5.

```
[  1   3   5   7   2   9  11  13  15  17   4  19  21  23  25  27   6
  29  31  33   8  10  35  37  12  39  14  41  43  16  18  45  47  20
  49  51  22  53  24  26  55  57  59  61  28  30  32  63  34  65  67
  69  71  73  36  75  38  77  79  81  40  42  83  85  44  46  48  50
  52  54  87  56  89  58  60  91  93  62  64  66  68  70  95  72  97
  74  76  99  78 101  80  82  84 103  86 105 107 109  88  90  92 111
 113  94  96 115 117 119 121  98 123 100 102 104 125 106 108 110 112
 114 116 127 129 118 120 131 133 122 135 137 139 141 143 145 147 149
 124 151 153 126 128 155 130 157 132 159 161 134 163 165 167 169 136
 138 171 173 140 175 177 179 142 181 183 185 187 144 146 148 189 150
 152 154 191 156 158 193 160 195 197 162 199 164 166 168 170 172 174
 176 178 180 182 184 186 188 190 192 194 196 198 200]
```

Figure 6.5: Observed symbols (arrangement of integers)

Then we replaced an even element with 1 and an odd element with -1. The transformed data are shown in figure 6.6.

```
[-1 -1 -1 -1  1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1  1 -1 -1 -1  1  1 -1
 -1  1 -1  1 -1 -1  1  1 -1 -1  1 -1 -1  1 -1  1  1 -1 -1 -1 -1  1  1
  1 -1  1 -1 -1 -1 -1 -1  1 -1  1 -1 -1 -1  1  1 -1 -1  1  1  1  1  1
  1 -1  1 -1  1  1 -1 -1  1  1  1  1  1 -1  1 -1  1  1 -1  1 -1  1  1
  1 -1  1 -1 -1 -1  1  1  1 -1 -1  1  1 -1 -1 -1 -1  1 -1  1  1  1 -1
  1  1  1  1  1  1 -1 -1  1  1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1
 -1  1  1 -1  1 -1  1 -1 -1  1 -1 -1 -1 -1  1  1 -1 -1  1 -1 -1 -1  1
 -1 -1 -1 -1  1  1  1 -1  1  1  1 -1  1  1 -1  1 -1 -1  1 -1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
```

Figure 6.6: Transformed observed symbols

As seen in the figure 6.7, we labeled the elements from figure 6.5 with 1 if it was greater than its predecessor or 0 otherwise.

```
[1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 0 1 1 0 1 0 1 1 0 0 1 1 0 1
 1 0 1 0 0 1 1 1 1 0 0 0 1 0 1 1 1 1 1 0 1 0 1 1 1 0 0 1 1 0 0 0 0 0 0
 1 0 1 0 0 1 1 0 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 1 1 1 0 0 0 1 1 0 0
 1 1 1 1 0 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0
 0 1 0 1 0 1 1 0 1 1 1 1 0 0 1 1 0 1 1 1 0 1 1 1 1 0 0 0 1 0 0 0 1 0 0
 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Figure 6.7: Hidden states

The next step was to input these data to our `forward_backward_algorithm` function. The output matrices for the model are shown in figure 6.8.

```
[[0.5380553484464966, 0.4619446515535038],
 [0.48842075330828444, 0.5115792466917161]],
 shape=[2, 2], strides=[2, 1], layout=C (0x1), const ndim=2

[[0.4388792976416454, 0.5611207023583543],
 [0.5643828300326317, 0.43561716996736854]],
 shape=[2, 2], strides=[2, 1], layout=C (0x1), const ndim=2
```

Figure 6.8: HMM

We finished the process of this experiment with using the model to predict the hidden states on its own. We used the matrices from figure 6.8 along with the transformed sequence of observable sequence as an input to our `viterbi_algorithm` function. The output is shown in the figure 6.9.

```
[1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 0 1 1 0 1 0 1 1 0 0 1 1 0 1
 1 0 1 0 0 1 1 1 1 0 0 0 1 0 1 1 1 1 1 0 1 0 1 1 1 0 0 1 1 0 0 0 0 0 0
 1 0 1 0 0 1 1 0 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 1 1 1 0 0 0 1 1 0 0
 1 1 1 1 0 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0
 0 1 0 1 0 1 1 0 1 1 1 1 0 0 1 1 0 1 1 1 0 1 1 1 1 0 0 0 1 0 0 0 1 0 0
 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Figure 6.9: Predicted sequence of hidden states

6.2.3 The result of experimenting with the arrangements

From the generated data, the Hidden Markov model we built was able to learn to output the hidden states correctly. Unlike the model from the previous section, this model could predict the state s only by looking at $s - 1$. The hidden state of the observable state s was always defined only by the previous state $s - 1$, it did not need to look any further nor did the distance change. We conclude that we can use Hidden Markov model on an arrangement of integers (each element transformed to 1 if it is even, -1 if it is odd) to predict, whether the integer on the s-th position is greater than its predecessor on the position $s - 1$ or not.

