Problem Set 12 (December 5, 2005) **Petar Maymounkov**
6.854 Advanced Algorithms petar@csail.mit.edu
with Benjamin Rossman, Oren Weimann, and Pouya Kheradpour

Problem 2. We build a data structure that upon a query returns all segments that cross or are fully contained in the query rectangle. We maintain separate data structures respectively for the horizontal and vertical line segments. So for the rest we assume we are dealing only with horizontal segments.

Each line segment is identified by x_0, x_1 and y. We maintain a master balanced binary tree whose leaves correspond to the y values of the line segments, and whose internal nodes correspond to disjoint intervals on the y axis, much like the data structures described in class.

Each node (internal or leaf) contains a secondary interval tree data structure, where we insert all line segment within the node's corresponding y interval, where the line segments are inserted only with their x_0 and x_1 coordinates.

Upon a query, we identify the minimal subset of the master tree's nodes whose union covers the query rectangle. This is achieved in $O(\log n)$ steps as discussed in lecture. For each node interval, we query the corresponding interval tree using a query that is the query rectangle's projection on the x-axis. There are $O(\log n)$ such sub-queries, and each takes $O(\log n + k_i)$ steps, where k_i is the number of positive matches within the i-th tree. Therefore, the total query time is $O(\log^2 n + k)$.

The size of the master tree is $O(\log n)$, and observing that each line segment is present in $O(\log n)$ interval trees, the total space taken by the interval subtrees is $O(n \log n)$. Therefore the overall space is $O(n \log n)$.

Problem 3. Begin by reading in the locations (coordinates) of all segment endpoints, and for each endpoint Q compute the angle between the line PQ, where P is the "pivot" (point of view) point, and the x-axis: runtime $O(n)$. Sort all endpoints by their corresponding angle, and break ties as follows: for all "arriving" endpoints of line segments break ties in descending order of distance to the pivot, and otherwise for all "departing" points. Runtime $O(n \log n)$.

Initialize an empty priority queue implemented by a binary search tree, where line segments will be present by their "current" distance away from the pivot P. Traverse the sorted endpoints. The sorting ensures that this traversal visits the endpoints in the same order as

a, say, counter-clockwise sweep line.

Define an "arrival" event to be the event of traversing the first endpoint of a given line segment. Respectively, define a "departure" to be the traversal of the second endpoint (in sort order) of a line segment. Upon arrival, insert the endpoint's corresponding line segment in the heap such that the insertion location preserves the heap order (define above). To do this, do a binary search into the tree to find the propert insertion point. (At each step of the insertion binary tree, we compute the current distance of the heap element being compared against). Upon departure, remove the corresponding line segment from the heap.

Because line segments do not intersect, between sweep-line events the heap order of line-segments matches the order of line segments defined by distance from the pivot. Furthermore this order can never change, while the line segments are in the heap.

Multiple sweep-line events might occur at the same time step. Process all departures before arrivals. Upon departure, remove line segments from the top (closest to pivot) of the heap going down. Output each departing line-segment as visible, if it is removed from the current top of the heap. Upon arrival, process all arriving events in descending order of distance from the pivot. Insert each arriving line-segment in the heap, and if it is inserted at the top of the heap, also output it as a visible segment. Once all arrivals have been processed, if the heap is not empty, output the line segment at the top of the heap as the single segment visible between the current and next sweep events.

Begin the sweep-line traversal at an arbitrary endpoint event. Initialize the initial state of the heap by a one-time linear traversal of the endpoint sequence, ensuring that the heap starts off in a valid state.

The total runtime is $O(n \log n + n)$ for sorting and establishing the initial heap configuration, plus an additional $O(n \log n)$ time for processing all endpoint events. The total space used is simply the size of the heap $O(n)$.

Problem 4a. The points (x, y) on a circle with center (x_c, y_c) and radius r are defined by the Pythagorean relation $(x - x_c)^2 + (y - y_x)^2 = r^2$. Three points are therefore sufficient to determine a unique solution. To see this, pick 2 different pairs of equations and subtract one from the other in each pair. This produces a system of 2 linear equations on 2 unknowns. This produces a solution for the center. Then plug this solution back into the original system, and observe that it is either solvable or not (only in the case when the points are colinear).

Problem 4b. By assumption there are no 4 co-circular points in H, therefore $O(H)$ can contain $0, 1, 2$ or 3.

If it has 0 points, find the farthest point from the center C, and shrink the radius to equal the distance from the center to that point. This produces a smaller circle – contradiction.

If it has 1 point, let A be on the point on the circle, B be the farthest point from the center C (after A), and X be any other point in H. Let $|CB| = r - \delta$ where $\delta > 0$. Consider a new circle centered at $C' \in CA$ with $|CC'| = \delta/2$ and $r' = r - \delta/2$. By triangular inequality $|C'B| \leqq |CC'| + |CB| = \delta/2 + r - \delta = r'$, $|C'A| = r'$, and $|C'X| \leqq |C'C| + |CX| \leqq |CC'| + |CB| \leqq \delta/2 + r - \delta = r'$. Contradiction.

