1. Suppose you are given a set L of n lines in the plane in general position. Describe an algorithm to compute the smallest axis-aligned rectangle that contains all $\binom{n}{2}$ vertices of the arrangement of L. For full credit, your algorithm should run in $O(n \log n)$ time. *[Hint: What is the dual of the rightmost arrangement vertex?]*

Solution: The arrangement vertex with largest x-coordinate is dual to the line with largest slope through two dual points in L^*. Those two points must be adjacent in x-coordinate order.

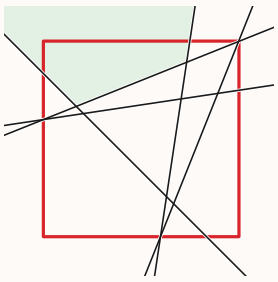
We can prove this claim as follows: Fix three arbitrary points p, q, r with distinct x-coordinates, in order from left to right. If q lies above $\overleftrightarrow{pr}$, then $\overleftrightarrow{pq}$ has larger slope than $\overleftrightarrow{pr}$. On the other hand, if q lies below $\overleftrightarrow{pr}$, then $\overleftrightarrow{qr}$ has larger slope than $\overleftrightarrow{pr}$. In both cases, $\overleftrightarrow{pr}$ is not the line with maximum slope.

Equivalently, the two lines that determine the rightmost arrangement vertex must be adjacent in slope order. We can prove this claim directly using a sweep-line argument: Imagine sweeping a vertical line ℓ from right to left across the arrangement of L. Initially, when ℓ is far to the right, the lines in L intersect ℓ in slope order. Thus, as we move ℓ to the right, the first change in the intersection order along ℓ (the rightmost arrangement vertex) must be between two lines with adjacent slopes.

Thus, we can find the rightmost arrangement vertex in $O(n \log n)$ time by sorting the lines in L by slope, computing the intersection point of each adjacent pair in sorted order, and returning the rightmost of those $n-1$ points.

Similar algorithms find the highest, lowest, and leftmost arrangement vertices. In fact, all four extreme vertices are intersections of lines that are adjacent in slope order (wrapping around at $\pm\infty$). The entire algorithm runs in $\boldsymbol{O(n \log n)}$ ***time***. ■

Non-solution: The highest arrangement vertex (that is, the vertex with maximum y-coordinate) must lie on the upper envelope of L. Thus, to find the highest arrangement vertex, we can compute the upper envelope in $O(n \log n)$ time, and then find the highest of its vertices in $O(n)$ time. Similar algorithms find the other three extreme vertices. The entire algorithm runs in $\boldsymbol{O(n \log n)}$ ***time***.



The highest vertex in a line arrangement is not necessarily on the upper envelope.

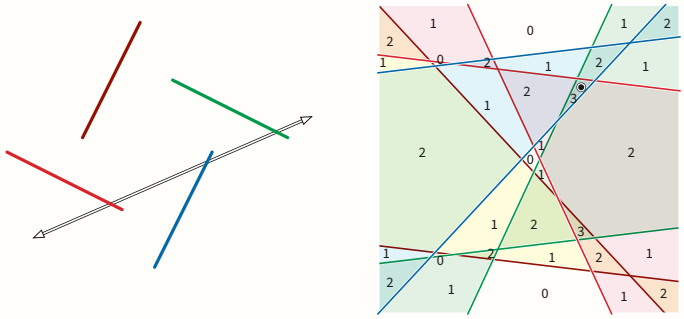
♣

Rubric: 10 points

2. Suppose you are given a set S of n line segments in the plane.

   (a) Describe and analyze an algorithm that either finds a line that intersects every segment in S, or correctly reports that no such line exists. *[Hint: What is the dual of a line segment?]*

Solution: We solve a dual formulation of the problem, using the standard duality $(a, b) \leftrightharpoons y = ax - b$. A line segment $s = \overline{pq}$ is the set of all points between one endpoint p and the other endpoint q on the line $\overleftrightarrow{pq}$. Thus, the dual s^* of s is the set of all lines between one line p^* and another line q^* that pass through the dual point $(\overleftrightarrow{pq})^*$. This set of lines sweeps out a double wedge through $(\overleftrightarrow{pq})^*$, with p^* and q^* on its boundary. Another line ℓ intersects s if and only if the dual point ℓ^* lies inside the double wedge s^*; that is, either ℓ^* above p^* and below q^*, or ℓ^* is above q^* and below p^*. So let's put on our duality glasses: **Given a set S^* of n double-wedges, we want to find a point that lies in as many double-wedges in S^* as possible.**



Let L be the set of $2n$ bounding lines of these wedges (dual to the endpoints of segments in S). We can construct the arrangement of L in $O(n^2)$ time. Each edge e of the arrangement lies on the boundary of exactly one double-sedge in S^*. We direct every edge e in the arrangement as follows: If e lies on the upper boundary of its double wedge, direct e from right to left; otherwise, direct e from left to right. (We can distinguish these two cases in $O(1)$ time.)

