Implementation of Geometric Algorithms

Instructor - N R ARAVIND

I .Raja sekhar - CS20BTECH11020
Patnala Vikas - CS20BTECH11037

Quick Overview

- Convex Hulls
- K-Dimensional trees
- Min Circle
- PMS (Perfectly Matched Sets)
- LSI (Line Segments Intersection)
- Closest Pair of Points

Convex Hulls

Problem Statement : Given a set of points in the 2D plane, find the smallest convex polygon that contains all the points inside it ?

Naive Approach : Finding all the possible subsets from the given set and for each subset check if the polygon formed covers the entire set, By keeping the track of smallest polygon. Return the smallest polygon after all.

Our Approach : Divide the set of points into two halves by selecting the line joining lowest-left & highest-right points as end points. For each half of convex hull traverse from the one end point to other by maintaining a stack traverse and comparing the side of each newly added point. Add the two halves and return the polygon formed.

Algorithm

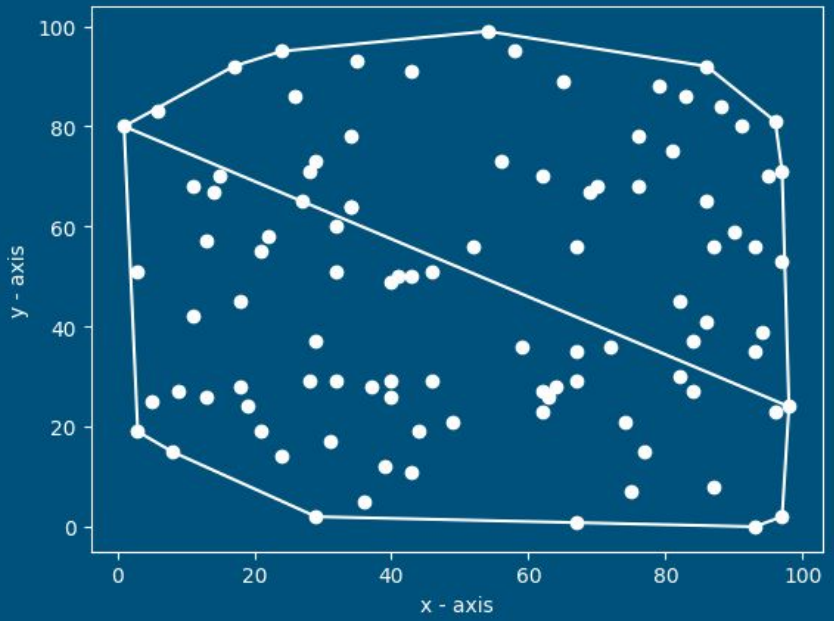
1. Take the input set of points and store it in a vector.
2. Sort them using the compare function so that all the points are sorted on the basis of x-coordinate and if the x-coordinates are same, then sort them based on the y-coordinate.
3. Now mark the left-lowest & right-highest points. Using the line formed by these two points divide the whole set of points to two.
4. Starting from the left-lowest maintaining a stack with left-lowest as initial point move until we get the right-highest point. At each point if the stack-size is less than 2 directly add it into the stack, else check the turning of edge between the last two points in the stack and last point in the stack , new point in the set.
5. Based on this turing add the last point, new point if the turning makes it concave, else continue in the loop to get the upper half of the hull.
6. Similarly run a loop for the lower half of the hull and return them by joining.

Results

Set of randomly generated 100 points is given as input to the algorithm. We get back set of points of the convex-hull. Using these the plot is made for the convex-hull.

Note : The straight line in between the points is the line joining the highest-right & lowest-left points.

Plot for the example



Complexity

Naive Algorithm : O (2^n x n^2) . There will be 2^n number of subsets. For each subset to check whether the polygon is convex or not it takes O (n) time and for checking whether all points lie inside this set, takes O (n^2) time.

Our Algorithm : O (n log n). Sorting of points takes O (n log n) time, further while creating the upper hull & lower hull each point may be added/ deleted once from the stack.Once it is popped from the stack we doesn't come back again to it. So this part of the code takes O (n) time.

Link for the Code : Link

K Dimensional Trees

Problem Statement : Perform the range queries on a large set of points in a 2D space.

Naive Approach : For each query just traverse through all the points in the given set.