6.3 Complete rooted binary trees on $2n + 1$ vertices

In this section, we will work with complete rooted binary trees on $2n + 1$ vertices. To begin with, we recapitulate the properties of such trees, to better understand what we are dealing with. A binary tree is a tree in which every node has at most two children. A complete binary tree is a tree in which every node, except for the leaves has exactly two children. A complete rooted binary tree is a tree with one node singled out that is called the root of the tree. Therefore the tree we will work with will have $2n + 1$ vertices, where the 1 stands for the root. We traverse these trees by going first left and then right if the node has children.

6.3.1 Choosing the data to study such binary trees

Once again we generated a sequence that corresponds to a complete rooted binary tree. We will label a vertex in a sequence of vertices as 1 if it is a leaf (meaning it has no children) or 0 otherwise. The hidden states of the sequence will be a sequence of 1's and 0's defined as follows.

To illustrate the process, consider the rooted ordered binary tree displayed in figure 6.10.

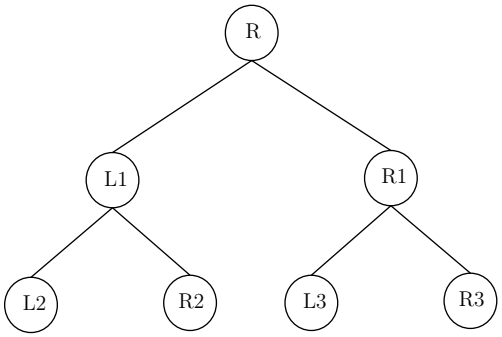

Figure 6.10: Complete rooted binary tree on 7 vertices

The sequence representing this tree would be: $R, L1, L2, R2, R1, L3, R3$. The leaves in this tree are the nodes $L2, R2, L3$ and $R3$. Therefore the original sequence of hidden symbols is $0, 0, 1, 1, 0, 1, 1$.

Another complete rooted binary tree with 7 vertices could be the tree shown in figure 6.11.

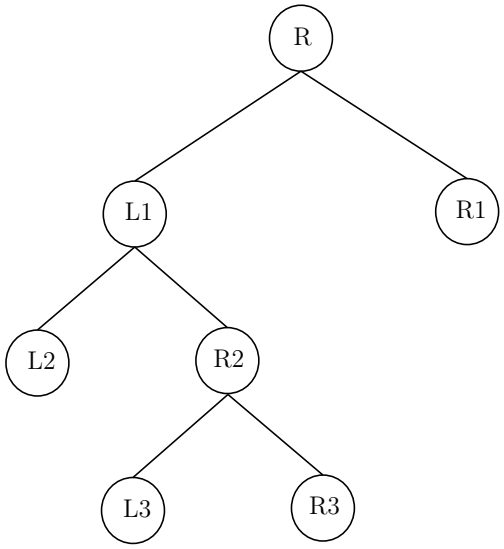

Figure 6.11: Another complete rooted binary tree on 7 vertices

Following the same rules, the sequence representing this tree would be the sequence: $R, L1, L2, R2, L3, R3, R1$. For this sequence, we would label the nodes $0, 0, 1, 0, 1, 1, 1$ to obtain the original sequence of hidden symbols in the same way as before.

To avoid the same problem we faced in the previous chapter, we need to find a simpler (much less than $2n + 1$ symbols) labeling for vertices to prevent the model having $2n + 1$ states. Simply we labeled left children as 0, right children as 1 and the root as 1. For the second complete rooted binary tree shown in figure 6.11, the new observation sequence would be $1, 0, 0, 1, 0, 1, 1$.