For each face f in the arrangement, let $W(f)$ denote the number of double wedges that contain f. When f is the face bounded by the upper envelope, we have $W(f) = 0$. Let $\mathsf{above}(e)$ and $\mathsf{below}(e)$ respectively denote the faces just above and below any edge e in the line arrangement. If edge e is directed from right to left, we have $W(\mathsf{below}(e)) = W(\mathsf{above}(e)) + 1$; otherwise, we have $W(\mathsf{below}(e)) = W(\mathsf{above}(e)) - 1$. Thus, we can compute $W(f)$ for every face f in $O(n^2)$ time, by traversing the dual graph of the line arrangement. (Essentially we are treating the $2n$ wedges as curves and computing winding numbers.)

Finally, we return the line dual to any point in any face f that maximizes $W(f)$. The entire algorithm runs in $\boldsymbol{O(n^2)}$ ***time***. ■

Rubric: 10 points = 4 for dual problem formulation + 4 for Alexander numbering dual arrangement + 2 for running time

(b) Describe an analyze a *faster* algorithm for the special case where each segment in S is either horizontal or vertical.

Solution: We solve this problem by reducing it to two instances of the red-blue point separation problem, one for stabbing lines with positive slope, and the other for stabbing lines with negative slope. (Assuming general position, there are no horizontal nor vertical stabbing lines.)

There is a stabbing line with positive slope if and only if there is a line ℓ satisfying the following constraints:

- For every vertical segment s, the top endpoint of s is above ℓ, and the bottom endpoint is below ℓ.
- For every horizontal segment s, the left endpoint of s is below ℓ, and the right endpoint is above ℓ.

These $2n$ constraints define a linear program with two variables (slope and y-intercept) that is feasible if and only if there is a stabbing line with positive slope. We can solve this linear program (with ab arbitrary objective function) in $O(n)$ expected time using Seidel's algorithm.

A similar linear program finds any stabbing lines with negative slope in $O(n)$ expected time.

■

Rubric: 5 points.

3. In class we saw the classical *funnel* algorithm to compute shortest paths inside a triangulated simple polygons. How would you modify this algorithm to find shortest paths in a polygon with holes?

   (a) Describe and analyze an algorithm to compute the shortest path between two given points in the interior of a polygon with *one* hole. *[Hint: Which way does the path go around the hole?]*

   (b) Describe and analyze an algorithm to compute the shortest path between two given points in the interior of a polygon with *two* holes.

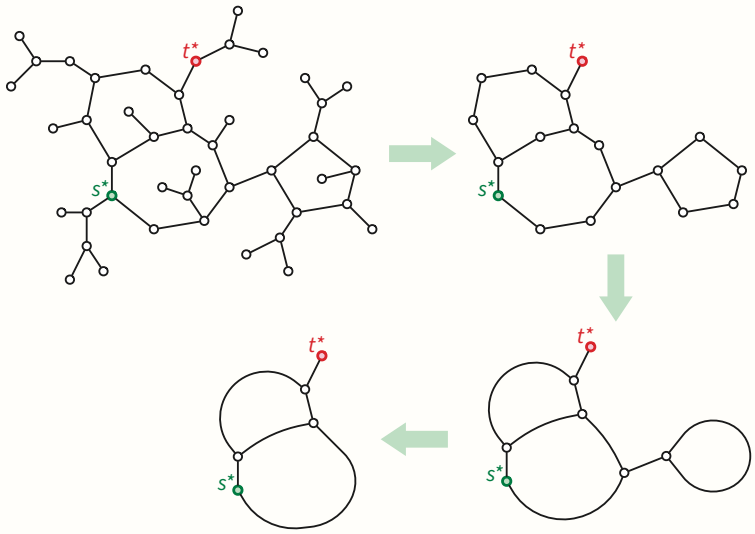
   (c) **[Extra credit]** Describe and analyze an algorithm to compute shortest paths in a polygon with h holes; analyze your algorithm as a function of both n (the total number of polygon vertices) and h (the number of holes).

   In all cases, you can assume that you are given a triangulation of the input polygon. For full credit, your algorithms should run in $O(n)$ time (for any constant h).

Solution: Suppose we are given a triangulation T of a polygon P with h holes, along with two points s and t in P. The shortest path from s to t crosses each diagonal in the triangulation at most once. Thus, the sequence of triangles that the shortest path intersects describes a simple path (meaning no repeated vertices or edges) in the dual graph T^*.

