CS 600 Homework 11 Solutions

R-21.5 Worst case depth of a k-d tree defined on n points in the plane

Recall that the k-d tree is similar to a quadtree data structure, but is binary. Each node v in a k-d tree is associated with a rectangular region R. Each time a split operation is performed for a node v in a k-d tree, it is done a line perpendicular to a coordinate axis. Dimensionality doesn't affect this operation.

The depth, therefore, is $O(\log n)$ since after every split, we need to split an additional $n/2$ number of points. For higher dimensions, the depth is again $O(\log n)$ for the same reason.

C-21.5 Static data structure to store a two-dimensional set S of n points

For this problem, we need to design a static data structure – one that doesn't support insertions and deletions, which stores a two-dimensional set S of n points. Further, the data structure should be able to answer the countAllInRange(a, b, c, d) operation in $O(\log^2 n)$ time, where countAllInRange(a, b, c, d) returns the number of points in S with x coordinates in the range $[a, b]$ and y coordinates in the range $[c, d]$.

To do this, we can use a range tree. In additional to storing the usual information, we also store the number of items stored in the subtree rooted in each auxiliary tree. From Theorem 21.2, constructing this data structure would take $O(n \log n)$ time, and the space taken by this data structure is $O(n \log n)$. To answer the countAllInRange(a, b, c, d) query, we simply run a modified version of findAllInRange(a, b, c, d) where instead of returning the elements, we simply return the count we stored. Unlike findAllInRange which takes $O(\log^2 n + s)$ time (where s is the number of elements returned), this only requires $O(\log^2 n)$ time.

A-21.3 Constructing a Priority Search Tree in $O(n)$ time

We are given a set of n two-dimensional points such that each coordinate is in the range $[0, 4n]$ and are required to construct a PST for this set in $O(n)$ time.

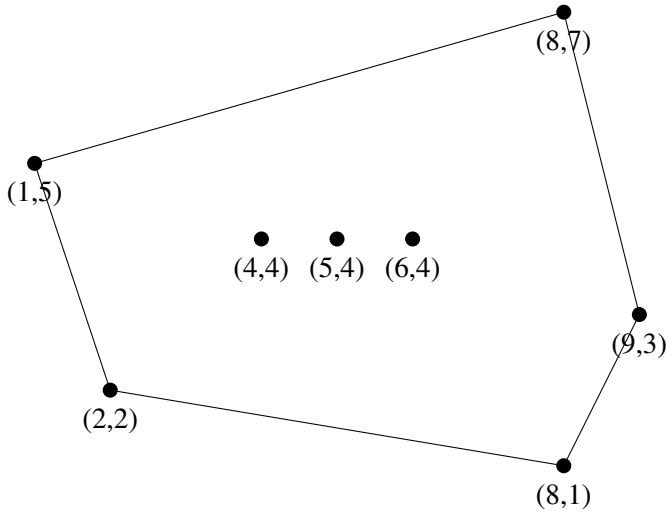
Recall that to build a PST, we first begin by sorting the set using the x coordinate (in increasing order), and then call the buildPST method. This sort itself takes $O(n \log n)$ time. In order to reduce the time complexity, we need to sort the n points by the x coordinates in $O(n)$ time. We can use bucket sort to achieve this since we know the coordinates are in the range $[0, 4n]$.

We then perform a recursive algorithm similar to heapify. This algorithm starts at the root, and recursively constructs a PST for the left and right child. Then it selects the child storing a point with the larger y coordinate, places that point at the root, and repeats this down-heap bubbling action. The time required for this phase is $O(n)$.

R-22.8 Convex Hull for a set of points

We have a set of points $\{(2,2),(4,4),(6,4),(8,1),(8,7),(9,3),(1,5),(5,4)\}$ and are required to draw the convex hull.

The convex hull of this set of points is given by the set $\{(9,3),(8,7),(1,5),(2,2),(8,1)\}$. Here is a diagram.



C-22.5 $O(n)$ time algorithm to test whether an n vertex polygon in convex

Recall that a polygon is convex if it is simple and all its internal angles are less than π.

To determine whether a given n vertex polygon, P, is convex, we start by traversing the vertices of P in a counterclockwise order, making sure each one requires a left turn. However this is not enough, since P could contain self-intersections. Therefore, we then choose a vertex v on P and determine, for every other vertex w that the line $v \to w$ is locally interior to P with respect to the edges incident on v and w respectively.

A-22.2 Gerrymandering

In this problem, we are given a voting district defined by an n vertex simple polygon, P. We are required to construct an $O(n)$ time algorithm for testing whether a point q is inside or outside of P. Further, we are allowed to assume that q is not on the boundary of P and that there is not vertex of P with the same x coordinate as q.

To achieve this, consider a ray, r emanating vertically up from q. We now scan through the edges of P and count the number of edges that intersect with r. If this number is odd, then q is inside P. If it is even then q is outside P.