Lecture - 20

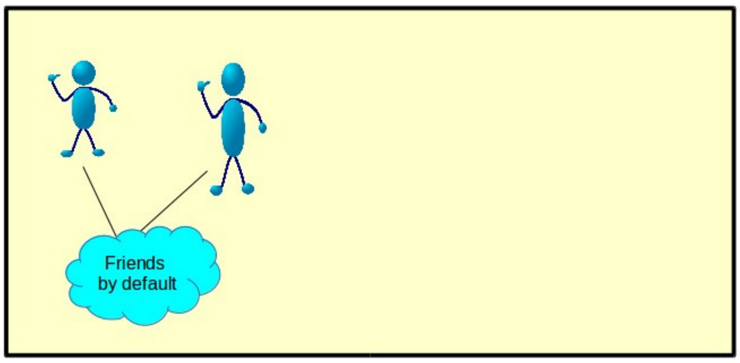
Friday, 9 September 2016 (16:15- 17:05)

Randomised Closest Pair Solution

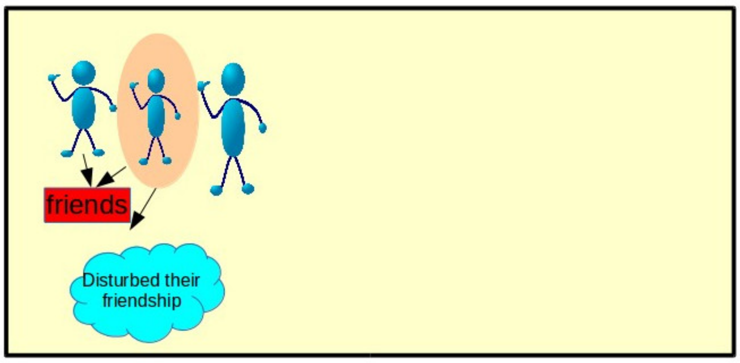
1 The Analogous Height Problem

Consider the scenario of a party, where people enter one by one and become friends in accordance with their height. The procedure works as follows.

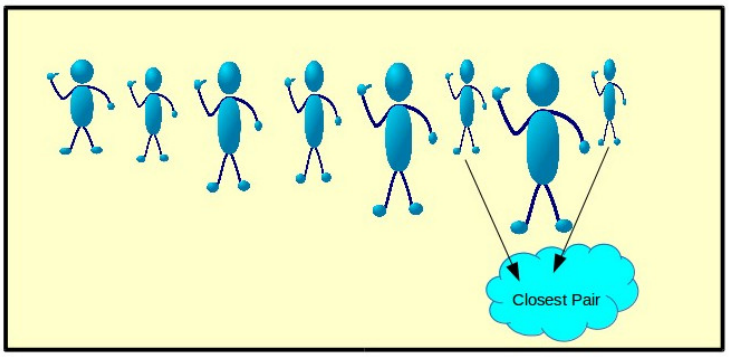
- The first person is, by default, friend with second person because there is no other person. Let the difference between their height be δ.



- When the third person comes, if his height difference with either of the persons is less than δ, he disturbs their friendship, and becomes friend with the person whom he is closer in height to.



- So, a newly arriving person, might or might not be, disturbing the already existing friendship. Please note that their is exactly one friendship which sustains. This is the friendship between the people who are closest in height.



- There can be any number of friendship break-ups in between depending on height of the people which enter the party.
- In the worst case, there can be $n-2$ break-ups.
- The question is - Given n such people in a party, how does one determine the two people in closest in height.

Randomised closest pair solution is also similar. Given n points in a plane, devise an algorithm to find the closest pair of points (Please note that we consider euclidean distance to find the closeness between the points).

Note: To solve these problems, we assume that the people/points in the second case, come by one by one to the picture and we change the answer to the best seen pair.

Brute Force Method: One way to solve this problem is to compare every incoming person with all the already arrived persons and change δ if necessary. The time complexity of this algorithm will be $O(n^2)$.

2 Probability that the i_{th} person disturbs the existing friendship

We will be using this result in the analysis of our algorithm later on.

When the i_{th} person arrives, $i-1$ people have already arrived. We also consider the people to arrive in random order without any height bias. In these i points, there is exactly one pair of people which has the least height difference. Let us denote these people as person A and person B. So the height between A and B is least. Now the i_{th} person will disturb an already existing friendship only if the i_{th} person is either A or B. It is because if the i_{th} person is some other person, he can not disturb the friendship between A and B which is the least.

