

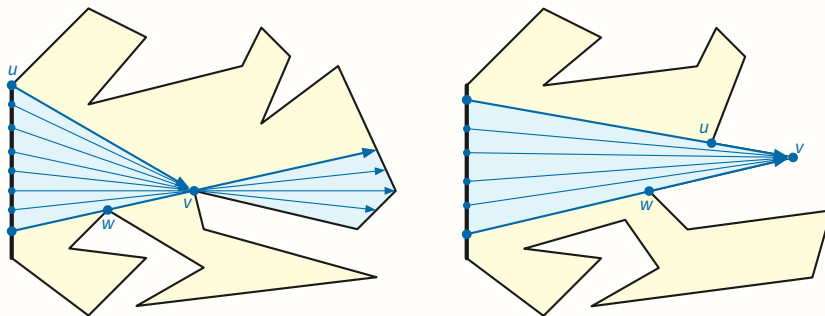
1. Suppose you are given a simple polygon P with a single vertical edge ℓ . Describe and analyze a data structure that can efficiently answer queries of the following form: Given a ray r whose basepoint lies on ℓ , find the first edge of P that r hits. For full credit, your data structure should use $O(n)$ expected space, and your query algorithm should run in $O(\log n)$ expected time. [Hint: Reduce to a problem you've seen before!]

Solution: Without loss of generality, assume that P lies entirely in the halfplane $x \geq 0$ and its unique vertical edge has endpoints $(0,0)$ and $(1,0)$. To avoid later ambiguity, call the vertical edge ℓ (for “leftmost”) instead of “ e ”.

We reduce our ray-shooting problem to point location in a planar straight-line graph. Each ray from ℓ lies on a unique non-vertical line; we represent the ray along the line $y = ax - b$ by its dual point (a, b) . Thus, the space of all rays is the infinite horizontal slab $\mathbb{R} \times [-1, 0]$.

We need to disambiguate some boundary conditions. Consider a ray r that hits a vertex v of P before hitting the interior of any edge. If both edges incident to v are on the same side of r , then the ray r continues beyond v ; but if the edges incident to v lie on opposite sides of r , then the ray stops at v (and we can report either incident edge as the edge that r hits first.)

If we continuously change a ray r by moving its dual point $r^* = (a, b)$, the edge that r first hits changes only when r passes through a vertex of P . For each vertex v , let $S(v)$ denote the set of all rays that contain vertex v . If $S(v)$ is nonempty, then it is a *segment* of the line dual to v . Specifically, starting with any ray r through v , the endpoints of $S(v)$ can be found by rotating r around v either clockwise or counterclockwise until it intersects another vertex u that is closer to ℓ than v . See the figure below for two examples.



The segments $S(v)$ define a planar straight-line graph G . If v is one of the endpoints of ℓ , the segment $S(v)$ is actually an infinite horizontal line; otherwise, each endpoint of $S(v)$ lies in the interior of another segment $S(u)$. Every vertex of G is an endpoint of some segment $S(v)$, so G has two vertices at infinity with degree 2 and at most $2n - 4$ vertices with degree 3, and therefore has at most $3n - 4$ edges and (by Euler's formula) at most $n - 1$ faces. Each face of G contains all rays that end at a particular edge of P . (There is no face for the vertical edge ℓ .)

If we already know the graph G (or even just the segments $S(v)$), we can build a trapezoidal decomposition of G in $O(n \log n)$ expected time using a randomized incremental algorithm. The history dag of the randomized incremental construction has expected size $O(n)$ and answers point-location queries in $O(\log n)$ expected time. ■

This application of projective duality was first described by Chazelle and Guibas;^a they also described a divide-and-conquer algorithm to construct the graph G directly from the polygon P in $O(n \log n)$ time. There is a conceptually simpler data structure (but with more hairy low-level details) that handles ray-shooting queries in simple polygons from *any* interior base point in the same space, preprocessing-time, and query-time bounds.^{bc}

^aBernard Chazelle and Leonidas J. Guibas. [Visibility and intersection problems in plane geometry](#). *Discrete Comput. Geom.* 4(6):551-581, 1989.

^bBernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas Guibas, John Hershberger, Micha Sharir, and Jack Snoeyink. [Ray shooting in polygons using geodesic triangulations](#). *Algorithmica* 12(1):54-68, 1994.