Further, we prove that if 2 points define $O(H)$ then these points must be diametric. Assume not. Let X and Y be the basis points, C be the center, D be C's projection on XY, Z be the point farthest from the center (after X and Y), Q be any other point of H. Let $|CZ| = r - \delta$ and $|CD| = \epsilon$ where $\epsilon, \delta > 0$. Consider a new circle, centered at $C' \in CD$ with $|CC'| = \gamma = \min\{\delta/2, \epsilon/2\}$. Then the new radius is $r' = |XC'| = |YC'| \geqq r - \epsilon/2$ (by $\triangle XCC'$), and $|ZC'| \leqq r'$ by $\triangle ZCC'$, and $|QC'| \leqq r'$ by $\triangle QCC'$.

So, algorithmically, consider all choices of 2 or 3 basis points: $\binom{n}{3} + \binom{n}{2} = O(n^3)$, and for each check that all points fall within in $O(n)$ time. For a total runtime of $O(n^4)$.

Problem 4c. To the contrary, assume $C = O(B(H))$ and C misses a point of H. Then $B(H) \neq B(B(H))$, because otherwise it wouldn't miss a point in H. So $|B(B(H))| < |B(H)|$, but also $|B(H) = 2, 3|$ and $|B(B(H))| = 2, 3$, therefore, $|B(H)| = 3$ and $|B(B(H))| = 2$ (in the remaining cases the claim we are proving holds).

Therefore one of the 3 points in $B(H)$ is inside $O(B(H))$ and therefore the circumcenter of $B(H)$ is outside the triangle defined by $B(H)$. But then, using the same triangular inequality argument from part 4b, we can push the center of $O(H)$ towards one of $B(H)$'s points by δ and shrink the radius, which contradicts that it is the smallest circle capturing H. Therefore, $B(H) = 2$ or $B(H)$ forms an acute triangle.

Problem 4d. First, we prove that $O(S)$ is unique. Assume not, let C and D be two equally small circles covering S. Then C and D must intersect in a non-zero region, where all the points lie. Consider the circle E whose diameter is the segment between the 2 intersection points of the circumferences of C and D. Then E is strictly smaller than either and contains all points of S – contradiction.

Assume for contradiction that p is not on the boundary of $O(S \cup p)$. Then $B(S \cup p) \subset S$. By

part 4c, $O(S \cup p) = O(B(S \cup p)) = X$. Therefore (a) $X = O(B(S \cup p))$ is the smallest circle containing a subset $B(S \cup p)$ of S and (b) at the same time X contains all of $S \cup p$ because $X = O(S \cup p)$. Point (a) implies that X is smaller than or equal in radius with $O(S)$. If it is smaller, then we reach a contradiction because of property (b). If it is equal in radius to $O(s)$, then we must have $X \neq O(S)$ because $p \in X$ and $p \notin O(S)$, but then we violate the uniqueness that we proved above. And we are done.

Problem 4e. For two points. Fix A and B. Then consider the radius of the circle C^*, defined by diamter AB. Also consider the radii of all circles defined by A, B and D for all D among the remaining vertices for which $D \notin C^*$. Pick the circle that induces the maximum radius and check if it contains all points. This is done in $O(n)$.

Correctness. Consider just the half-plane above AB. For any point D not in C^*, the smallest circle through AB that contains this point has radius at least as big as that of the circle through ABD. Apply similar argument on the other half-space.

Order the points randomly, call that order $p_1, \ldots, p_{n-1}$ with p_0 the fixed point. Start at stage 1 where the circle is defined by p_0 and p_1. At stage $k+1$ consider p_{k+1}. If it is in the current circle, do nothing. Otherwise, run the 2-fixed-points algorithm on p_0 and p_{k+1}. Here we are assuming that both p_0 and p_{k+1} are on the boundary of the new circle. This doesn't necessarily follow from part 4d (because it doesn't address fixed points), but it follows from the fact that p_0 is a true basis point for the entire set, and hence it is a basis point for all subsets that include it.

Let R_k be the running time up to the k-th stage. For R_{k+1} we have:

$$E[R_{k+1}] = E[R_k] + \frac{2}{k+1} T(k+1)$$

where $T(k) = O(k)$ is the runtime of the 2-fixed-point algorithm, and $2/(k+1)$ is an upper bound on the probability that p_{k+1} is one of the basis points of $p_0, \ldots, p_{k+1}$ other than p_0. The recursion resolves to $E[R_n] = O(n)$.

Problem 4f. Randomize the order of all points, WLOG $p_1, \ldots, p_n$. Let H_i be the set of the first i points. Start with $O(H_2)$ and call this stage 2. At stage $k+1$, consider p_{k+1}. If it is in $O(H_k)$, do nothing, otherwise run the 1-fixed-point algorithm on p_{k+1} and H_k, where p_{k+1} is fixed. Let R_k be the runtime up to the end of the k-th stage:

$$E[R_{k+1}] = E[R_k] + \frac{3}{k+1} Q(k+1)$$

where $Q(k) = O(k)$ is the expected runtime of the 1-fixed-point algorithm. (Note that this equation is correct, because the 1-fixed-point algorithm is probabilistically independent from the present algorithm). Also $3/(k+1)$ is an upper bound on the probability that p_{k+1} is one of the 2 or 3 basis points of H_{k+1}. The recursion for R_n resolves to $R_n = O(n)$.