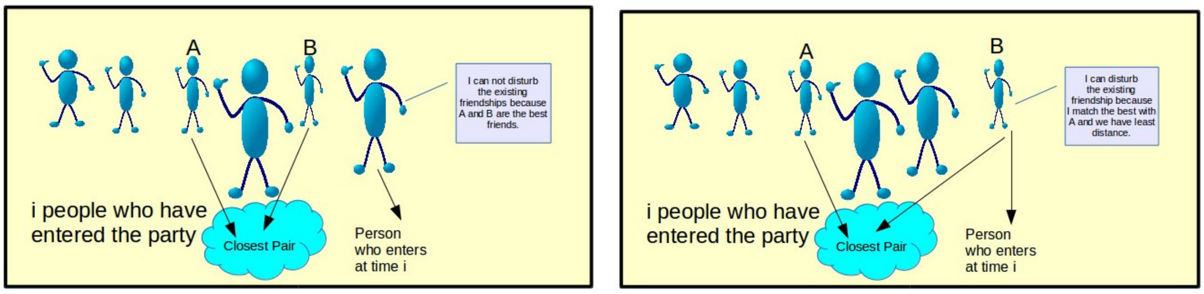

A
B
I can not disturb the existing friendships because A and B are the best friends.
i people who have entered the party
Closest Pair
Person who enters at time i
A
B
I can disturb the existing friendship because I match the best with A and we have least distance.
i people who have entered the party
Closest Pair
Person who enters at time i

Fig. 1: **Left**: Person i does not disturb the already existing friendship. **Right**: Person i disturbs the friendship

Hence, there are only two options where the last person disturbs the existing friendship and results in a breakup. It is only when this person is either A or B. Hence the probability that i_{th} person results in a breakup is $\frac{2}{i}$.

3 Randomised Closest Pair Solution

Consider an XY plane, which has n points. We have to devise an algorithm to find the nearest paper.

3.1 The Algorithm

Let the distance between first two points which have arrived is δ.
We divide the entire XY plane in squares with lengths of height $\frac{\delta}{2}$ as shown in Figure.

Now, a new point comes. We wish to ensure that there should be no existing point within a distance of δ from it. If there exists such a point, then this new point will disturb the closest distance existing till now. How do we ensure that?

What if a point lies in the same block as the new point As shown in the Figure 3, in this case, the new point is definitely going to change the closest pair. It is because any point lying in the same square as this box will be within a distance δ from it. In the worst case (when the points are farthermost apart), this distance will be $\frac{\delta}{\sqrt{2}}$, as shown by the red points in the Figure 3.

What if a point lies in the adjacent block as the new point As shown in the Figure 4, if the new point lies in the adjacent block to any of the old points, there are the chances of disturbance(change of the closest pair), so the adjacent blocks need to be checked.

What if a point lies in the adjacent to adjacent block as the new point It is still possible that the two points lie at a distance of less than δ from each other as shown in figure 5

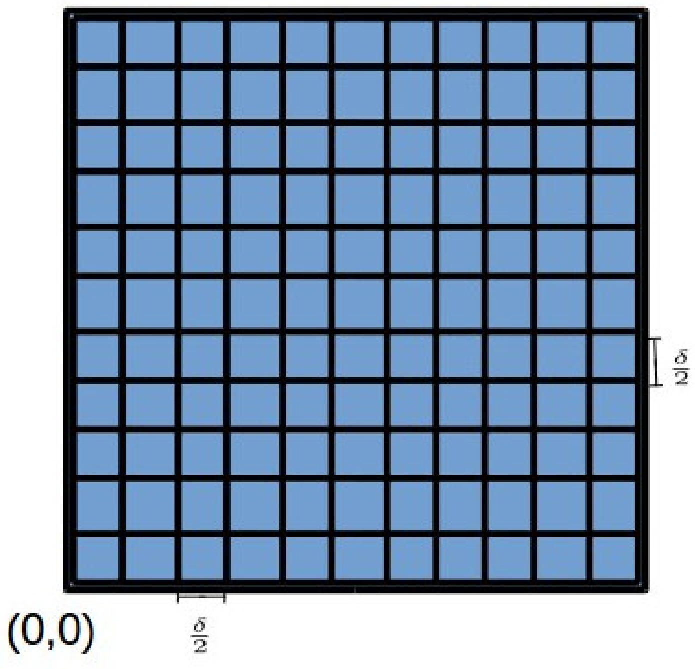


Fig. 2: The grid

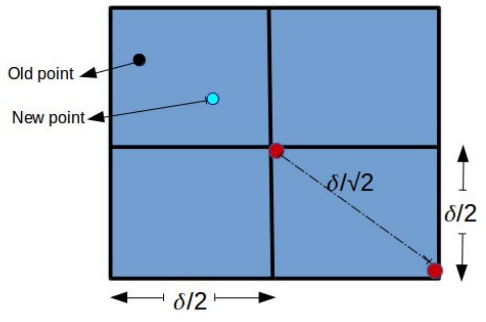


Fig. 3: Same Block

3.2 What if a point lies in the adjacent to adjacent to adjacent block as the new point

As shown in Figure 6, no matter how close we try to keep the points, there will be a distance of atleast δ between the two points.

