Romil V. Shah

20008692

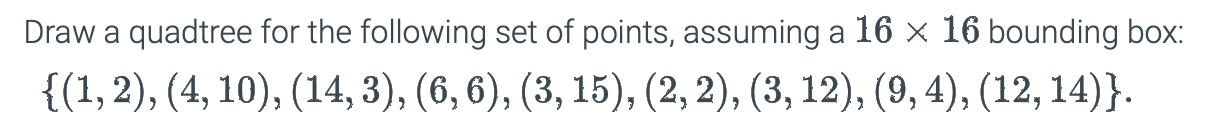
CS / CPE 600

Prof. Reza Peyrovian

Homework Assignment 11

Submission Date: 12 / 04 / 2022

Q1. No. 21.5.7



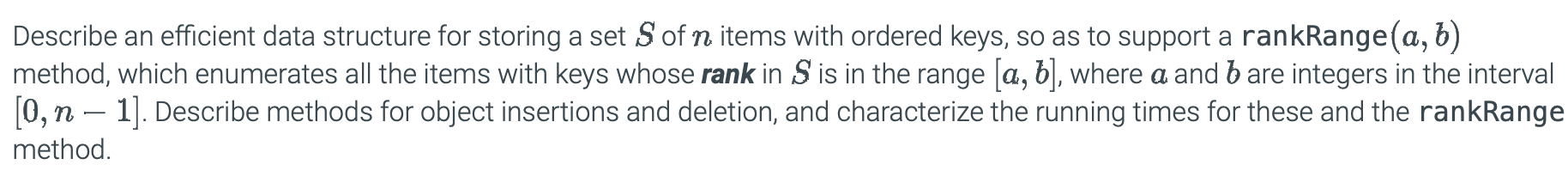
Sol.

Assuming all the points that lie on the intersection will come in first quadrant whether NE (North East), NW (North West), SE (South East), SW (South West),

A picture containing graphical user interface

Description automatically generated

Q2. No. 21.5.13



Sol.

An efficient data structure for storing a set S of n items with ordered keys can be a balanced binary tree(AVL or Red-Black Trees). The rankRange(a, b) method will work by searching for the lower end of a range x1 and upper end of a range x2 where x1 and x2 satisfies (x1 ≤ x ≤ x2 ) and counting all the elements in the search tree that exists between x1 and x2 using in-order ordering.

Insertion in AVL tree:

Algorithm insertAVL(k, e, T ):

Input: A key-element pair, (k, e), and an AVL tree, T

Output: An update of T to now contain the item (k, e)

v ← IterativeTreeSearch(k,T)

if v is not an external node then

return “An item with key k is already in T ”

Expand v into an internal node with two external-node children

v.key ← k  
v.element ← e  
v.height ← 1  
rebalanceAVL(v, T )

Deletion in AVL tree:

Algorithm removeAVL(k, T ):

Input: A key, k, and an AVL tree, T  
Output: An update of T to now have an item (k, e) removed

v ← IterativeTreeSearch(k,T)

if v is an external node then

return “There is no item with key k in T ”

if v has no external-node child then

Let u be the node in T with key nearest to k

Move u’s key-value pair to v  
v←u

Let w be v’s smallest-height child

Remove w and v from T , replacing v with w’s sibling, z

rebalanceAVL(z, T )

Rebalance Tree:

Algorithm rebalanceAVL(v, T ):

Input: A node, v, where an imbalance may have occurred in an AVL tree, T

Output: An update of T to now be balanced

v.height ← 1 + max{v.leftChild().height, v.rightChild().height}

while v is not the root of T do

v ← v.parent()  
if |v.leftChild().height − v.rightChild().height| > 1 then

Let y be the tallest child of v and let x be the tallest child of y

v ← restructure(x) // trinode restructure operation

v.height ← 1 + max{v.leftChild().height, v.rightChild().height}

The rankRange method will take O(log n + k) time where k is the number of points reported in a range and both the deletion and insertion method will take O(log n) time.