^cJohn Hershberger and Subhash Suri. [A pedestrian approach to ray shooting: Shoot a ray, take a walk](#). *J. Algorithms* 18:403-431, 1995.

2. Suppose you are given n points in the plane, in general position. These points are the centers of circular disks, all with the same radius r .
- Suppose you are also given the radius r . Describe and analyze an efficient algorithm to determine whether the union of these disks is connected.
 - Now suppose you are *not* given the radius r . Find the minimum radius r such that the union of radius- r disks centered at the given points is connected. (Yes, this algorithm immediately implies a one-line solution to part (a).)

For full credit, both algorithms should run in $O(n \log n)$ time. Don't forget to prove that your algorithms are correct!

Solution: To answer part (b), we return half the length of the longest edge in the Euclidean minimum spanning tree of the input points. We can compute this length in $O(n \log n)$ time by first computing the Delaunay triangulation T and then computing the minimum spanning tree of T using any textbook algorithm.

Okay, there's a lot to unpack here. I'll start by solving part (a) directly. First let's establish some notation.

- Let P be the given set of points.
- Let $Del(P)$ denote the Delaunay triangulation of P , and let $Del_r(P)$ denote the subgraph of $Del(P)$ of all edges with length at most $2r$.
- For each point $p \in P$, let $D_r(p)$ denote the closed disk of radius r centered at p , and let $U_r(P) = \bigcup_{p \in P} D_r(p)$.
- Finally, for each point $p \in P$, let $Vor(p)$ denote the closed Voronoi region of p in the Voronoi diagram of P , and let $Vor_r(p)$ denote the restricted Voronoi region $Vor(p) \cap D_r(p)$.

Lemma 2.1. $U_r(P) = \bigcup_{p \in P} Vor_r(p)$

Proof: By definition, $Vor_r(p) \subseteq D_r(p)$ for every point $p \in P$. It immediately follows that $\bigcup_{p \in P} Vor_r(p) \subseteq \bigcup_{p \in P} D_r(p) = U_r(P)$.

On the other hand, let q be any point in $U_r(P)$. We must have $q \in D_r(p)$ and thus $|pq| \leq r$ for some $p \in P$. Let p' be q 's nearest neighbor in P , so $q \in Vor(p')$. We must have $|p'q| \leq |pq| \leq r$, and thus $q \in D_r(p')$. We conclude that $q \in Vor_r(p')$. \square

Lemma 2.2. $U_r(P)$ is connected if and only if $Del_r(P)$ is connected.

Proof: Suppose $U_r(P)$ is connected. Let p and q be any two points in P ; these two points are connected by a path $\pi: [0, 1] \rightarrow U_r(P)$. Let $p = p_0, p_1, p_2, \dots, p_\ell = q$ be the sequence of nearest neighbors of $\pi(t)$ as t increases from 0 to 1.^a The nearest neighbor of $\pi(t)$ changes from p_{i-1} and p_i precisely when $\pi(t)$ lies on the Voronoi edge $Vor(p_{i-1}) \cap Vor(p_i)$. Thus, for every index i , segment $p_i p_{i-1}$ is a Delaunay

edge; moreover, $|p_{i-1}p_i| \leq 2r$, so $p_i p_{i-1}$ is a edge of $Del_r(P)$. We conclude that $p_0 \rightarrow p_1 \rightarrow p_2 \rightarrow \dots \rightarrow p_\ell$ is a walk from p to q in $Del_r(P)$.

On the other hand, suppose $Del_r(P)$ is connected. Let p and q be two arbitrary points in $U_r(P)$, not necessarily in P . Let \bar{p} and \bar{q} be the nearest neighbors in P of p and q , respectively. Because $Del_r(P)$ is connected, there is a walk $p_0 \rightarrow p_1 \rightarrow p_2 \rightarrow \dots \rightarrow p_\ell$ in $Del_r(P)$ from $p_0 = \bar{p}$ to $p_\ell = \bar{q}$. The polygonal path $\bar{p} \rightarrow p_0 \rightarrow p_1 \rightarrow p_2 \rightarrow \dots \rightarrow p_\ell \rightarrow \bar{q}$ lies entirely in $U_r(p)$. \square

Theorem 2.3. *Given a set P of n points in the plane and a real number $r > 0$, we can determine in $O(n \log n)$ time whether $U_r(P)$ is connected.*

Proof: First construct $Del(P)$ in $O(n \log n)$ time, using, for example, a randomized incremental algorithm. Then remove all edges longer than $2r$, and finally test whether the remaining subgraph $Del_r(P)$ is connected (using whatever-first search). The last two steps take only $O(n)$ time, because the Delaunay triangulation is planar. \square

To solve part (b), we need to find the smallest value of r such that $Del_r(P)$ is connected.