6.3.2 Experimenting with such binary trees

We generated a sequence of 1's and 0's of length $2n + 1$. The generated sequence can be seen in figure 6.12.

```
[1 0 0 1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 0 1 1 0 1 1 1 0 0 1 1 1 1 0 1 1 1
 0 1 1 0 1 1 0 1 0 0 0 1 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0 0 1 1 0 0 1 0 0
 0 0 0 1 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0 1 0 1 1 1 1 0 1 0
 1 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1 0 1 0 0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1
 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1 1 1 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0 1 1 0
 1 1 1 1 0 0 0 1 0 1 1 0 0 1 1 1 0 1 0 0 0 0 1 1 1 1 1]
```

Figure 6.12: Observed symbols (sequence of nodes)

We continued with labeling the data as described in the previous section. The original sequence of hidden symbols is shown in figure 6.13.

```
[0 0 1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 0 1 1 0 1 1 1 0 0 1 1 1 1 0 1 1 1 0
 1 1 0 1 1 0 1 0 0 0 1 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0 0 1 1 0 0 1 0 0 0
 0 0 1 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0 1 0 1 1 1 1 0 1 0 1
 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1 0 1 0 0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0
 0 0 0 0 0 1 0 0 1 1 0 0 0 1 1 1 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0 1 1 0 1
 1 1 1 0 0 0 1 0 1 1 0 0 1 1 1 0 1 0 0 0 0 1 1 1 1 1]
```

Figure 6.13: Hidden states

As before, we continued with running our *forward_backward_algorithm* function with the sequence from figure 6.12 as input. The resulting Hidden Markov model is displayed in figure 6.14.

```
[[0.4839714123823383, 0.5160285876176615],
 [0.43789676727408944, 0.5621032327259106]],
 shape=[2, 2], strides=[2, 1], layout=C (0x1), const ndim=2

[[0.4547717955206335, 0.5452282044793664],
 [0.5338397222179014, 0.4661602777820985]],
 shape=[2, 2], strides=[2, 1], layout=C (0x1), const ndim=2
```

Figure 6.14: HMM

The next step was to use this model to predict the hidden states of the sequence. We did this by running our *viterbi_algorithm* function described in chapter 5. The output sequence is visible figure 6.15.

```
[0 1 1 0 1 0 1 1 1 0 1 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 1 1 0 1 0 1 0 1 0
 1 1 0 1 1 0 1 0 1 1 1 0 1 0 1 1 0 1 1 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 0 1 1 1 1 0 1 1 0 1 1 0 1 1 1 1 0 1 0 1 1 1 0 1 1 0 1 0 1 0 1
 0 1 0 1 1 1 0 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0 1 0 1 1 0 1 1 1 0 1 0
 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 1 0 1
 1 0 1 0 1 1 1 0 1 1 0 1 1 0 1 0 1 0 1 1 1 1 1 0 1 0]
```

Figure 6.15: Predicted sequence of hidden states

Unfortunately, the predicted sequence of hidden states is not the same as the original one. However, we noticed a relationship between the elements of the sequences representing complete rooted binary trees. Consider an element v of the sequence representing a complete rooted binary tree. v is therefore a node of a tree (root, some other node's left or right child) and assume v is not a leaf. The tree is complete, hence v must have two children, left one and right one. In the sequence representing the binary tree, the next element after v must be its left child. Consider the opposite case, if v were a leaf. Then, it has no children, so the next element of the sequence will not be a successor of node v, but it must be from another branch of the tree. We traverse the tree from left to right, thus the branch in which v was (can be only one node) must be a left branch of some predecessor of v, let us call it w. After traversing the left subtree of w (the one where v is) we traverse the right subtree of w. The right children of w is the root of the right subtree of w. We know that the root of the subtree exists, because the whole tree is a complete tree and the left subtree of w consisted of at least one node, v. From these observations, we conclude that in a complete rooted binary tree represented by a sequence of 0's (for left children) and 1's (for right children and root) any node (either 0 or 1) is a leaf if the next element of the sequence is a right children (1) of some node.

However, Hidden Markov models are supposed to calculate the state s based on the state $s-1$. What we want is to know the state s based on the state $s+1$, which seems like the opposite. Therefore we decided to reverse the observed symbols and rerun the whole experiment.