Our Approach : Build a KD Tree, for the given set of points so that each point takes the leaf node. Now for each query just traverse through the tree. At each node based on the division line add the points.

Algorithm

1. Store the points in a vector. Take the x-sorted & y-sorted vectors of the input set.
2. Now start building the KD Tree, alternatively shifting the median line parallel to x-axis, y-axis.
3. Using the sorted vectors of set of points at each stage separate them into two branches of the tree. At last you will end up with one point name it as leaf nodes.
4. Using this KD Tree, for any given query starting from the root. If the branch of any node is completely inside the region asked then output all the points on that branch else recursively search them in the branch.
5. To find the branch is completely inside the region or not, use a region defined data-type to store the minimum rectangular region enclosing all the points inside it. Use this to find whether the region of branch is completely inside or has a conflict with the query.

Results

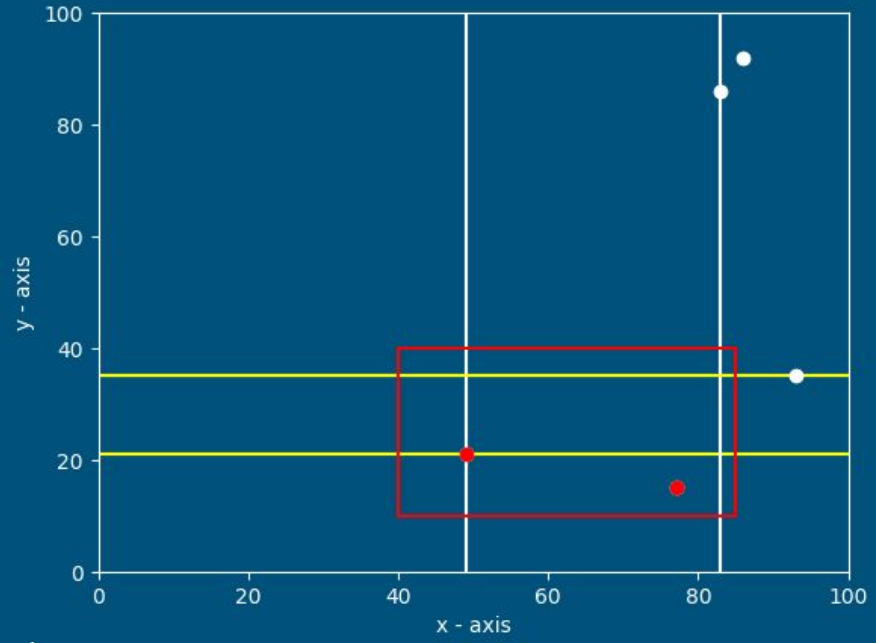
A sample input and output are

Input
5
83, 86
77, 15
93, 35
86, 92
49, 21
reg 40 85 10 40

Output
49, 21
77, 15

- White lines divide points along vertically
- Yellow lines divide points along horizontally
- Red rectangle is our query and red points are result

Plot for the example



Complexity

Naive Algorithm : O (n) . There will be n number of points. For each query check whether each point is inside or not. Hence for ‘m’ queries the time is O (n x m).

Our Algorithm : Time - O (n log n). As initially we are sorting it takes O (n log n) time and for building the tree it takes O (n log n) time. Searching takes O (log n) time. Hence for ‘m’ queries the time is O (nlogn + mlogn).

Link for the code : Link

Minimum Circle Enclosing all Points

Problem Statement : Given some number of 2D points we have to find the minimum circle (circle with the minimum radius) enclosing (call MCE) all the given points?

Some Observations

- MCE can intersect at least one point else we can shrunk until it passes through a point
- Given a point on the circle now the circle can be further shrunk by moving center towards that point.
- If the circle intersects two points
    - if the distance between them is equal to diameter of circle then circle cannot be shrunk anymore
    - else the center can be moved toward the midpoint of the two points

Minimum Circle Enclosing all Points

From the observation we know if three points lie on the circle we can find the equation of the circle and if we know two points which lie on diameter then also we can find eqn.

Naive Approach : We will take every triplet (and every pair of points as diameter) and find the passing circle passing these 3 points and then check every point whether it is in the circle or not.

