CSS.201.1 Algorithms

Instructor: Umang Bhaskar

TIFR 2024, Aug-Dec

Scribe: Soham Chatterjee

sohamchatterjee999@gmail.com
Website: sohamch08.github.io

Contents

Chapter 1

Finding Closest Pair of Points

Find-Closest(S)
Input: Set $S = \{(x_i, y_i) \mid x_i, y_i \in \mathbb{R},\ \forall\ i \in [n]\}$. We denote $P_i = (x_i, y_i)$.
Question: Given a set of points find the closest pair of points in $\mathbb{R}^2$ find P_i, P_j that are at minimum l_2 distance i.e. minimize $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

1.1 Naive Algorithm

Now the naive algorithm for this will be checking all pairs of points and take their distance and output the minimum one. There are total $\binom{n}{2}$ possible choices of pairs of points. And calculating the distance of each pair takes $O(1)$ time. So it will take $O(n^2)$ times to find the closest pair of points.

Idea: $\forall\ P_i, P_j \in S$ find distance $d(P_i, P_j)$ and return the minimum. Time taken is $O(n^2)$.

1.2 Divide and Conquer Algorithm

Below we will show a Divide and Conquer algorithm which gives a much faster algorithm. We will follow the algorithm from [CTL+, Section 33.4]

Definition 1.2.1: Divide and Conquer

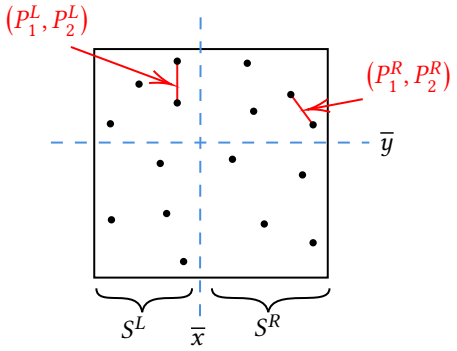
- Divide: Divide the problem into two parts (roughly equal)
- Conquer: Solve each part individually recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
- Combine: Combine the solutions to the subproblems into the solution.

1.2.1 Divide

So to divide the problem into two roughly equal parts we need to divide the points into two equal sets. That we can do by sorting the points by their $x-$coordinate. Suppose S^x denote we get the new sorted array or points. And similarly we obtain S^y which denotes the array of points after sorting S by their $y-$coordinate.

Algorithm 1: Step 1 (Divide)

Function *Divide*:
- Sort S by $x-$coordinate and $y-$coordinate
- $S^x \longleftarrow S$ sorted by $x-$coordinate
- $S^y \longleftarrow S$ sorted by $y-$coordinate
- $\bar{x} \longleftarrow \lfloor \frac{n}{2} \rfloor$ highest $x-$coordinate
- $\bar{y} \longleftarrow \lfloor \frac{n}{2} \rfloor$ highest $y-$coordinate
- $S^L \longleftarrow \{P_i \mid x_i < \bar{x},\ \forall\ i \in [n]\}$
- $S^R \longleftarrow \{P_i \mid x_i \geq \bar{x},\ \forall\ i \in [n]\}$



1.2.2 Conquer

Now we will recursively get pair of closest points in S_L and S_R. Suppose the (P_1^L, P_2^L) are the closest pair of points in S^L and (P_1^R, P_2^R) are the closest pair of points in S^R.

Algorithm 2: Step 1 (Solve Subproblems)

Function *Conquer*:
- Solve for S_L, S^R.
- (P_1^L, P_2^L) are closest pair of points in S_L.
- (P_1^R, P_2^R) are closest pair of points in S_R.
- $\delta^L = d(P_1^L, P_2^L)$, $\delta^R = d(P_1^R, P_2^R)$
- $\delta_{min} \longleftarrow \min\{\delta^L, \delta^R\}$

1.2.3 Combine

Now we want to combine these two solutions.

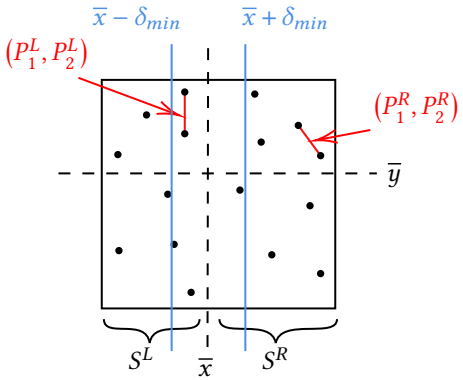
Question 1.1: We are not done

Is there a pair of points $P_i, P_j \in S$ such that $d(P_i, P_j) < \delta_{min}$

If Yes:

- One of them must be in S_L and the other is in S_R.
- $x-$coordinate $\in [\bar{x} - \delta_{min}, \bar{x} + \delta_{min}]$.
- $|y_i - y_j| \leq \delta_{min}$

So we take the strip of radius δ_{min} around $\bar{x}$. Define $T = \{P_i \in S \mid |x_i - \bar{x}| \leq \delta_{min}\}$



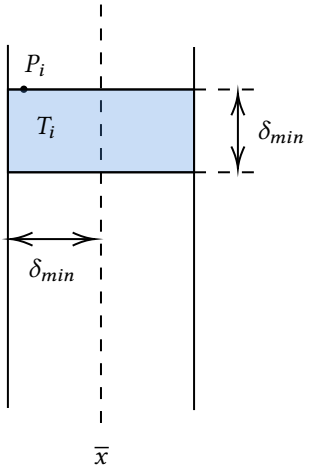
We now sort all the points in the T by their decreasing $y-$coordinate. Let T_y be the array of points. For each $P_i \in T_y$ define the region

$$T_i = \{P_j \in T_y \mid 0 \leq y_j - y_i \leq \delta_{min}, j > i\}$$

Lemma 1.2.1

Number of points (other than P_i) that lie inside the box is at most 8

Proof: Suppose there are more than 8 points that lie inside the box apart from P_i. The box has a left square part and a right square part. So one of the squares contains at least 5 points. WLOG suppose the left square has at least 5 points. Divide each square into 4 parts by a middle vertical and a middle horizontal line. Now since there are 5 points there is one part which contains 2 points but that is not possible as those two points are in S_L and their distance will be less than δ_{min} which is not possible. Hence contradiction. Therefore there are at most 8 points inside the box. ■



Hence by the above lemma for each $P_i \in T_y$ there are at most 8 points in T_i. So for each $P_j \in T_i$ we find the $d(P_i, P_j)$ and if it is less than δ_{min} we update the points and the distance

1.2.4 Pseudocode and Time Complexity

Assumption. *We will assume for now that for all $P_i.P_j \in S$ we have $x_i \neq x_j$ and $y_i \neq y_j$. Later we will modify the pseudocode to remove this assumption*

Algorithm 3: FIND-CLOSEST(S)

Input: Set of n points, $S = \{(x_i, y_i) \mid x_i, y_i \in \mathbb{R},\ \forall\ i \in [n]\}$. We denote $P_i = (x_i, y_i)$.
Output: Closest pair of ponts, (P_i, P_j, δ) where $\delta = d(P_i, P_j)$

```
1  begin
2      if |S| ≤ 10 then
3          Solve by Brute Force (Consider every pair of points)
4      S^x ⟵ S sorted by x−coordinate,        S^y ⟵ S sorted by y−coordinate
5      x̄ ⟵ ⌊n/2⌋ highest x−coordinate,        ȳ ⟵ ⌊n/2⌋ highest y−coordinate
6      S^L ⟵ {P_i | x_i < x̄, ∀ i ∈ [n]},      S^R ⟵ {P_i | x_i ≥ x̄, ∀ i ∈ [n]}
7      (P_1^L, P_2^L, δ^L) ⟵ FIND-CLOSEST(S^L),      (P_1^R, P_2^R, δ^R) ⟵ FIND-CLOSEST(S^R)
8      δ_min ⟵ min{δ^L, δ^R}
9      if δ_min < δ^L then
10         P_1 ⟵ P_1^R, P_2 ⟵ P_2^R
11     else
12         P_1 ⟵ P_1^L, P_2 ⟵ P_2^L
13     T ⟵ {P_i | |x_i − x̄| ≤ δ_min}
14     T_y ⟵ T sorted by decreasing y−coordinate
15     for P ∈ T_y do
16         U ⟵ Next 8 points
17         for P̂ ∈ U do
18             if d(P, P̂) < δ_min then
19                 δ_min ⟵ d(P, P̂)
20                 (P_1, P_2) ⟵ (P, P̂)
21     return (P_1, P_2, δ_min)
```

Notice we used the assumption in the line 5 for finding the medians. So the line 4 takes $O(n \log n)$ times. Lines 5,6 takes $O(n)$ time. Since $\overline{x}$ is the median, we have $|S^L| = \lfloor \frac{n}{2} \rfloor$ and $|S^R| = \lceil \frac{n}{2} \rceil$. Hence FIND-CLOSEST($S^L$) and FIND-CLOSEST($S^R$) takes $T\left(\frac{n}{2}\right)$ time. Now lines $8-12$ takes constant time. Line 13 takes $O(n)$ time. And line 14 takes $O(n \log n)$ time. Since U has 8 points i.e. constant number of points the lines $16-20$ takes constant time for each $P \in T_y$. Hence the for loop at

line 15 takes $O(n)$ time. Hence total time taken

$$T(n) = O(n) + O(n \log n) + 2T\left(\frac{n}{2}\right) \implies T(n) = O(n \log^2 n)$$