The new sequence of observed symbols is shown in figure 6.16.

```
[1 1 1 1 0 0 0 0 1 0 1 1 1 0 0 1 1 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0
 1 0 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 1 1 0 0 1 1 0
 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1 1 0 1 0 1 1 1 1 0 1
 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 0
 0 0 1 0 1 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0 1 1 0 1 1 0 1 1 1 0 1 1 1 1 0
 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1 0 0 0 1 1 1 0 0 1]
```

Figure 6.16: Reversed observed symbols (sequence of vertices)

To match the new observation sequence, we also had to reverse the original sequence of hidden states. The result can be seen in figure 6.17.

```
[1 1 1 1 1 0 0 0 0 1 0 1 1 1 0 0 1 1 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1
 0 1 0 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 1 1 0 0 1 1
 0 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1 1 0 1 0 1 1 1 1 0
 1 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0
 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0 1 1 0 1 1 0 1 1 1 0 1 1 1 1
 0 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1 0 0 0 1 1 1 0 0]
```

Figure 6.17: Reversed sequence of hidden states

We created a new Hidden Markov model, displayed in figure 6.18.

```
[[0.48417259341878344, 0.5158274065812162],
 [0.4336225846482268, 0.5663774153517732]],
 shape=[2, 2], strides=[2, 1], layout=C (0x1), const ndim=2

[[0.452332414425285, 0.547667585574715],
 [0.5355601042313677, 0.46443989576863226]],
 shape=[2, 2], strides=[2, 1], layout=C (0x1), const ndim=2
```

Figure 6.18: HMM

Then we continued with the experiment by predicting the hidden states with our *viterbi_algorithm* function from chapter 5. The results are shown in figure 6.19.

```
[1 1 1 1 1 0 0 0 0 1 0 1 1 1 0 0 1 1 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1
 0 1 0 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 1 1 0 0 1 1
 0 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1 1 0 1 0 1 1 1 1 0
 1 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0
 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0 1 1 0 1 1 0 1 1 1 0 1 1 1 1
 0 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1 0 0 0 1 1 1 0 0]
```

Figure 6.19: Predicted sequence of hidden states

6.3.3 The result of experimenting with such binary trees

The data we chose to work with were chosen well as the results of the experiment are in the end successful. Unlike in the two previous sections, this time we had to reverse the data. This way, the Hidden Markov model displayed in 6.18 was able to learn to label the hidden states correctly. We conclude that we can use Hidden Markov model to find leaves in complete rooted binary trees represented by a reversed sequence of 1's and 0's where 0 stands for left children and 1 for right children and a root.

6.4 Result

In this chapter, we chose three situations, in which the Catalan number occur. The chosen examples were from distinct field, as we wanted to have different samples as the data. We studied the data, transformed it (if it was necessary) and constructed a Hidden Markov model from it. Then we used the model to predict the hidden states of the observation symbols. We were able to construct the model in all the cases, however, the model was able to learn to predict the hidden states correctly only in two of the three cases. In the end, we think the whole experiment was successful, because even the unsuccessful case led us to new findings and we were able to learn from it to do better in the other two experiments.

Conclusion

The goal of this thesis was to look for a connection between the Catalan numbers and the Hidden Markov models. In order to do so, we worked our way from the Catalan numbers discussed in the first chapter. We looked at their history and the formula. We continued by examining the various occurrences of the Catalan numbers. In order to be able to create connections to other fields, we looked at the Catalan numbers rather as one phenomenon in whose occurrences we could find the bijections between, then separate examples. From Catalan numbers we moved to the next part where we talked about the Hankel matrices. To get to the Hidden Markov models, we studied from Markov chains through Markov models.

Then we put these areas together and implemented algorithms for building Hidden Markov models and working with them. We chose the Rust programming language for the implementation and we ran into several obstacles during the development process (some of which related to the implementation language we chose). However, we were able to resolve them all and learn from them. We described and summarized the results of this thesis in the final chapter. We chose three quite distinct of occurrences of Catalan numbers to experiment with. We ended up with both successful and unsuccessful results.

Dealing with only a few examples and experimenting with them, we realized that the Catalan numbers provide numerous possibilities to be studied. We hope that this thesis will not only provide a new unusual view on the Catalan numbers from the angle of Hidden Markov models, but also motivate others to study this sequence as well. By tightening a gap between Catalan numbers and a concrete statistical model, Hidden Markov model, we hope to narrow the gaps between Catalan numbers and other field, as the results, one has the opportunity to come up with, can be surprising.

Bibliography

[1] Cosmin Cartas. Rust – the programming language for every industry. *ECONOMY INFORMATICS JOURNAL*, 19, 2019.

[2] The SciPy community. Scipy documentation, 2018. Accessed online on 18th of March 2020 at `https://docs.scipy.org/doc/numpy/reference/generated/numpy.hstack.html`.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 2009.

[4] Aleksandar Cvetkovic, Predrag Rajkovic, and Milos Ivkovic. Catalan numbers, the hankel transform and fibonacci numbers. *Journal of Integer Sequences*, 5, 2002.

