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

Handout 27
Wednesday, November 28, 2005

Problem Set 12 Solutions

Problem 1.

We can solve the problem independently for vertical and horizontal segments, and therefore, we assume without loss of generality that all line segments are vertical. For x-coordinates of line segments we build a balanced binary search tree, and at each node v of the tree we remember the maximum and minimum, i.e. the range, of the coordinates in the subtree rooted at v. For any query rectangle $[x_1, x_2] \times [y_1, y_2]$ we can easily find in $O(\log n)$ time $O(\log n)$ nodes w_i such that subtrees rooted at them contain line segments of the x-coordinate in $[x_1, x_2]$, and none of these nodes is an ancestor of any other. Furthermore, in each node v of the tree we create the interval tree of orthogonal projections of the line segments in the subtree rooted at v onto the y-axis. Using these trees at the previously selected nodes w_i, we can choose out of all line segments of the x-coordinate in $[x_1, x_2]$ these which y-coordinates intersect interval $[y_1, y_2]$. This can be done in $O(\log^2 n + k)$ time since each line segment belongs to at most one subtree rooted at w_i.

The size of the whole structure is bounded by $O(n \log n)$. This is implied by the fact that the size of the interval tree is $O(n)$ (this is not mentioned in the problem statement but true), where n is the number of intervals in the tree. We have $O(\log n)$ levels and on each of them a line segment belongs to at most one interval tree. Knowing this the total size of each level is of order n, and in total we get $O(n \log n)$.

Problem 2.

We choose an arbitrary half-line h of the end at the player (denote this point by p). Starting from and finishing at h we will sweep a half-line around p. We assume that p doesn't lie on any line containing any of the line segments. This assumption is not necessary, and can be avoided by an implementation of the algorithm that is more careful about details.

First of all, using h we split all line segments which it intersects into two line segments. If at least one of them is visible the corresponding line segment is visible. Now each line segment appears and disappears exactly once in one encirclement of p.

Looking from p, for each line segment we have two oriented angles α and β ($\alpha < \beta$, both in $[0, 2\pi]$) describing respectively when the line segment appears and disappears during the sweeping.

The sweep, which is a half line h', holds at a given time all the line segments it intersects. Using a balanced binary search tree, we can hold a list of the line segments which the sweep intersects in the order in which the intersections occur while moving from p along h'. Only line segments that are first on the list for some angle are visible from p. Both removal and insertion of a line segment cost $O(\log n)$ time. We can determine the right place for insertion of a line segment, comparing its intersection with h' to intersections with h' of line segments that already are in the sweep.

The algorithm sweeps half-line h' around p, inserting and removing line segments from the sweep. There are at most $2n$ line segments, and $4n$ angles at which we need to stop. We can sort them in the beginning in $O(n \log n)$ time. Since the line segments do not intersect, their order on the sweep does not change, and only line segments that are first on the list at some given angle are visible. The whole algorithm works in $O(n \log n)$ time.

Problem 3.

1. We have 3 points: U, V, W, and we can easily determine equations of bisectors of line segments UV and VW. Since the points are non-colinear, there exists a point P in which the bisectors intersect. P is easily computable from the equations of the bisectors, and the distance of P to each of the three given points is the same. Therefore, P is the center of the required circle, and $|PU|$ is its radius.

2. First, since the points are in general position, no circle passes through more than 3 points. On the other hand, if a circle C contains all points in finite set H, and there is no point on the boundary of the circle, we can decrease the radius of the circle by a little, getting a smaller circle, so C cannot be the smallest circle containing H. Also when there is only one point A on the boundary of C, we can move the center of the circle a little towards point A, getting a circle still containing all points and having no point on the boundary. This way we reduce it to the previous case, finishing the proof of the fact that $B(H)$ consists of 2 or 3 points.

Now, we will make a very important and useful observation, which we will denote by ($\star$). Note that all points in $B(H)$ cannot belong to an arc of $O(H)$ which is smaller than a half of the circle. As for one point on the boundary, this would imply that we can move the center of the circle by a bit towards the middle of the arc, getting a circle of the same radius, still containing H, but with no points in H on the boundary. The new circle could be shrunken as well.

Note that if there are only 2 points, B_1 and B_2, in the basis of $O(H)$, then by observation ($\star$) they must be on "opposite sides" of the circle so that B_1B_2 is a diameter of $O(H)$. It follows that the center of $O(H)$ is the middle of the line segment B_1B_2. To determine $O(H)$ we can enumerate all pairs and triples of points, for each choice create the corresponding circle, and among circles containing H find the smallest one. It has to be $O(H)$. The algorithm considers $O(n^3)$ candidates for $B(H)$, and for each of them it checks in time $O(n)$ if the corresponding circle contains H. The running time is then $O(n^4)$.

