Ahmet Faruk Ulutaş - 21803717 – CS478 – HW2

1-

First of all, let's take the locations where M[i][j] is true, that is, where the points are located, into an L list in order not to constantly wander the matrix. Then, let's use the divide and conquer algorithm to create a convex hull from the point list. It can be reduced to O(nlog(logn)) time using the DeWall approach in the Divide and Conquer algorithm. In the Dewall approach, the splitter line is rotated 90 degrees at each step, while the split operation in the Divide and conquer algorithm is performed. Thus, a faster and more efficient algorithm is obtained.

2-

It is desired to add and remove points while preserving the convex feature of the shape from the S set, where the points are located convexly. Let's call the set of convex points in the set S as Sc.

Adding Points

Draw a line from point P to each point that will be in the ith row.

If the points before and after i are on the same side of this line, these points are called supporting vertices. Check the supporting vertices. If there is no support point, it is inside the convex hull. Therefore, it is independent of Sc.

If there is a support point, there are 2 supporting vertices. Subtract all points from the right support vertex to the left support vertex from Sc and add the original point to Sc.

This algorithm works in O(n) complexity as each point in the list needs to be checked individually.

Point Removal

If the removed point is located on Sc, subtract the point from Sc. If the dot is not in Sc, do nothing.

This algorithm has O(1) complexity as it is one step and O(n) space complexity as it stores n points in memory.

3-

It is the CheckCollision(allObjects) function that creates O(N^2) complexity in the algorithm. With the development of this function, lower complexity can be achieved. In order to increase the efficiency of the algorithm, range search trees should be used. All objects are inserted into the tree. Then the leaves of the tree are checked. If there is more than one element in one of the leaves, there is a collision situation. This results in O(nlogn) complexity.

4-

Two attributes must be completed to produce a monotone complete set of chains:

Property 1: the union of members of S contains G.

Property 2: For any two chains C i and C j of S, the vertices of C i which are not vertices of C j lie on the same side of Cj.

Also, balanced tree and top-down sweepline algorithm should be applied to WEIGHT-BALANCING IN REGULAR PSLG to achieve the desired.

5-

Explanation

To ensure O(nlogn) complexity, it is sorted from largest to smallest by looking at the x values using quick sort or merge sort. Then the y-coordinate for each point in the list and the maximum y-coordinate before it is compared. If it is greater than maxy, maxy cannot dominate it and the point where the y value is located is added to the maxset.

Algorithm

1. Get the initial points list L.

2. Initialize an empty maximal set list and max y values (maxY = -INF, maximalSet = []).

3. Sort the list L in descending order of x coordinate.

4. Check the y values for each element in the list and add it to the maximal set list if it's greater than the maximum y value so far.

For each element in the list

If itemy is greater than maxy

maxy is equal itemy

add element to maximalSet

5. Reverse maximalSet list

6. return maximalSet list

Analysis

It has O(nlogn) complexity because merge or quicksort is used while sorting. Traversing the list has O(n) complexity. Reversing the list has O(n) complexity. As a result, the algorithm has the complexity O(nlogn) + O(n) + O(n) = O(nlogn).

Proof of Algorithm

The algorithm can be proved by contradiction. The list of maximal points given with x and y coordinates [(x1, y1), (x2, y2), ..., (xk, yk)] is sorted according to X'values (x1<x2<...<xk).

If the x and y values of one of the two selected points are greater than the other, it dominates the other point (yj<yi and xj<xi thus, j<i).

As a result, the other point is not the maximal point.