Therefore, when adding a new point, we need to check only the 25 adjacent blocks to see if there is an old point within a proximity of δ. If there is a point, we change the δ to the new closest distance and then again divide the XY plane.

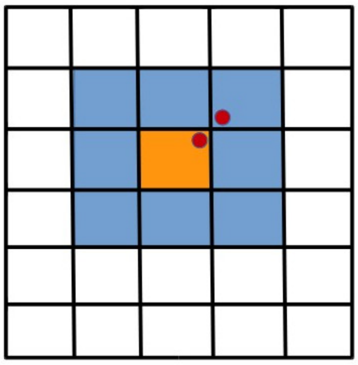


Fig. 4: Adjacent Blocks

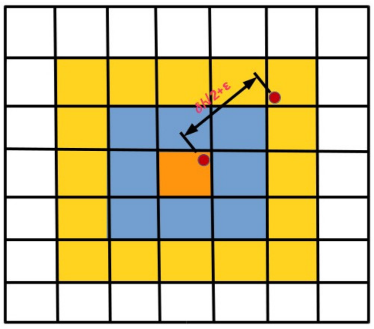


Fig. 5: Adjacent to adjacent Blocks

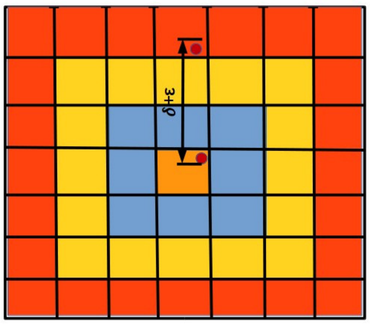


Fig. 6: Adjacent to adjacent Blocks

3.3 Re-meshing the XY plane

We seek help of the hashing technique to do this.

Hashing Assume we want to store the phone numbers of students of a class in a data structure such that we can quickly retrieve one when demanded, or check for the absence or presence of a

phone number.
One way to do this is to make an array and store all the number there. But this technique turns out to be inefficient because it taken $O(n)$ time to search for an entry here. Moreover, when a new entry has to be entered or deleted, all the elements have to be shifted down or up.

In hashing, we store the elements according to a function. Assume the elements to be hashed are $\{a_1, a_2, ..., a_n\}$. Then, we have a function $f(a_i) = k_i$. We store a_i at the k_ith place. If there is already an element at the place k_i, we create a chained list there and add the element. This is illustrated in the following example.

Let there are some phone numbers we want to store. Our key if $f(a_i) = a_i\%10$. Then we will get a hash table like the one shown in Figure 7.

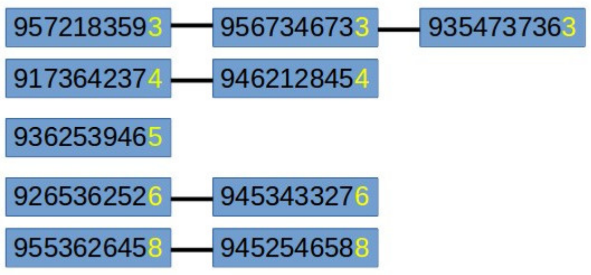


Fig. 7: Hash Table

Now, we know that an entry a_i will be at the location $a_i\%10$ in the hash table. We might need to search across a few items in case the entry is in a chained list. For example- we know that the phone number 9354737363 will be found at location 3, we still have to search across 3 elements, since there are 3 elements chained at this location. But, if the elements have a uniform distribution, this will take a constant amount of time only. Hence, a search operation takes $O(1)$ times in a hash table setting. Insertion and deletion of an element also takes $O(1)$ time.

Re-meshing : First assume the value of δ to be 1 for the sake of simplicity. We know that the entire XY plane is a mesh of unit squares. What we are interested in, is to find the block number of a particular element. In this setting, the point (x, y) will belong to the block (x, y). As shown in the Figure 8, all the points $((2, 1)$ ... $(3, 2)]$ lie in block $(3, 2)$.

Given a point, now it is very easy to determine its block number. The block number of a point (x, y) will be in the grid defined by the function :
$g(x, y) = (\lceil x \rceil, \lceil y \rceil)$
The intelligent reader can easily decide on the boundary conditions. So, given a point now, we clearly know now, in which block it is.

