

1 Problem 1

Reference: “Computing common tangents without a separating line” by David Kirkpatrick and Jack Snoeyink. (<http://www.springerlink.com/content/73012v3604577424/>)

2 Problem 2

Reference: “Algorithms for bichromatic line-segment problems and polyhedral terrains” by Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, and Micha Sharir.

(<http://www.springerlink.com/content/12571r7432j4tp28/>)

3 Problem 3 (From Steve Gu)

First note that maximal points can be computed in $O(n \log n)$ running time. We can sort all the points according to their y coordinates in a non-increasing order and find the maximal points by tracking the x coordinates in an increasing order. To make this algorithm output-sensitive, we follow the similar thinking to the $O(n \log h)$ algorithm for convex hull where the original problem is divided to n/h component, each containing h points. We can find the maximal points for each component in $n/h * h \log h = O(n \log h)$ time. To merge those maximal points, we start from the top point and each time we use binary search among the maximal points of each component (which is already sorted) to find the next highest point whose x coordinate is increasing. The running time is again $n/h * h \log h = O(n \log h)$. Therefore, the total running time is $O(n \log h)$.

In \mathbb{R}^3 , the algorithm needs a little bit extra work. We first sort all the points according to their z coordinates. We then use divide and conquer to recursively find the maximal points for each layer, which one can think of as a (x, y) plane with z specified and all the points are sorted by (x, y) coordinate in dictionary order. Merging two sets of maximal points takes linear time of the number of maximal points because one can take one scan of all the points and determine if it belongs to the new maximal points set. Therefore, the total running time $T(n) = 2T(n/2) + n$ leads to $O(n \log n)$ algorithm. To make it output-sensitive, we apply the same technique, that is, we divide all the points to n/h components, each containing h points. We start with the top point and each time we add one maximal point by querying all the components and find the next valid maximal points using binary search. The total running time is again $n/h * h \log h = O(n \log h)$.

4 Problem 4 (From Sayan Bhattacharya)

Sort the segment endpoints in increasing order according to the angles they make in counterclockwise direction with a downward vertical ray originating from p . Rotate a sweep ray originating from p in counterclockwise direction, the event points being the segment endpoints, sorted as mention above. At any time instant, the data structure associated with the sweep ray is a balanced binary tree storing all the segments intersecting the ray. For any two segments s_1 and s_2 stored in the tree, s_1 comes before s_2 in inorder traversal if and only if s_1 is nearer to p than s_2 in the direction of the sweep ray. A segment is marked if at some point of time it has been discovered to be visible from p , and unmarked otherwise. At each event point, the algorithm does the following:

1. Search the tree to find out whether or not the corresponding segment s is already present.
2. If s is already present, then delete it from the tree. If the segment nearest to p in the new tree is unmarked, report and mark it as visible.
3. If s is not already present, then insert it into the tree. If s turns out to be the segment nearest to p in the new tree, report and mark it as visible.

Since the tree is balanced, each of the above steps takes $O(\log n)$ time. Since there are $2n$ number of event points, the worst case run time of the above algorithm is $O(n \log n)$.

5 Problem 5

Reference: “Solving Geometric Problems with the Rotating Calipers” by G. T. Toussaint. (<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.40.2140>)

The proof that one of the sides of the rectangle is flushed with a convex hull edge is presented in the following paper: “Determining the minimum area encasing rectangle for an arbitrary closed curve” by Freeman and Shapira. (<http://portal.acm.org/citation.cfm?id=360919>)

5.1 Sample solution from Albert and Sharath

5.1.1 Subdivision

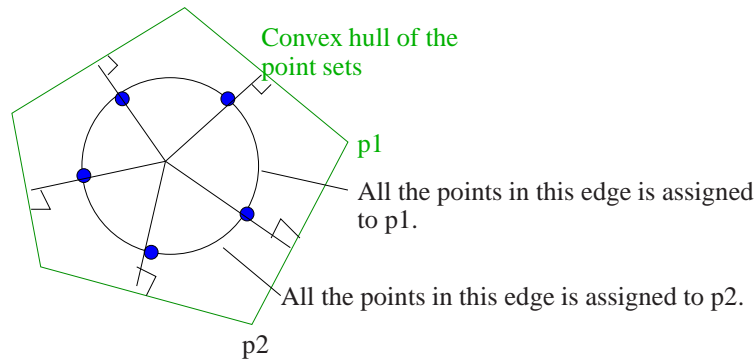


Figure 1: Case 1: The origin is inside the convex hull.

See figures 1 and 2. Let $Conv(P)$ be the convex hull of P . The subdivision P^* of \mathbb{S}^1 is the dual graph of $Conv(P)$. Every edge e of $Conv(P)$ corresponds to a vertex e^* on \mathbb{S}^1 such that the direction represented by e^* corresponds to the normal of e . Every vertex v of $Conv(P)$ corresponds to an arc v^* on \mathbb{S}^1 where the endpoints of the arc correspond to the normals of the two edges incident on v .

