CS 600 Advanced Algorithms

Sri Dhanush Reddy Kondapalli

CWID: 10476150

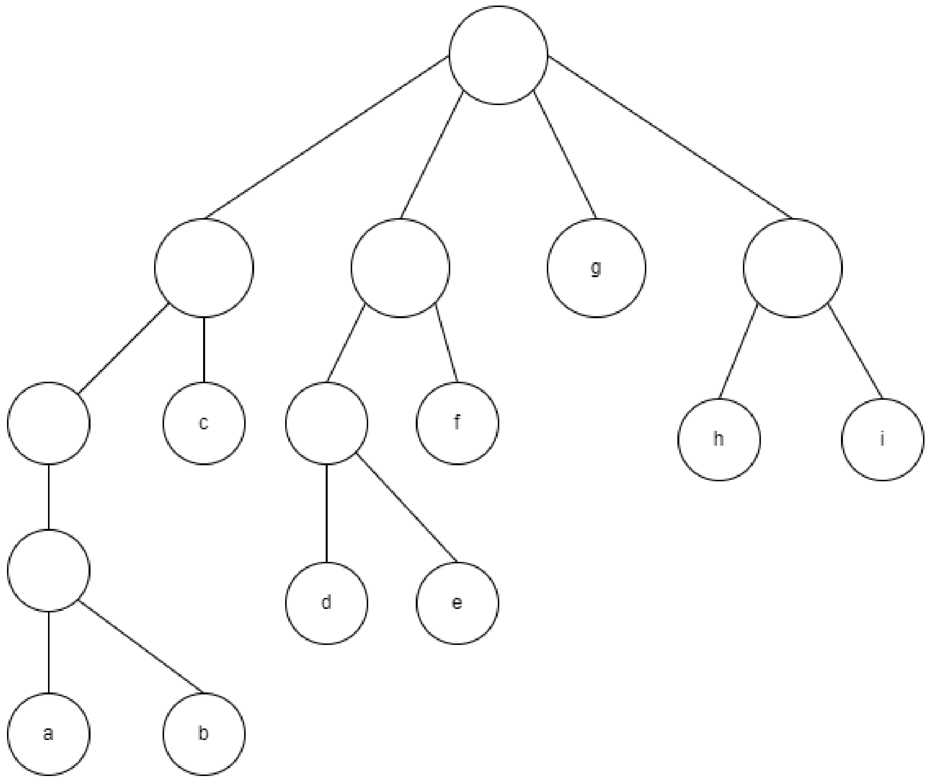
Homework 11

1 R-21.5.7 Draw a quadtree for the following set of points, assuming a 16 x 16 bounding box

The bounding box looks like

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		a														
2		b														
3												d			f	
4										e						
5																
6						c										
7																
8																
9				h												
10																
11																
12														g		
13																
14			i													
15																
16																

And the quadtree looks like



2 C-21.5.13 Describe methods for object insertions and deletion, and characterize the running times for these and the rankRange method.

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 ($x1 \leq x \leq x2$) 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}

Running Time: For the rankRange approach, the time required is $O(\log n+k)$, where k is the number of points reported in a range. Furthermore, both the deletion and insertion methods will take $O(\log n)$

3 A-21.5.27 Explain how you can use D to answer two-dimensional range queries for the rectangles in S, given a query rectangle, R, would return every bounding box, R_i, in S, such that R_i is completely contained inside R.

We may query two-dimensional data using the Range Trees data structure. We can store an array ordered by Y-coordinates instead of an auxiliary tree. We will do a binary search for y_1 at x_{split}. We can use pointers to maintain track of the result of the binary search for y_1 in each of the arrays along the path while we continue to look for x_1 and x_2. This technique is often referred to as fractional cascading search.

Running Time: The method runs in $O(\log^{d-1} n + s)$ for d dimensions and $O(\log^3 n + s)$ for 4 dimensions.

4 R-22.6.7 Give a pseudocode description of the plane-sweep algorithm for finding a closest pair of points among a set of n points in the plane.

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.

Input: Set S of n points in the plane
Output: Finding Closest pair (p, q) of points
Algorithm closestPair():
    Let X be the structure in an array A[1,...,n] containing set S points sorted by x−coordinates.
    $\delta :=$ dist(A[1], A[2]) (minimum distance among all points to the left of SL)
    Let Y be the empty dictionary.
    while(point p $\leq$ n)
        p $\leftarrow$ p+1
        when new point is found
        if (dist (p, q) $<$ δ)
        then
        A[1] $\leftarrow$ p,
        A[2] $\leftarrow$ q
        d $\leftarrow$ 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) − δ, y(p) + δ ] )
    find q point closest to p
return (p,q)
```

Running Time: Element sorting will require O(n log n) time. Insertion and deletion of an element in the dictionary will require O(log n) time. And in S, each range query takes O(log n). As a result, the algorithm's total running time is $O(n \log n)$.

5 C-22.6.16 Let C be a collection of n horizontal and vertical line segments. Describe an O(n log n)-time algorithm for determining whether the segments in C form a simple polygon.

We may utilize a collection C of n horizontal and vertical line segments to build a basic polygon.

1. Using plane sweep, locate all pairs of crossing segments with same coordinates.
2. Find the locations that share similar horizontal and vertical points in the plane when the seep line SL moves from left to right.
3. Whether a common coordinate is discovered, it is saved in the dictionary, and everytime we locate another line segment, we check the dictionary to see if they have common endpoints.
4. We can retrieve all the coordinated in the dictionary in this manner and see if they form a closed loop. It denotes the existence of a polygon.

Running Time: The total number of points in collection C is n. The Plane Sweep algorithm will use O(n log n). Moving for coordinates for the line segments will take O(n). As a result, the method runs in $O(n \log n)$ time.

6 A-22.6.30 Describe an efficient algorithm for determining whether there is a line, L, that separates the red and blue points in S. What is the running time of your algorithm?

We may utilize the convex hull property to find a line L that separates the blue and red points into two distinct sets. Separately forming a convex hull around blue and red spots. Then determine whether or not both convex hulls intersect.

If the convex hulls cross, there is a line that splits the red and blue points independently.

Running Time: It will take O(n log n) to create a convex hull using the Graham Scan Algorithm.