Recall Kruskal's minimum-spanning-tree algorithm. Given any connected graph G with weighted edges, we initialize a forest F containing the vertices of G but no edges. Then we consider the edges of G in increasing order of length; for each edge e , if the endpoints of e are different components of F , we add e to F .

After the k th iteration of the main loop, F is the minimum spanning forest of the subgraph of G induced by its k lightest edges. The last edge e^* added to F is the heaviest edge of the minimum spanning tree. It follows that the subgraph of G induced by edges lighter than e^* is disconnected, but adding e^* to that subgraph makes it connected.

Now let $G = Del(P)$, and let pq be the longest edge of the minimum spanning tree of $Del(P)$. We have just argued that $r = |pq|/2$ is the smallest value of r such that $Del_r(P)$ is connected. (The minimum spanning tree of $Del(P)$ happens to be the Euclidean minimum spanning tree of P , but we don't really need that fact here. See David Mount's lecture notes for a short proof.) \blacksquare

^aFormally, this sequence might have infinite or even uncountable length if the path π fractally wiggles around Voronoi vertices/edges, but a simple compactness argument implies that there is a suitable finite subsequence.

3. Let P be a convex polygon in the plane with n vertices, represented as a circular doubly-linked list of vertices in counterclockwise order. A simple randomized incremental algorithm constructs the Delaunay triangulation of the vertices of P .

- (a) Prove that the expected number of flips performed by this algorithm is $O(n)$.

Solution: Imagine running the algorithm backward: Choose a random point q , flip away each diagonal incident to q , delete q and its two incident convex hull edges, and recursively delete the remaining triangulation. The Delaunay triangulation of P has exactly $n - 3$ diagonals, each incident to two vertices, so the expected number of diagonals incident to a random vertex is $(2n - 6)/n < 2$. Thus, the expected number of flips to delete q is less than 2.

We conclude that the expected number of flips over the entire algorithm is less than $2n = O(n)$. (The *exact* expected number of flips is $2n - 6H_n + 5$.) ■

- (b) We already know from Lawson's algorithm that we can perform the necessary flips in $O(1)$ time each. But how do we choose the random vertex q in $O(1)$ time?

Solution: In a preprocessing phase, we build an array $Shuffled[1..n]$ of pointers, where $Shuffled[i]$ points to the i th vertex of P , and then randomly shuffle this array. Building and shuffling this array takes $O(n)$ worst-case time.

Finally, we modify CONVEXDELAUNAY to insert points in the (random!) order specified by the array $Shuffled$. ■

4. One way to solve the “Formula 6–7” problem from the midterm is to build a variant of motorcycle graphs from Homework 2. We model each car as a moving point $p_i(t) = (x_i + t \cdot u_i, t)$ whose x -coordinate is the position of car i at time t , and the y -coordinate is the current time t . Each moving point p_i traces out a line segment s_i in the plane, which ends when p_i is passed by a faster moving point p_j . The resulting set of segments is called the *6-7-graph* of the moving points.

Suppose we construct the 6-7-graph by choosing one moving point p_j at random, recursively constructing the 6-7-graph of the other $n - 1$ moving points, and then inserting the final segment s_j .

- (a) What is the *exact* expected number of segments that terminate on s_j , as a function of n ?

Solution: The exact answer actually depends on a second parameter; let k denote the number of components of the 6-7-graph (= cars that are never passed). There are n segments and $n - k$ termination points. Thus, the expected number of termination points on a random segment is **exactly $1 - k/n$** . ■

- (b) Each segment in the 6-7-graph has a unique face to its left. What is the *exact* expected number of segments s_i that touch the boundary of the face to the left of s_j ?