3. We will show that $O(H) = O(B(H))$. This obviously implies the fact which we need to proof.

Suppose that $O(H)$ is not the smallest circle containing $B(H)$. There is a circle C of smaller radius, containing $B(H)$. The center of C must differ from the center of $O(H)$, as this would imply that $B(H)$ is outside C. We shift $O(H)$, so that its center is the center of $B(H)$. On the one hand, the shifted circle contains C, and hence contains $B(H)$. On the other hand, the shifted circle does not contain a half-circle of the original $O(H)$, and by observation $(\star)$ this half-circle passes trough at least one point in $B(H)$. A contradiction.

4. Suppose that there is a point p not contained by $O(S)$, and that it does not belong to $B(S \cup \{p\})$. Let r_1 be the radius of $O(S \cup \{p\})$ and r_2 of $O(S)$. We have proven in part 3 that $O(S \cup \{p\}) = O(B(S \cup \{p\}))$. On the one hand, $S \subseteq S \cup \{p\}$, so $r_2 \leq r_1$, but on the other, $B(S \cup \{p\}) \subseteq S$, hence $r_1 \leq r_2$. Eventually, $r_1 = r_2$, and both $O(S)$ and $O(S \cup \{p\})$ must have a common center, since if they differed, shifting the center of $O(S)$ to the center of $O(S \cup \{p\})$, we would, similarly to the proof in part 3, exclude some point of $B(S)$. Finally, p is contained by $O(S \cup \{p\})$ which equals $O(S)$. A contradiction.

5. Suppose first that we are expected to find a circle passing through two specific points p_1 and p_2. If $C = O(\{p_1\} \cup \{p_2\})$, contains all the points, we are done. Otherwise, points that are not contained by C, must be on only one side of the line p_1p_2, since there is no circle passing through p_1 and p_2 that would contain more points on both sides. We need to move the center of the circle to this side along the bisector of p_1 and p_2 until the circle which it determines contains all the points. The radius of the circle increases, and we only need to find the point outside C that imposes the largest. We consider each point q outside C and compute the radius of the circle passing through p_1, p_2 and q. The smallest circle passing through p_1, p_2 is the circle passing also through q for which the radius is the largest. The running time is $T_2(n) = O(n)$.

Suppose now that we are given only one point p. We consider all other points in random order, adding one of them at a time. When we add the first point, we create the smallest circle containing it and p. For any next point p' we have two possible cases. Either p' belongs to the smallest circle for the points that we have already added, or it does not. If it does, we are done. Otherwise, the smallest circle passing through p and containing all points that we have already added must also pass through p' (we have not proven this yet!), and we call the linear algorithm for 2 fixed points to find the smallest circle.

What is the expected running time T_1 of the algorithm? If we have n points, not including the one that is fixed, the recurrence equation is

$$T_1(n) \leq T_1(n-1) + \frac{2}{n}T_2(n-1) + O(1),$$

since the probability that the point which we add is on the border is less than $2/n$. In total we get $T_1(n) = O(n)$.

We still do not know why the algorithm is correct. It turns out that all that was said for circles without fixed points has its equivalent for circles with one fixed point. The major difference is that instead of shifting circles in proofs we consider rotations around the fixed point. The equivalent of the key observation ($\star$) is the following:

> Let p be a fixed point, H a set of points ($p \notin H$), and $O_p(H)$ the smallest circle passing through p and containing H. At least one of points of H belongs to each of the two closed half-circles of $O_p(H)$ into which the diameter of an end at p splits $O_p(H)$.

We do not cover all the details, but everything is analogous to the case when we do not have fixed points. Specifically, we can prove the following claim:

> Let p be a fixed point, H a set of points, and q an arbitrary point. Assuming that all required circles exist, if $O_p(H)$ does not contain q, then q is on the boundary of $O_p(H \cup \{q\})$.

6. The general idea is like in the algorithm that is given one fixed point. We add points in random order one at a time. For two points the answer is trivial. If a newly added point p is contained by the smallest circle containing the hitherto subset of points, we are done. Otherwise, p is on the boundary of the new smallest circle (we have proven this!), and we use the algorithm that is given one fixed point to determine the smallest circle. Since the probability that the newly added point is on the boundary is at most $3/n$, we get the following recurrence equation for the expected running time $T_0(n)$ of the algorithm:

$$T_0(n) \leq T_0(n-1) + \frac{3}{n} T_1(n-1) + O(1).$$

It follows that the expected running time is of order $O(n)$.