Our Approach : We used the randomized algorithm (welzl algorithm) which randomly picks one point and checks whether the point is inside the circle formed by the remaining point if it is not inside the circle then consider this point as a point on circle and again find the min circle formed by considering the point on circle

Algorithm

Let the given points be P and the points on the circle are R

algo(P, R):

1. if P = φ and |R| = 3:
2. return trivial_cirlce(R)
3. pick a random point p from P and remove it from P
4. let circle d = algo(P, R)
5. if p is enclosed by d, return d
6. else, p must lie on the boundary of the MEC
7. R.insert(p)
8. return algo(P, R)

trivial_cirlce(R):

- if |R| = 1, return point
- if |R| = 2, return circle formed by R[0], R[1]
- if |R| = 3, return circle formed by R[0], R[1], R[2]

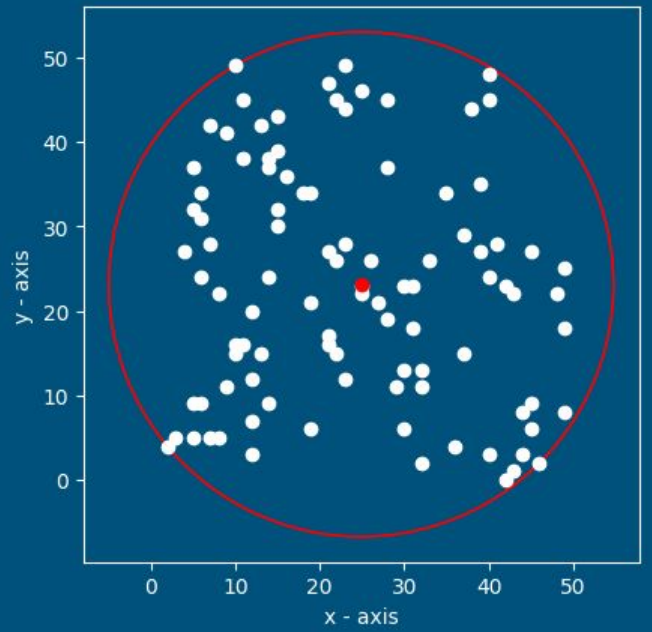
Results

A randomly generated 100 points in the 2D plane with x ϵ [0, 50] and y ϵ [0, 50]

Min circle enclosing all the points and the center of the circle are plotted in red color.

In this example we got 3 points on the circle and it encloses all the points.

Plot for the example



Complexity

Naive Algorithm : O (n^3 x n) . As there will be n^3 number of triplets and for each triplet we have to check all the points are inside or not.

Our Algorithm : Expected time - O (n). With every call, the size of P gets reduced by one. Also, the size of R can remain the same or can be increased by one. Since |R| cannot exceed 3, then the number of different states would be 3n. Therefore, this makes the expected time complexity to be O (n).

Link for the code : Link

Perfectly Matched Sets

Problem Statement : Given two set of points divided by a straight line one set act as a transmitter and other set act as receiver. Now pair transmitter and receiver, a receiver can be paired to the transmitter if the distance between them is less than ‘d’ (some constant), a receiver must not have any interference from multiple transmitters and one transmitter can only transmit to one receiver find the max pairs which satisfies these constraints?

Naive Approach : Check for every possible assignment between transmitter and receiver whose distance is less than ‘d’.

Our Approach : We used a dynamic programming approach which decreases the multiple iterations. Let dp[i][j] be the max matching value till the i^{th} transmitter and j^{th} receiver then the recurrence will be dp[i][j] = max({dp[i-1][j], dp[i][j-1], dp[i-1][j-1] + 1})

Algorithm

1. From the given 2D points we construct an adjacency matrix where we add a edge between i^{th} transmitter and j^{th} receiver if the distance between them is less than d.
2. Let the adj matrix be ‘adj’ now initialize a 2D vector (let it be ‘dp’) with zeros.
3. Traverse through every pair and now use the recurrence
    a. if adj[i][j] == d
        i. dp[i][j] = max({dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1] + 1})
        ii. store this pair in the ‘res’ vector
    b. else
        i. dp[i][j] = max({dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]})
4. Now output the ‘res’ vector