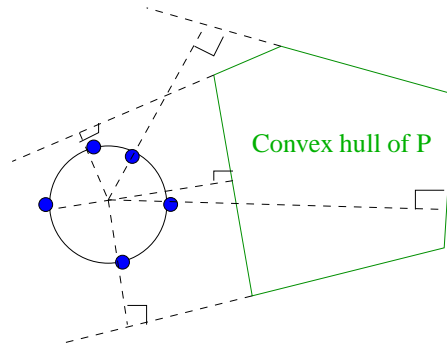


Figure 2: Case 2: The origin is outside the convex hull.

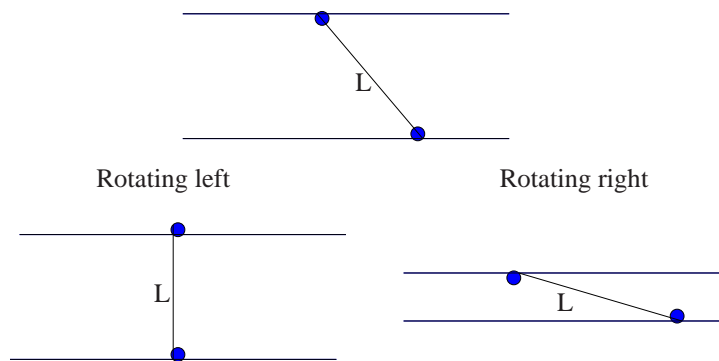


Figure 3: The distance between the parallel lines can increase or decrease depending on the direction of the rotation.

5.1.2 Containing an edge of $Conv(P)$

Consider two points p_i and p_j as well as the two parallel lines through these two points. The distance between these two lines can increase or decrease depending on the direction of rotation as shown in Figure 3. Assume a strip that realizes the width of P does not contain an edge. Let p_i and p_j be the antipodal pair. Observe that p_i and p_j always lie on the two parallel lines when the lines are rotating. Therefore, it is always possible to rotate the parallel lines to further decrease the width.

5.1.3 Computing the width of P

Algorithm:

1. Find the convex hull of P
2. For each edge of the convex hull, find the corresponding antipodal pair.
3. Return the antipodal pair that has the minimum width.

Computing a convex hull takes $O(n \log n)$ time. Next, we will show that for each edge e , the corresponding antipodal pair can be computed in $O(\log n)$ time. Refer to figure 4. Finding

the corresponding vertex of an antipodal pair is the same as finding the edge that has the opposite normal in the dual graph. The edge can be found by doing a binary search on the edges of P^* . This takes $O(\log n)$ time. Since there are $O(n)$ edges, the total running time is $O(n \log n)$.

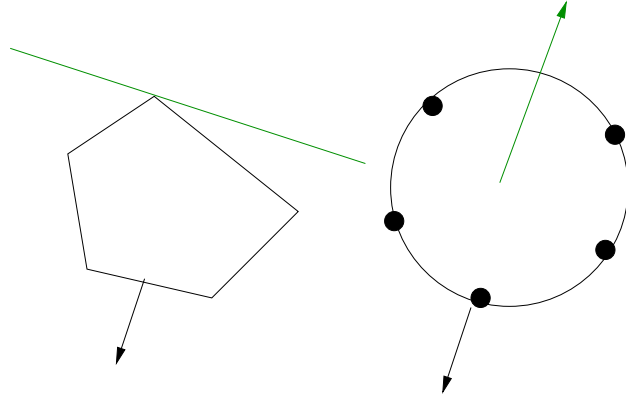


Figure 4: The minimum width problem can be solved in the dual graph.

5.1.4 Computing the minimum-area rectangle of P

Any rectangle can be described by four vertices $v_1, v_2, v_3, v_4 \in \text{Conv}(P)$. We describe a way of obtaining $O(n)$ candidates for these vertices. Consider a dual P^* of $\text{Conv}(P)$. Recollect that P^* is a subdivision of S' . Obtain P_1^*, P_2^*, P_3^* by rotating P^* by $\pi/2, \pi, 3\pi/2$, respectively. Overlay P_1^*, P_2^*, P_3^*, P^* to obtain Q^* . Note Q^* has at most $4n$ vertices and arcs. Also, in the primal, every arc of Q^* corresponds to a vertex v and a set of normals N . It is easy to see that for any rectangle whose side passes thru v with a normal $n \in N_S$, the four vertices which define the rectangle are identical. Thus, for each arc of Q^* , we obtain a candidate quadruple. For each quadruple, we find the minimum area rectangle that passes through them. We output the minimum area rectangle of all the $4n$ such rectangles.