1.3 Improved Algorithm for $O(n \log n)$ Runtime

Notice once we sort the points by $x-$coordinate and $y-$coordinate we don't need to sort the points anymore. We can just pass the sorted array of points into the arguments for solving the smaller problems. Their is another time where we need to sort which is in line 14 of the above algorithm. This we can get actually from S^y without sorting just checking one by one backwards direction if the $x-$coordinate of the points satisfy $|x_i - \overline{x}| \leq \delta_{min}$. So

$$T_y = \textsc{Reverse}(\{P_i \in S^y \mid |x_i - \overline{x}| \leq \delta_{min}\})$$

So we form a new algorithm which takes the input S^x and S^y and then finds the closest pair of points. Then we will use that subroutine to find closest pair of points in any given set of points.

Algorithm 4: Find-Closest-Sorted(S^x, S^y)

Input: Set of n points, $S = \{(x_i, y_i) \mid x_i, y_i \in \mathbb{R},\ \forall\ i \in [n]\}$. S^x and S^y are the sorted array of points with respect to $x-$coordinate and $y-$coordinate respectively

Output: Closest pair of ponts, (P_i, P_j, δ) where $\delta = d(P_i, P_j)$

```
1  begin
2      if |S| ≤ 10 then
3          Solve by Brute Force
4      x̄ ⟵ ⌊n/2⌋ highest x−coordinate
5      ȳ ⟵ ⌊n/2⌋ highest y−coordinate
6      S^L ⟵ {P_i ∈ S^x | x_i < x̄, ∀ i ∈ [n]}
7      S^L_y ⟵ {P_i ∈ S^y | x_i < x̄}
8      S^R ⟵ {P_i ∈ S^x | x_i ≥ x̄, ∀ i ∈ [n]}
9      S^R_y ⟵ {P_i ∈ S^y | x_i ≥ x̄}
10     (P^L_1, P^L_2, δ^L) ⟵ Find-Closest-Sorted(S^L, S^L_y)
11     (P^R_1, P^R_2, δ^R) ⟵ Find-Closest-Sorted(S^R, S^R_y)
12     δ_min ⟵ min{δ^L, δ^R}
13     if δ_min < δ^L then
14         P_1 ⟵ P^R_1, P_2 ⟵ P^R_2
15     else
16         P_1 ⟵ P^L_1, P_2 ⟵ P^L_2
17     T ⟵ {P_i | |x_i − x̄| ≤ δ_min}
18     T_y ⟵ Reverse({P_i ∈ S^y | |x_i − x̄| ≤ δ_min})
19     for P ∈ T_y do
20         U ⟵ Next 8 points
21         for P̂ ∈ U do
22             if d(P, P̂) < δ_min then
23                 δ_min ⟵ d(P, P̂)
24                 (P_1, P_2) ⟵ (P, P̂)
25     return (P_1, P_2, δ_min)
```

Algorithm 5: Find-Closest(S)

Input: Set of n points, $S = \{(x_i, y_i) \mid x_i, y_i \in \mathbb{R},\ \forall\ i \in [n]\}$. We denote $P_i = (x_i, y_i)$.

Output: Closest pair of ponts, (P_i, P_j, δ) where $\delta = d(P_i, P_j)$

```
1  begin
2      if |S| ≤ 10 then
3          Solve by Brute Force
4      S^x ⟵ S sorted by x−coordinate
5      S^y ⟵ S sorted by y−coordinate
6      return Find-Closest-Sorted(S^x, S^y)
```

This algorithm only sorts one time. So time complexity for Find-Closest-Sorted(S^x, S^y) is

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \implies T(n) = O(n \log n)$$

and therefore times complexity for Find-Closest(S) is $O(n \log n)$.

1.4 Removing the Assumption

For this there nothing much to do. For finding the median $\overline{x}$ if we have more than one points with same $x-$coordinate which appears as the $\lfloor \frac{n}{2} \rfloor$ highest $x-$coordinate we sort only those points with respect to their $y-$coordinate update the S^x like that and then take $\lfloor \frac{n}{2} \rfloor$ highest point in S^x. We do the same for S^y and update accordingly. All this we do so that S^L and S^R has the size $\frac{n}{2}$.

Chapter 2

Median Finding in Linear Time

Median-Find(S)
Input: Set S of n distinct integers
Question: Find the $\lfloor \frac{n}{2} \rfloor^{th}$ smallest integer in S

2.1 Naive Algorithm

The naive algorithm for this will be to sort the array in $O(n \log n)$ time then return the $\lfloor \frac{n}{2} \rfloor^{th}$ element. This will take $O(n \log n)$ time. But in the next section we will show a linear time algorithm.