[5] Tom Davis. Catalan numbers, 2010. Accessed online on 18th of November 2019 at `http://www.geometer.org/mathcircles/`.

[6] S. Dharmadhikari and M. Nadkarni. Some regular and non-regular functions of finite markov chains. *The Annals of Mathematical Statistics*, 41, 1970.

[7] Vigneshwer Dhinakaran. *Rust Cookbook*. Packt Publishing, 2017.

[8] Michael Dougherty, Christopher French, Benjamin Saderholm, and Wenyang Qian. Hankel transforms of linear combinations of catalan numbers. *Journal of Integer Sequences*, 14, 2011.

[9] Przemyslaw Dymarski. *HIDDEN MARKOV MODELS, THEORY AND APPLICATIONS*. InTech, 2011.

[10] Ralph P. Grimaldi. *Discrete and combinatorial mathematics*. Pearson Education, Inc., 2004.

[11] Ralph P. Grimaldi. *Fibonacci and catalan numbers : an introduction*. John Wiley & Sons, Inc., 2012.

[12] Vladimir Grujic. Generating functions of graph-catalan numbers. *Mathematics Subject Classification*, 2016.

[13] hmmlearn documentation. Accessed online on 10th of December 2019 at `https://hmmlearn.readthedocs.io/en/latest/tutorial.html`.

[14] J. Jeong, S. Choi, and C. Kim. A method to solve overflow or underflow errors resulting from activation functions in convolutional neural networks. *Information (Japan)*, 2017.

[15] Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. Safe systems programming in rust: The promise and the challenge, 2020. Accessed online on 1st of May 2020 at `https://www.semanticscholar.org/paper/Safe-Systems-Programming-in-Rust%3A-The-Promise-and-Jung-Germany/55601b2f884cf4e1bebc4fb409044ca0d3bb20e8`.

[16] Steve Klabnik and Carol Nichols. *The Rust Programming Language*. William Pollock, 2018.

[17] Kyu-Hwan Lee and Se-jin Oh. Catalan triangle numbers and binomial coefficients. *Contemporary Mathematics*, 2018.

[18] Desainte-Catherine M. and X. G. Viennot. Enumeration of certain young tableaux with bounded height. *Combinatoire Énumérative*, 1986.

[19] Nicholas D. Matsakis and Felix S. Klock. The rust language. *Ada Lett.*, 34(3), 2014.

[20] M. E. Mays and J. Wojciechowski. A determinant property of catalan numbers. *Discrete Math. 211*, 14, 2000.

[21] Mersenne numbers. Accessed online on 14th of December 2019 at `https://oeis.org/A000225`.

[22] S.P. Meyn and R.L Tweedie. *Markov chains and stochastic stability*. Springer, 1993.

[23] Igor Pak. History of catalan numbers. *Department of Mathematics, UCLA*, 2014.

[24] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Fernandes, and João Saraiva. Energy efficiency across programming languages: how do energy, time, and memory relate? *International Conference of Software Engineering*, 2017.

[25] Lawrence Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77, 1898.

[26] Mark Stamp. A revealing introduction to hidden markov models. *Department of Computer Science, San Jose State University*, 2018.

[27] Tanja Stojadinovic. The catalan numbers. *THE TEACHING OF MATHEMATICS*, 18, 2015.

[28] Ulrich Tamm. Some aspects of hankel matrices in coding theory and combinatorics. *Electronic Journal of Combinatorics*, 8, 2002.

[29] Toshiki Teramura. Crate ndarray_linalg, 2018. Accessed online on 26th of February 2020 at `https://docs.rs/ndarray-linalg/0.12.0/ndarray_linalg/`.

[30] Toshiki Teramura and Yuji Kanagawa. Crate numpy, 2018. Accessed online on 7th of March 2020 at `https://docs.rs/numpy/0.7.0/numpy/`.

[31] Anders Tolver. *An introduction to Markov chains*. University of Copenhagen, 2016.

[32] Jim Turner. Crate ndarray, 2018. Accessed online on 26th of February 2020 at `https://docs.rs/ndarray/0.13.0/ndarray/`.

[33] Karen Rustad Tölva. Ferris the Rustacean, 2018. Accessed online on 18th of March 2020 at `https://rustacean.net/`.

[34] M. Vidyasagar. The realization problem for hidden markov models: The complete realization problem. In *Proceedings of the 44th IEEE Conference on Decision and Control*, Dec 2005.

[35] A. J. Viterbi. Viterbi algorithm. *Scholarpedia*, 4, 2009. revision #91930.