Theory of Algorithms. Homework 3

Peter Racioppo

Computational Geometry – Algorithms and Applications

Ch.5: Qs. 5.9, 5.11, 5.13 a&b and Ch 2: Qs 2.11, 2.12

5.9.) One can determine whether a particular point (a,b) is in a given set by performing a range query with range [a:a]×[b:b].

a.) Prove that performing such a range query on a kd-tree takes time $O(\log n)$.

A point in a binary search tree can be queried in $O(\log n)$ time since at each of the $O(\log n)$ levels we only need to make a constant time decision about whether to go down the right or left branch. The cost saving comes from the fact that the query rectangle is a point and can only intersect the region in which (a,b) lies, so we never have to check the gray subtrees.

b.) What is the time bound for such a query on a range tree? Prove your answer.

Query the primary structure for the desired x-coordinate in $O(\log n)$ time, then, if the x-coordinate is in the set, query the associated structure corresponding to this node for the y-coordinate in $O(\log n)$ time. Altogether, it's $O(\log n)$.

This also follows from the relevant theorem: Querying a range tree for the points that lie in a rectangular region in a set of n points in the plane takes $O(\log n + k)$ time using fractional cascading. In this case, $k = 1$, so the query takes $O(\log n)$ time.

5.11.) Let S_1 be a set of n disjoint horizontal line segments and let S_2 be a set of m disjoint vertical line segments. Give a plane-sweep algorithm that counts in $O((n+m)\log(n+m))$ time how many intersections there are in $S_1 \cup S_2$.

Record the y-coordinates of the endpoints of the vertical segments and the y-coordinate of the horizontal segments in $O(n+m)$ time. Sort them into a balanced binary tree on the y-coordinates in $O((n+m)\log(n+m))$ time. Sweep from the top of the plane to the bottom and add an event every time you encounter a horizontal segment or an upper or lower endpoint of a vertical segment. The initial sorting tells us which type of endpoint we encounter. We'll also have another balanced binary search tree that stores the x-coordinates of all of the vertical segments that we're currently considering. Vertical segments are added to the tree when the sweep line reaches their upper endpoint and removed when it reaches their lower endpoint, in a total of $O((m)\log(m))$ time for all of the endpoints. If the sweep line encounters a horizontal segment, search the binary search tree for the vertical segments and record the number of them with x-coordinates between the endpoints of the horizontal segment. This takes $O(\log(m))$ time for each of the n horizontal segments. Altogether, the algorithm is $O((n+m)\log(n+m))$.

5.13.) Range searching among objects other than points:

a.) Let S be a set of n axis-parallel rectangles in the plane. Describe a data structure that reports all rectangles in S that are completely contained in a query rectangle [x:x']×[y:y'] and that uses $O(n\log 3n)$ storage and has $O(\log 4n+k)$ query time, where k is the number of reported answers.

Given any three vertices of a rectangle, we can compute the x and y coordinates of the four sides. Store one of the vertices in each rectangle, say the bottom left vertex with coordinates (x_A,y_A), in the primary structure of a range tree according to its x-coordinate. Make a secondary structure on the y-coordinate of the bottom left vertices, a third level structure on the length L of the rectangle containing that vertex, and a fourth level structure on its height h. This is a 4D range tree so it uses $O(n\log^3 n)$ storage and has $O(\log^4 n+k)$query time. To determine if a given rectangle is inside the query rectangle, check whether the bottom left vertex is inside by going through the first two structures, then check if the bottom right vertex is inside by going to the third level and checking whether $x_A+L < x'$, then check whether the top left vertex is inside by going to the fourth level and checking whether $y_A+h < y'$.

b.) Let *P* consist of a set of *n* polygons in the plane. Again describe a data structure that uses $O(n\log^3 n)$ storage and has $O(\log^4 n+k)$ query time to report all polygons completely contained in the query rectangle.

Determine the points on the polygon with the lowest and highest coordinates in both the x and y directions, and use them to form a rectangle that bounds the polygon. Finding maxima and minima should take linear time, and forming the rectangle only requires finding four intersections, so it should take constant time. Repeat the procedure in *a*).

2.11.) Let *S* be a set of *n* circles in the plane. Describe a plane sweep algorithm to compute all intersection points between the circles. (Because we deal with circles, not discs, two circles do not intersect if one lies entirely inside the other.) Your algorithm should run in $O((n+k)\log n)$ time, where *k* is the number of intersection points.

Two circles intersect twice if and only if the distance between their centers is less than the sum of their radii and once if these two value are equal. I will assume that no three circles intersect at the same point. Sweep a line from top to bottom in the plane and add a circle to a balanced binary search tree on the x-coordinate of its center as soon as we intersect its topmost point, also recording its radius (the x-coordinate of the center and the first point touched by the sweep line are always the same). (Before starting the sweep line, sort the circles on their x-coordinates.) Check whether adjacent circles intersect and how many times. If they intersect once, record the intersection point and leave the ordering of the circles unchanged. If twice, record the intersection with the greater y-coordinate, switch the ordering of the circles, and check adjacent circles again. When the bottommost point of a circle is reached, remove it. There are $2n$ events for the upper and lowermost points of the circles and k intersections, where k is no more than $2\binom{n}{2}$. Initializing the upper and lower points takes $O(n\log n)$ time and each of the $O(n+k)$ events takes $O(\log n)$ time. Altogether, the algorithm takes $O((n+k)\log n)$ time.

2.12.) Let *S* be a set of *n* triangles in the plane. The boundaries of the triangles are disjoint, but it is possible that a triangle lies entirely inside another triangle. Let *P* be a set of *n* points in the plane. Give an $O(n\log n)$ algorithm that reports each point in *P* lying outside all triangles.

Sort the vertices of the triangles on their y-coordinates. Move a horizontal sweep line from top to bottom, adding the triangles to a balanced binary search tree on the x-coordinate of the first vertices we reach. Since the triangle boundaries are disjoint, this tells us everything about the ordering of the triangles. We only ever have to consider two of the three boundary segments. If we encounter the second vertex of a given triangle, the only necessary update is to change the calculation of the x-direction extremum for that triangle to take account of the fact that we're now on a different border segment. When we reach the third vertex of a triangle, remove the triangle from the BST. If the sweep line encounters a point in P, we query the BST for the triangles with first vertices to the left and right of this point. We then calculate the x-direction extrema at this height for both of the triangles, to see if the point is inside either triangle. If not, the point is not inside any triangle on or above the sweep line, since the triangles cannot intersect. It's also not inside a triangle below the sweep line, because any such triangle has an upper extremum below the point. There are $3n$ events for the vertices of the triangles and n events for the points and each event takes $O(\log n)$ time, so the algorithm is $O(n\log n)$.