2.2 Linear Time Algorithm

In this section we will show an algorithm to find the median of a given set of distinct integers in $O(n)$ time complexity. We will follow [CTL+, Section 9.3]. Consider the following two problems:

Rank-Find (S, k)
Input: Set S of n distinct integers and an integer $k \leq n$
Question: Find the k^{th} smallest integer in S

Approximate-Split(S)
Input: Set S of n distinct integers
Question: Given S, return an integer $z \in S$ such that z where $rank(z) \in \left[\frac{n}{4}, \frac{3n}{4}\right]$

2.2.1 Solve Rank-Find using Approximate-Split

Algorithm 6: Rank-Find(S,k)

Input: Set S of n distinct integer and $k \in [n]$
Output: k^{th} smallest integer in S

```
1 begin
2     if |S| ≤ 100 then
3         Sort S, return k^th smallest element in S
4     z ⟵ Approximate-Split(S)        (z is the r^th smallest element for some r ∈ [n/4, 3n/4])
5     S_L ⟵ {x ∈ S | x ≤ z}, S_R ⟵ {x ∈ S | x > z}
6     if k ≤ |S_L| then
7         return Rank-Find(S_L, k)
8     return Rank-Find(S_R, k − |S_L|)
```

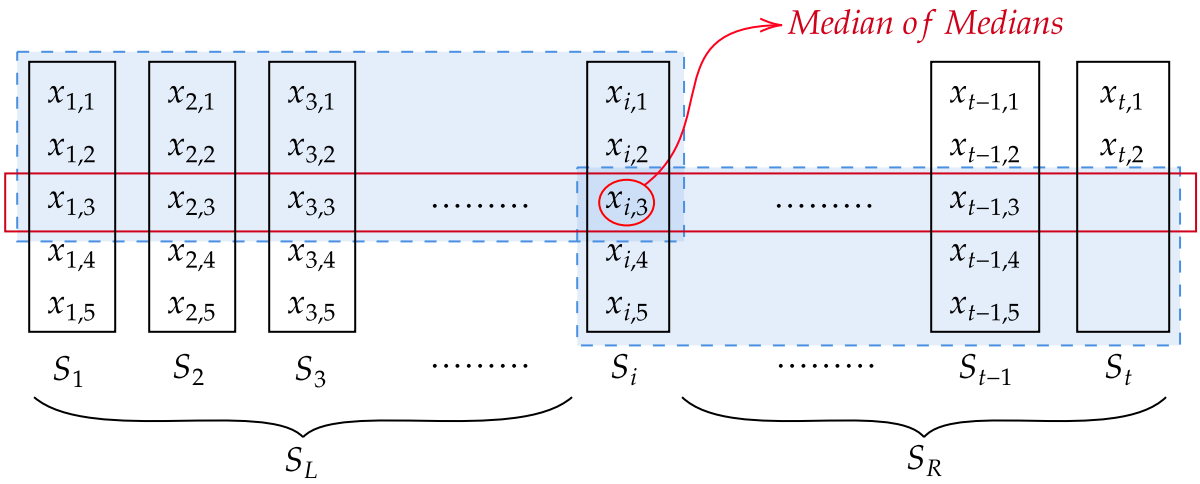
Certainly if we can solve RANK-FIND(S, k) for all $k \in [n]$ we can also solve MEDIAN-FIND. We will try to use both the problems and recurse to solve RANK-FIND in linear time.

In the above algorithm $rank(z) \in \left[\frac{n}{4}, \frac{3n}{4}\right]$. So $\frac{n}{4} \leq |S_L|, |S_R| \leq \frac{3n}{4}$. For now suppose RANK-FIND(S, k) takes $T_{RF}(n)$ time and APPROXIMATE-SPLIT(S) takes $T_{AS}(n)$ time. Then the time taken by the algorithm is

$$T_{RF}(n) \leq O(n) + T_{AS}(n) + T_{RF}\left(\frac{3n}{4}\right)$$

2.2.2 Solve APPROXIMATE-SPLIT using RANK-FIND

We first divide S into groups of 5 elements. So take $t = \lceil \frac{n}{5} \rceil$. Now we sort each group. Since each group have constant size this can be done in $O(n)$ time. So now consider the scenario:



Chapter 3

Polynomial Multiplication

Chapter 4

Bibliography

[CTL+] Thomas H. Cormen, H. Thomas, Leiserson, E. Charles, Ronald L. Rivest, L. Ronald, Stein, and Clifford. *Introduction to Algorithms*. Brand: PHI Learning.