Q3. No. 21.5.27

Text, letter

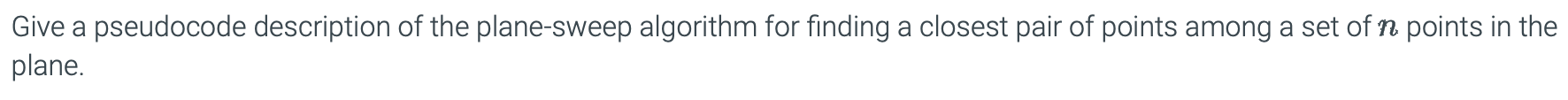
Description automatically generated

Sol.

We can use Range Trees data structure for querying two-dimensional data. Instead of an auxiliary tree, we can store an array, sorted by Y-coordinates. At xsplit we will do binary search for y1 As, we continue to search for x1 and x2, we can use pointers to keep track of the result of binary search for y1 in each of the arrays along the path. This method is also known as fractional cascading search.

The run time of the algorithm for d dimension is O(logd-1 n + s) and for 4 dimension it will be O(log3 n + s).

Q4. No. 22.6.7



Sol.

Let S be a set of n points.

For finding the minimum distance between two points P and Q it can be calculated as

dist(a, b) =

d = dist(a, b)

For any point p in S, x(p) and y(p) denote the x and y coordinates.

Consider a sweep line SL is the vertical line through point p of S.

Algorithm closestPair()

Input: Set S of n points in the plane

Output: Finding Closest pair (p, q) of points

Let X be the structure in an array A[1, … ,n] containing set S points sorted by x-coordinates.

δ := dist(A[1], A[2]) (minimum distance among all points to the left of SL)

Let Y be the empty dictionary.

while (point p <= n)

p = p+1

when new point is found

if (dist (p, q) < δ)

then

A[1] ← p,

A[2] ← q

d ← dist(p, q)

Insert p into dictionary Y

Search closest point q to the left of p to points in Y.

for( all points whose y coordinates lie in [y(p) - , y(p) +  ] )

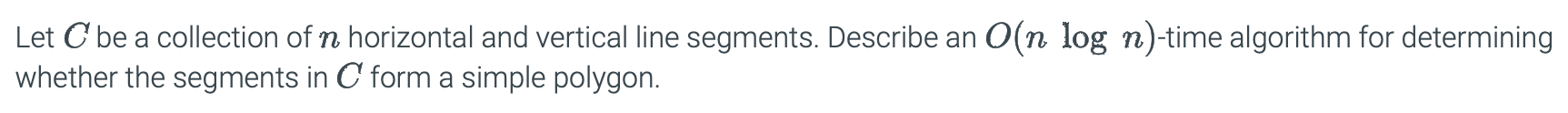
find q point closest to p

return (p,q)

Sorting of elements will take O(n log n)time. Insertion and deletion for an element will be done once in the dictionary takes O(log n) time and each range query in S takes O(log n).

So, the total running time of the algorithm is O(n log n).

Q5. No. 22.6.16



Sol.

It is given that in a collection C of n horizontal and vertical line segments to form a simple polygon we can use.

1. Using plane sweep, determine all pairs of intersecting segments with common coordinates.
2. As seep line SL move from left to right find the coordinates which have common horizontal and vertical points in the plane.
3. If a common coordinate is found then store it in dictionary, whenever we find another line segment check in the dictionary if they have common endpoints.
4. In this form we can return all the coordinated in the dictionary and check if they form the closed loop. It means there exists a polygon.

Number of points in collection C is n. Plane Sweep algorithm will take O(n log n) times. Moving for coordinates for the line segments will cover each line will take O(n) time.

So, algorithm will take O(n log n) time in total.

Q6. No. 22.6.30

Text

Description automatically generated

Sol.

* To determine a line L which divides the blue and red points into two separate sets we can use convex hull property.
* Forming a convex hull around blue and red points separately.
* Then checking if both the convex hull intersects each other not.
* If intersection between convex hull exists, then there is a line which divides red and blue points separately.

Creating a convex hull using Graham Scan Algorithm will takes O(n log n) time.