We can hash the block numbers of the existing points in the hash table, in the same way as we did above for the phone numbers. Now, when a new point comes, we need to see 25 blocks. So if the block number of the new point is (x, y), we need to check for 25 blocks - $(x, y), (x + 1, y), (x, y +$

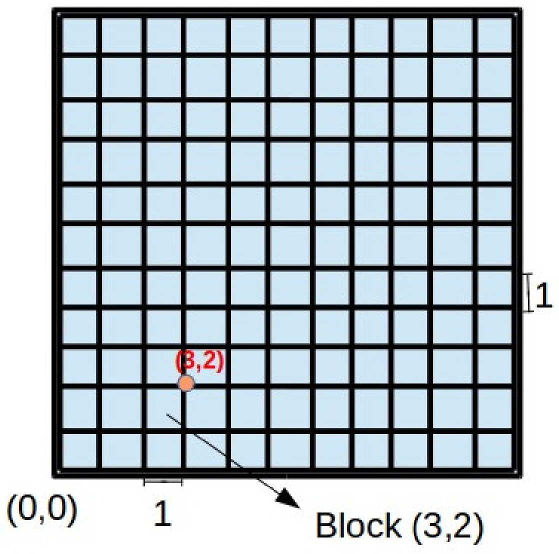


Fig. 8: Getting block number of a point

$1), (x+1, y+1), (x-1, y), (x, y-1), (x-1, y-1), (x+2, y), (x, y+2), (x-1, y+1), (x+1, y-1), (x+2, y+2), (x-2, y), (x, y-2), (x-2, y-2), (x+2, y+1), (x+1, y+2), (x-2, y+1), (x+1, y-2), (x+2, y-1), (x-1, y+2), (x-2, y-1), (x-1, y-2), (x-2, y+2), (x+2, y-2)$

We clearly know now the entries in the hash table, which need to be checked. This can be done in a constant time.

The length of the square is δ, not 1 : How do we extend the above method if the length of the square is δ?

As shown in the Figure 9, now we know very clearly that the points lying from $((0,0)....(0.5, 0.5)]$ lie in block $(1,1)$. Similarly the point $(2.5, 1.5)$, which would have previously lied in block $(3,2)$ will now lie in the block $(6,4)$. It is just a simple scaling. Since the scale is halved, the block number doubles.

Hence, if the length of a square is δ, its block number is given by $(\delta \times \lceil x \rceil, \delta \times \lceil y \rceil)$.

When a point causes a disturbance: When a point causes a disturbance, i.e. when at least one of the old points lie at a distance less than δ from it, we need to change *delta* and perform the re-meshing again.

Please note now that re-meshing does not mean anything other than creating the hash table again. We are not doing any actual re-meshing, it is a virtual meshing being depicted by the hash table.

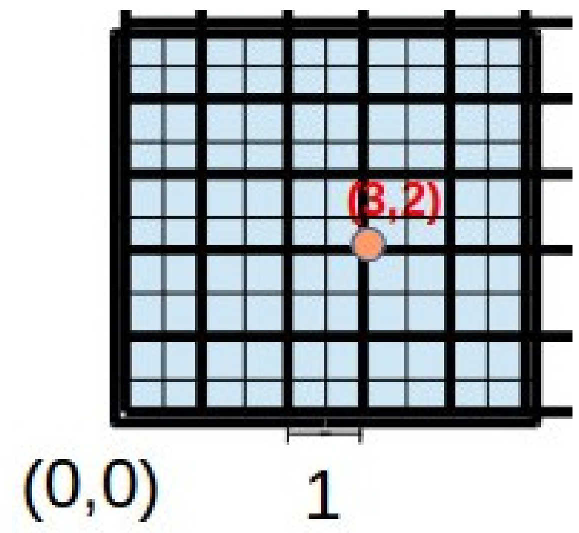


Fig. 9: δ=0.5

So, when there is a disturbance, we have to re-mesh, which means, create the hash table again, perform n deletions and n insertions. Hence, re-meshing is done in $O(n)$ time.

4 Analysis

As we saw, every time a new point comes, we need to check a constant number of blocks. We need to re-mesh the XY plane only in the case when there is a point having another point within a proximity of δ. As we saw earlier, the probability of this happening for the i_{th} point is $\frac{2}{i}$. Hence, the expected number of disturbances= $\sum_{i=1}^{n} \frac{2}{i}$
$=2\sum_{i=1}^{n} i$ $=2(\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+.....+\frac{n}{i})$
$=O(\log n)$

Each of these $\log n$ times, we need to re-mesh the XY plane which takes $O(n)$ times. Hence the overall complexity = $O(n \log n)$