CS345 Theoretical Assignment 1

Ayush Agarwal, 13180

M.Arunothia, 13378

Contents

1 Non-Dominated Points

1.1 Overview

Given a set of coordinates P, we create list of each layer in the following manner. First sort the coordinates based on Y-coordinate in descending order. Then maintain an array A of size n. Start with the first coordinate from the sorted array P(of all coordinates). This point will be a non-dominated point and will be a part of layer 1. Update the first index of A with the x-coordinate of this point. Now take the second point from P. If its x coordinate is greater than the x-coordinate of earlier point, it means that it will be part of layer 1. If so then add it to layer 1 and update the layer 1's index in A. Otherwise it will be in second layer, so add it in layer 2 and update the layer 2's index in A with it's x coordinate. Repeat the above procedure for all points.

1.2 Pseudo-Code

Non-Dominated-points(P)
{
    $P \longrightarrow reverse\_sort(P)$ //sort in descending order of Y
    $Layer[n]; A[n]$
    $A[0] = P[0].x$
    $Layer[1].push()$
    $i = 1; right = 1$
    $while(i < P.length())$
        $point = P[i]$
        $index = binary\_search\_predecessor(A, 0, right, point)$
            // returns the predecessor's index
        $Layer[index].push(point)$
        $A[index] = point.x$
        $If(index > right) right++$
    $returnLayer$

}

1.3 Time Complexity

Sorting step takes $O(nlogn)$ time, followed by binary_search for each point which takes $logn$ time per point. While iterates for all the points and in each iteration binary_search is invoked, thus the loop takes $n * logn$ time. Overall algorithm takes time
$O(nlogn) + O(nlogn) = O(nlogn)$

2 Open Rectangle Query

2.1 Data Structure Design :

Given an array 'a' of 'n' coordinate points, we construct a Binary Search Tree (BST) call it 'data' in the following manner.

- Sort the array 'a' w.r.t the x-coordinates of the points. Call this sorted array 'b'.
- Divide 'b' into $\frac{n}{Log[n]}$ parts, starting from the beginning. Index each of the part incrementally from 1 to $\frac{n}{Log[n]}$.
- Construct BST 'data' with $\frac{n}{Log[n]} = N$ nodes from 'b' using the above indexing for the comparisons.
- Now, we have a BST 'data' with 'N' nodes augmented with an array of $Log[n]$ size at every node. Sort this array at every node on basis of y-coordinates of the points.
- This completes the description of augmented BST 'data'.

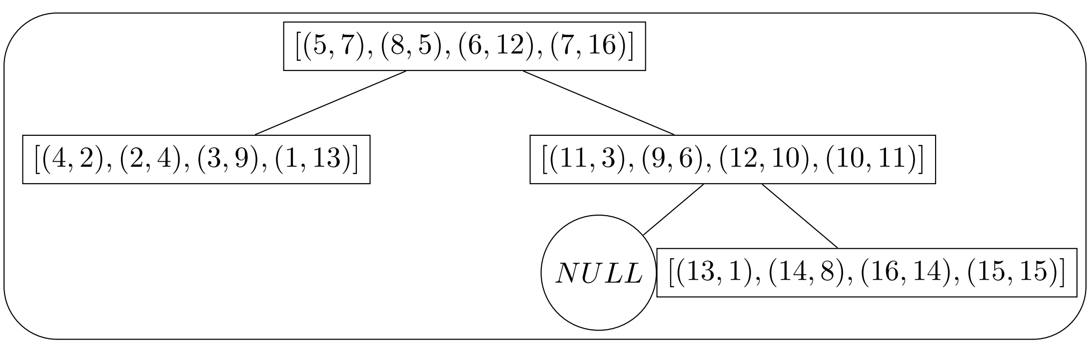
Given Array 'a'

(13,1)	(4,2)	(11,3)	(2,4)	(8,5)	(9,6)	(5,7)	(14,8)	(3,9)	(12,10)	(10,11)	(6,12)	(1,13)	(16,14)	(15,15)	(7,16)

Sorted Array 'b' based on x coordinates

(1,13)	(2,4)	(3,9)	(4,2)	(5,7)	(6,12)	(7,16)	(8,5)	(9,6)	(10,11)	(11,3)	(12,10)	(13,1)	(14,8)	(15,15)	(16,14)

BST 'data' constructed for this example



2.2 Algorithm :

STEP 1 : Start

STEP 2 : If($x_2 - x_1 < 2 * (Log[n])$, traverse elements in this range of x and return the points satisfying $y > y_{bottom}$.
else Initialise variables node_i to the x value of nearest node ahead of x_1 and node_j to the x value of nearest node behind x_2.

STEP 3 : Find the elements satisfying $y > y_{bottom}$ in the x range x_1 to node_i and in x range node_j to x_2, and report them.

STEP 4 : Find the elements satisfying $y > y_{bottom}$ in the x range node_i to node_j and report them.

STEP 5 : Stop.

2.3 Pseudo Code :

Report_points(x_1, x_2, y_bottom)
{
if($x_2 - x_1 < 2 * (Log[n])$)
Locate the required x range in the BST.
Report elements between that x range satisfying $y > y_{bottom}$ using binary search

else if(x_1 and x_2 exists in data points)
Locate the required x range in the BST.
Report elements between that x range satisfying $y > y_{bottom}$ using binary search

else
node_i $\longrightarrow$ x value of the nearest node ahead of x_1
node_j $\longrightarrow$ x value of the nearest node before x_2
report_i = Report_points(x_1, node_i, y_{bottom})
report_j = Report_points(node_j, x_2, y_{bottom})
report_rest = Report_points(nodeı, nodeȷ, y_{bottom})
}

2.4 Space Complexity :

The data structure we invented, is a BST of size N*(augmentation size). Therefore, space used is $N * Log[n]$= n. (Refer Sub section Data Structure Design). Implying space complexity is O(n).

2.5 Time Complexity :

2.5.1 Query Time :

2.5.2 Pre-processing Time :

- The first sort based on x coordinates requires O(n*Log[n]).
-
-
-