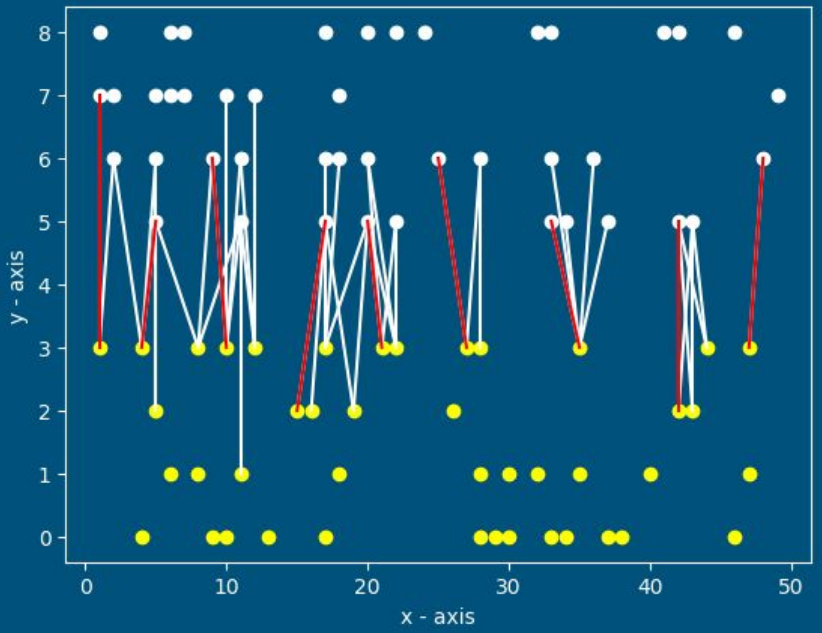
Results

A randomly generated 100 points and 50 of which are transmitters and 50 of which are receivers.

All the white lines in the diagram are transmitter and receivers whose distance is less than 4.

The red lines are the one which are outputted by the program.

Plot for the example



Complexity

Naive Algorithm : In the naive approach we will either consider or not consider a edge based on its distance and we recurse and find the max matching.

Our Algorithm : Time - O (n x m) where n is the number of transmitters and m is number of receivers. For constructing the adjacency matrix we take O (n x m) time and also we traverse n x m pairs and update our dp[i][j] values. Hence the overall time will be order of n^2.

Link for the code : Link

Line Segment Intersection

Problem Statement : Given a set of lines in the 2D plane, find the all intersection points of these lines?

Naive Approach : Checking every pair of lines for the intersection if there is an intersection report it else continue

Our Approach : We used sweepline algorithm to find the intersection points. We image a sweepline sweeping the plane from top to bottom to cover all the points and categorize points into 3 events 1. Starting point 2. Ending point 3. Intersection Point. At every event point we follow below algo and report intersections

Data Structures

- ‘class Point’ for storing a 2D point
- ‘class Segment’ for storing a line segment which contains
  - Two points starting and ending of a segment
- ‘class Event’ which contains
  - Type of event it is (starting, ending, intersection events)
  - Point and vector of Segments
- ‘set <Event *, Eventcompare> EventQ’ which maintains events in the order or their y-coordinate
- ‘set<Segment *, Segmentcompare> SegmentTree’ which maintains segments at a particular point time.
- ‘vector<Point*> res’ for storing the intersection points of lines

Algorithm

1. While there is an event point in EventQ
    a. Pop a event
        i. if the event is starting type
            1. insert segment in Segment Tree
            2. check for intersection with left and right neighbours in Segment Tree
            3. if there is intersection add this intersection event to the EventQ
        ii. if the event is ending type
            1. delete segment from Segment Tree
            2. check for intersection of newly became adjacent points
            3. if there is intersection add this intersection event to the EventQ
        iii. if the event is intersection
            1. swap the lines involving in intersection in the Segment Tree
            2. check for intersection with new neighbours in Segment Tree