At a high level, our algorithm considers all simple paths in T^* with the correct endpoints; for each such path, we run the standard funnel algorithm to find the shortest path from s to t through the corresponding sleeve of triangles in T. The running time of the algorithm is $O(Nn)$, where N is the number of dual paths the algorithm considers. I will prove that $N = O(8^h)$.

To efficiently enumerate simple paths in T^*, we first *reduce* T^* as follows. Let s^* and t^* denote the nodes in T^* dual to the triangles containing s and t, respectively.



- First we repeatedly remove vertices of degree 1 *except* s^* *and* t^*, until no such vertices remain. This operation is sometimes called a *leaf reduction*.

- Then we repeatedly replace paths through degree-2 vertices *except* s^* *and* t^* with single edges, until no such paths remain. (his operation is sometimes called a *series reduction*. As we perform series reductions, we maintain the path in T^* corresponding to each edge the remaining graph.
- The final phase is optional: If the remaining graph contains bridge—an edge whose deletion disconnects the graph—that does *not* separate s and t, we delete the bridge and the component not containing s and t, and then (if possible) perform another series reduction at the remaining endpoint of the bridge.

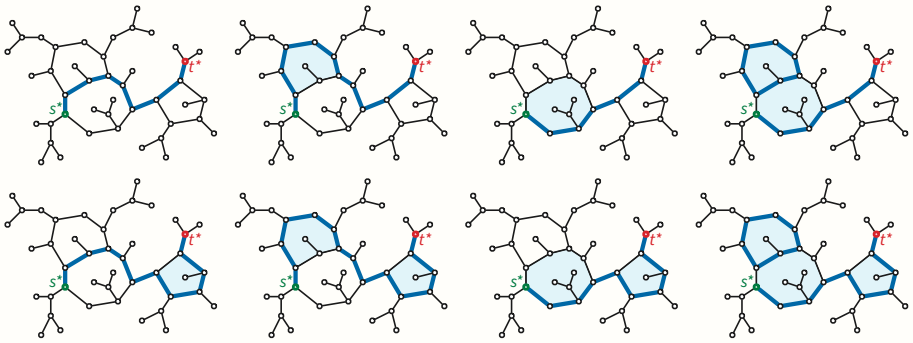
The entire reduction process can be performed in $O(n)$ time. Every simple path from s^* to t^* in the reduced dual graph R corresponds to a simple path from s^* to t^* in T^* and vice versa.

The reduced dual graph R is a planar graph with at most $h+1$ faces—one bounded face for each hole in P, plus the unbounded outer face. Every vertex has degree 3 except possible s^* and t^*, which might have degree 1 or 2. Routine calculations with Euler's formula now imply that R has at most $2h+2$ vertices and at most $3h+1$ edges.

Every simple path in R uses a subset of the edges; it follows immediately that R contains at most $2^{3h+1} = O(8^h)$ simple paths from s^* to t^*. Moreover, we can enumerate these paths in $O(8^h h)$ time by brute force: For every subset S of edges of R, test whether S is a simple path from s^* to t^* in $O(h)$ time.

For each simple path in R, we recover the corresponding path in T^* and run the funnel algorithm in the corresponding sleeve in T in $O(n)$ time. Finally, we return the shortest of all the funnel paths. The overall algorithm runs in $\boldsymbol{O(8^h n)}$ ***time***, which is $O(n)$ time for any constant h. ■

Solution: We can improve the previous solution by counting paths in T^* differently. The *symmetric difference* of two subgraphs of T^* is the subgraph of edges that appear in exactly one of those two subgraphs. The symmetric difference of any two paths from s^* to t^* in T^* is a subgraph of T^* in which every vertex has even degree; this subgraph must be the union of edge-disjoint simple cycles. Any union of disjoint cycles is the boundary of the union of a subset of the bounded faces of R, namely, the faces with odd depth/winding number. It follows that the number of simple paths from s^* to t^* is at most 2^h. (It should be easy to see that this bound is tight in the worst case, but not *always* tight.)



Eight potential paths from s^* to t^*. Only six of them are actually paths.

We can enumerate these paths by brute force in $O(2^h n)$ time as follows. For each subset of bounded faces, compute the boundary of its union, compute the symmetric difference of that boundary with some fixed path from s^* to t^*, and finally test whether that symmetric difference is connected (and therefore a simple path), all in $O(n)$ time. (Using the reduced graph R would reduce the running time of this part of the algorithm from $O(2^h n)$ to $O(2^h h)$.)

Finally, for each simple path from s^* to t^* in T^*, we can run the funnel algorithm in the corresponding sleeve in T in $O(n)$ time. The overall algorithm runs in ***$O(2^h n)$ time***, which is $O(n)$ time for any constant h. ■

Rubric: 15 points = 5 for each part. A correct algorithm for part (b) implies part (a), and a correct algorithm for part (c) implies parts (a) and (b). Any running time of the form $f(h) \cdot O(n)$ for part (c) is worth full credit; however, a proof of an explicit upper bound for $f(h)$ is required. This is not the fastest algorithm known for part (c); in particular, part (c) can be solved in $O(n \log n)$ time for any h!

