Massachusetts Institute of Technology
6.854J/18.415J: Advanced Algorithms
David Karger

Handout 24
Monday, November 28, 2005

Problem Set 12

Due: Wednesday, November 30, 2005 and Monday, December 5 2005.

Problem 1. Due Wednesday, November 30. On a separate page, turn in a brief (i.e. half a page) description of your planned project. If you have formed a group, turn in a single submission for the group, listing all members. List references you have found.

NONCOLLABORATIVE Problem 2. A problem last week found lines (and polygons) *contained* in a rectangle; here we consider finding lines *crossing* a rectangle. As a starting point, suppose you are given an *interval tree* data structure. This takes n possibly-overlapping intervals on the real line, and builds a size-n data structure that can, in $O(k + \log n)$ time, output the set of all intervals intersecting with a given query interval (you may optionally design this data structure if you wish). Given such a data structure, show that you can build a size $O(n \log n)$ data structure for the following problem: given n vertical and horizontal segments in the plane, and given a query rectangle, output all the segments that intersect that query rectangle in $O(k + \log^2 n)$ time.

Problem 3. Suppose you're implementing a video game in which the player can walk around a planar environment made up of walls, and at any time the screen displays only the walls that are (partially) visible by the player. More precisely, the player is modeled as a single point; the walls are modeled as noncrossing line segments; two points are *visible* if the line segment connecting them does not intersect any walls except at its endpoints; and a wall is *visible* from a point if at least one point on the wall is visible from the point. Give an $O(n \lg n)$-time algorithm to compute the set of walls visible from the player. **Hint**: Use a line-sweep algorithm, but instead of sweeping a horizontal line, sweep a half-line around a point.

Problem 4. Consider the problem of finding the smallest (minimum diameter) circle containing some set H of n points in the plane. We will assume that the points are in "general position"—no 3 points are colinear, and no 4 points are on the boundary of a common circle. This assumption can be achieved by small perturbations in the input. For any set of points S in the plane, let $O(S)$ denote the smallest circle containing S.

1. Show that for any 3 non-colinear points, there is a unique circle having all 3 of those points on the circle boundary. This circle (center and radius) can be computed in constant time from the points.

2. Show that $O(H)$ contains either 2 or 3 of the input points on its boundary. We will call these points the "basis" of the circle (hint, hint) and refer to them as $B(H)$. Deduce a simple $O(n^4)$-time algorithm for solving the problem.
3. Show that if a circle C excludes a point of H, then C cannot be the smallest circle containing $B(H)$.
4. Show that if p is *not* contained in $O(S)$ for some S then p is on the boundary of $O(S \cup \{p\})$.
5. Generalize the above to finding the smallest circumcircle of H that is required to pass through a specific set of (one or two) points (assuming it exists).
6. Give an $\tilde{O}(n)$ expected time randomized incremental algorithm for finding $O(H)$.

OPTIONAL Problem 5. The standard representation of a Voronoi diagram is a graph together with, for each vertex of the Voronoi diagram, a cyclic linked list of the incident edges in clockwise order around the vertex and, for each input point, a cyclic linked list of the vertices and edges around the Voronoi cell of that point.

(a) Show how to reduce the problem of sorting n numbers to the problem of computing the Voronoi diagram of $\Theta(n)$ points. Your reduction should take linear time, and can use standard arithmetic ($+$, $-$, $\cdot$, $/$, $\sqrt{\ }$) but cannot use trigonometric functions (sin, cos, etc.). (This is the *real RAM* model of computation.)

(b) Conclude that computing the Voronoi diagram of n points requires $\Omega(n \lg n)$ time in the worst case in the *algebraic decision tree* model of computation, in which the computation can branch based only on a binary decision of comparing two algebraic expressions (expressions involving inputs and $+$, $-$, $\cdot$, $/$, $\sqrt{\ }$), and the cost of a computation is the depth of that node in the tree.