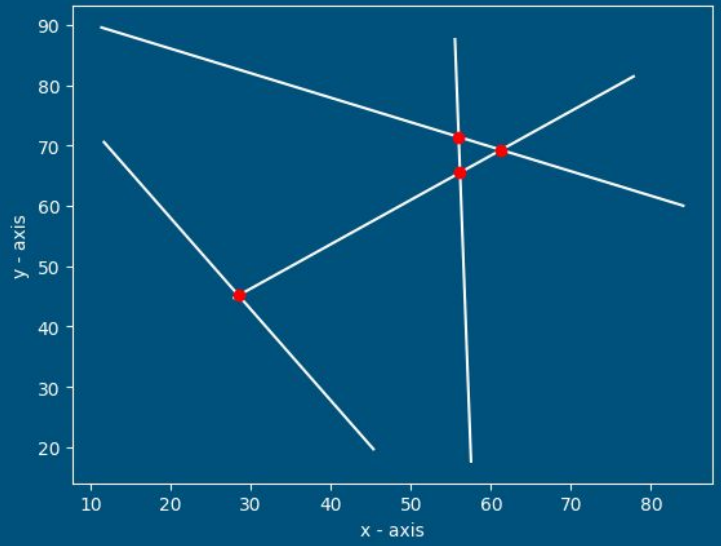
Results

A sample input and output are

Input
4
45.3258 19.6791 11.6638 70.5918
55.5085 87.6251 57.515 17.6294
11.3535 89.5904 84.0144 60.0459
77.8131 81.4742 27.9174 44.7606

Output
4
55.9722, 71.4481
61.2632, 69.2967
56.1419, 65.5284
28.4726, 45.1691

Plot for the example



Link to more example plots

Complexity

Naive Algorithm : O (n^2) . As there are n lines. Number of pairs of lines form them will be in the order of n^2.

Our Algorithm : As we process every event point once and every intersection point is a event point let number of intersections are k, total number of lines are n then we have 2*n + k event points at every event point we either perform insertion or deletion to segment tree which costs O (logn) hence the time will be O (nlogn + klogn).

Link for the code : Link

Closest Pair of Points

Problem Statement : Given a set of Points in the 2D plane, find the closest pair of points among the given?

Naive Approach : Check every pair of points and keep track of the closest pair. Another approach in which we use divide and conquer strategy.

Our Approach : We used a linear time randomized algorithm (rabin's algorithm). We randomly pick √n points and find the closest pair among them(d) and then divide the 2D plane into squares of dimension (dxd) and find the closest pair among them.

Algorithm

1. Randomly select √n points from given points.
2. Find the closest pair of points among them and let the distance between them is “d”.
3. Now partition the plane into squares of dimension dxd.
4. Place every given point(initial n points) in the square whose left corner is (floor(x/d), floor(y/d)).
5. Now find the closest pair in each cell and cells which are neighbours to it i.e top, bottom, left and right of the current cell.
6. Finally we will get the closest pair among all the pairs.

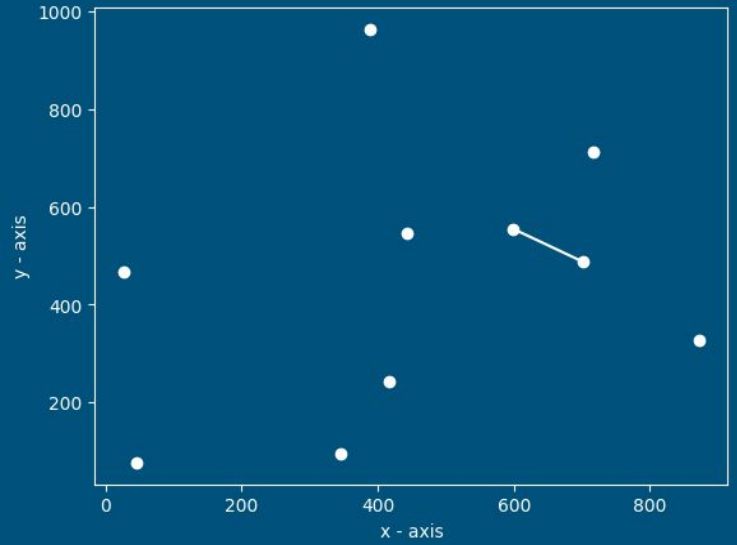
Results

A sample input and output are

Input
10
27, 468
416, 242
346, 96
443, 545
46, 77
716, 713
872, 328
599, 555
701, 488
389, 963

Output
701, 599
488, 555

Plot for the example



Complexity

Naive Algorithm : As naive approach will check every pair for distance time complexity will be $O(n^2)$

Our Algorithm :

- Step 2 of algorithm takes O (n) time as only $\sqrt{n}$ points are checked.
- From the theoretical analysis expected time for the Step 4 of the algorithm is O (n)
- Hence the expected time for the algorithm is O (n)

Link for the code : Link

Thank You