6.838: Geometric Computing

Fall 2003

Problem Set 2
Due on **Tuesday, October 21**

MANDATORY PART. All problems are worth 10 points.

Problem 1. Given a set S of n vertical segments $s_1 \ldots s_n$ in the plane, construct a data structure that uses no more than $O(n \log n)$ space, that given a *horizontal* segment s, checks if s intersects any of segments in S in $O(\log^{O(1)} n)$ time.

Problem 2. Consider a convex polygon $\mathcal{P}$. Assume that the vertices of $\mathcal{P}$ are given in sorted order along the boundary in an array.

Show that, given a query point q, one can detect if $q \in \mathcal{P}$ in $O(\log n)$ time.

Problem 3. Exercise 7.11, p. 163. Let P be a set of n points in the plane. Give an $O(n \log n)$ time algorithm to find, for each point in P, another point in P closest to p.

Problem 4. A set of n circles splits the plane into a number of cells C, such that for each cell $c \in C$, any two points $p, q \in c$ have a continuous path connecting p and q that does not cross any circle boundary.

Show that the number of cells in C is $O(n^2)$.

Problem 5. Exercise 9.11, p. 209. A Euclidean Minimum Spanning Tree (EMST) of a set P of n points in the plane is a tree of minimum total edge length connecting all the points. Show that the set of edges of a Delaunay triangulation of P contains an EMST for P.

Show that this implies that one can compute EMST in $O(n \log n)$ time.

OPTIONAL THEORETICAL PART. All problems are worth 15 points.

Problem A. Assume you are given a set of n circles in the plane, which splits the plane into cells C as in Problem 4.

Construct a data structure of size $O(n^2)$, that given any (fixed) point q, finds the cell $c \in C$ containing q in (expected) $O(\log n)$ time.

Problem B .

Given n points P in $\Re^d$ and $\epsilon > 0$, show how to construct a data structure of size $O((n/\epsilon)^{O(d)}$, that given any $q \in \Re^d$, finds an $(1+\epsilon)$-approximate nearest neighbor of q in P in $(d + \log n + 1/\epsilon)^{O(1)}$ time.

Your description of the data structure must be self-contained (i.e., references to literature are NOT OK).

Hint: Construct a partition of $\Re^d$ that (a) enables fast point location and (b) has the property that within each cell of the partition, there is a unique $(1+\epsilon)$-approximate nearest neighbor.

Optional Programming Part

Implement a Java applet that *simulates* the algorithm of Lecture 6 for constructing a data structure for point location. Note that you are not required to implement that particular algorithm (although you are welcome to do so).

Your applet should first enable the user to provide a planar subdivision. For simplicity, the planar subdivision can be given by a set of n lines (the subdivision is then obtained by taking the trapezoid decomposition of the arrangement of those lines).[1]

Then, your algorithm should simulate the algorithm (as in Lecture 6) that constructs the data structure for point location. In particular, you algorithm should:

- Enumerate all segments defining the decomposition.
- For each new segment, update the trapezoid decomposition, and display the new decomposition on the screen.

[1]If you have a better idea for entering the decomposition, let me know, I am willing to accept it.