Solution: Here we understand each “face” to be bounded by the x -axis, even though the x -axis is not actually part of the 6-7-graph. The exact answer for this problem depends on *three* parameters:

- n is the number of segments (= cars);
- k is the number of components of the 6-7-graph (= cars that are never passed);
- r is the number of edges of the rightmost face of the 6-7-graph (= cars that are ever in first place).

For each index i , let f_i denote the unique face with segment s_i on its right boundary, and let f_R denote the rightmost face. Each segment that touches the left boundary of a face contributes exactly one edge to that left boundary. So the expected number of segments touching f_j is equal to 1 plus the number of edges on the left boundary of focus on the expected number of *edges* on the left boundary of f_j .

The 6-7-graph is a forest of k binary trees with a total of n leaves, with an extra infinite edge going up from each root, so it has exactly $2n - k$ edges, r of which lies on the boundary of f_R . Each edge is on the left boundary of a unique face, so there are $2n - k - r$ pairs (e, f) where e is an edge on the left boundary of face $f \neq f_R$.

Thus, if we choose the index j uniformly at random, the expected number of edges on the left boundary of f_j is exactly $(2n - k - r)/n = 2 - k/n - r/n$.

We conclude that the expected number of segments touching f_j is **exactly $3 - k/n - r/n$** . ■

5. Explain how to solve each of the following problems in linear expected time, by reducing them to $O(1)$ instances of fixed-dimensional linear programming.

(a) Suppose you are given a set of n red points and n blue points in the plane. Find an *proper axis-aligned ellipse* that has all red points *strictly* inside and all blue points *strictly* outside, or correctly report that no such ellipse exists.

Solution: We search for the five coefficients of an axis aligned ellipse $Ax^2 + By^2 + Cx + Dy + E = 0$. To handle the strict inequalities describing both proper ellipses and strict separation, we add a sixth margin variable $\delta \geq 0$, which we try to maximize, using the following six-dimensional linear program:

maximize δ
subject to $Ax_i^2 + By_i^2 + Cx_i + Dy_i + E \leq -\delta$ for every red point (x_i, y_i)
$Ax_j^2 + By_j^2 + Cx_j + Dy_j + E \geq \delta$ for every blue point (x_j, y_j)
$A \geq \delta$
$B \geq \delta$
$\delta \geq 0$

Assuming the LP is feasible, let $(\delta^*, A^*, B^*, C^*, D^*, E^*)$ denote its optimal solution. There are three cases to consider.

- If $\delta^* > 0$, then $A^*x^2 + B^*y^2 + C^*x + D^*y + E = 0$ is the equation of a *proper* axis-aligned ellipse that *strictly* separates the red and blue points.
- If $\delta^* = 0$, no strictly separating proper ellipse exists. Instead, $A^*x^2 + B^*y^2 + C^*x + D^*y + E = 0$ is the equation of a *degenerate* axis-aligned ellipse that *weakly* separates the red and blue points.
- Finally, if the LP is infeasible, no separating ellipse exists, even if we allow degenerate ellipses and weak separation.



- (b) Let P be a simple polygon in the plane, represented as a cyclic sequence of vertices p_1, p_2, \dots, p_n in counterclockwise order. Find the largest circle inside the kernel of P .

Solution: For notational convenience, let $p_0 = p_n$. The signed distance between any interior point g to the line through any edge $p_{i-1}p_i$ is exactly

$$\frac{\Delta(g, p_{i-1}, p_i)}{\text{dist}(p_{i-1}, p_i)}$$

where Δ is the usual orientation determinant. In particular, if g is on the wrong side of $p_{i-1}p_i$, then this expression is negative.

The following linear program has three variables: the coordinates of the guard point g and the radius r of the circle.

$$\begin{array}{l} \text{maximize } r \\ \text{subject to } \Delta(g, p_{i-1}, p_i) - \text{dist}(p_{i-1}, p_i) \cdot r \geq 0 \quad \text{for all } i \end{array}$$

Each inequality specifies that the guard point g is not only on the correct side of the line through $p_{i-1}p_i$, but has distance at least r from that line. The determinant $\Delta(g, p_{i-1}, p_i)$ is linear in the coordinates of g , so this is fact a linear program.

The linear program is always feasible and bounded. Let (g^*, r^*) be the optimal solution. If $r^* < 0$, the polygon is not star-shaped; otherwise, g^* is the center and r^* is the radius of the desired circle. ■