4. A set $\mathcal{X}$ of objects in the plane is called *a family of pseudodisks* if (1) the boundary of each object in $\mathcal{X}$ is a simple closed curve, (2) the boundaries of any two objects in $\mathcal{X}$ either cross each other at exactly two points or do not intersect at all.

   A seminal result of Kedem, Livne, Pach, and Sharir,[1] which arises in the analysis of Minkowski sums, states that the union of any family of n pseudodisks consists of at most $O(n)$ boundary segments. This problem asks you to prove two special cases of this result.

   (a) Let $\mathcal{C}$ be a collection of n circular disks in the plane in general position, meaning no two disks are tangent. Prove that the boundary of the union of $\mathcal{C}$ consists of at most $O(n)$ circular arcs or complete circles. (Any set of circular disks in general position is clearly a family of pseudodisks.)

   Solution: Let $U = \bigcup \mathcal{C}$. Assuming general position, the boundary of U consists of disjoint simple closed curves. Trivially, at most n of these curves are complete circles. Each of the other closed curves consists of a cycle of circular arcs meeting at concave vertices. Thus, to solve the given problem, it suffices to show that the boundary of U has at most $O(n)$ vertices.

   Let $P(\mathcal{C})$ denote the power diagram of the boundary circles of disks in $\mathcal{C}$. $P(\mathcal{C})$ is a planar straight-line graph with at most n faces, and therefore, by Euler's formula, at most $3n - 6$ vertices.

   Each boundary vertex of U lies on the boundaries of two disks in $\mathcal{C}$ and outside every other disk in $\mathcal{C}$, and therefore lies on an edge of $P(\mathcal{C})$. Each edge e of $P(\mathcal{C})$ contains at most two boundary vertices of U, because the boundaries of the two disks that define e intersect in at most two points. We conclude that U has at most $6n - 12$ boundary vertices. ■

   Rubric: 5 points.

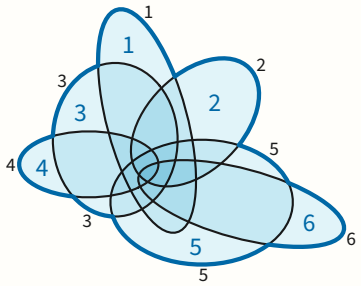
[1] Klara Kedem, Ron Livne, János Pach, and Micha Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.* 1(1):59–71, 1986.

(b) Let $\mathcal{X}$ be a family of n *convex* pseudodisks *that all contain a common point in their interiors*. Prove that the boundary of the union of $\mathcal{X}$ consists of at most $O(n)$ boundary segments. *[Hint: Convexity isn't actually necessary, but it simplifies the proof. First show that the union of $\mathcal{X}$ is star-shaped.]*

Solution: Let $U = \bigcup \mathcal{X}$, and let o be any point in $\bigcap \mathcal{X}$.

Any other point $p \in U$ must lie inside some pseudodisk $x \in \mathcal{X}$. Because X is convex, the entire segment op must lie in X, and therefore in U. It follows that U is star-shaped. In particular, the boundary of U is a simple closed curve, which consists of a cycle of boundary segments. Let $|\partial U|$ denote the number of these boundary segments.

Arbitrarily index the objects in $\mathcal{X}$ as $X_1, X_2, \ldots, X_n$. Label each boundary segment of U with the index i of the unique object X_i that contains it. Let L denote the resulting circular sequence of labels. For example, for the six pseudodisks shown below, we have $L = (13435652)$. We need to argue that L has length $O(n)$.



The label sequence L has two important combinatorial properties:

- Two consecutive labels in L must be different.
- L does not contain a subsequence of the form $i \ldots j \ldots i \ldots j$, where the same two labels i and j alternately appear twice. Otherwise, consider the two objects X_i and X_j in isolation. The boundary of $X_i \cup X_j$ must have at least two boundary arcs from X_i and two boundary arcs from X_j, which implies that the boundaries of X_i and X_j intersect at least four times, contradicting our assumption that our objects are pseudodisks.

Now I claim that some label appears in L exactly once. Let w be the shortest contiguous substring of L that starts and ends with the same label i. (If there is no such substring, then *every* label in L appears exactly once.)

- The substring w must have length at least 3; otherwise L would contain two i's in a row.
- Every label in w except i appears in w exactly once; otherwise, we could choose a shorter string w.
- Finally, except possibly i, no label in w appears in the complementary string $L \setminus w$; otherwise L would have an alternating subsequence.

We conclude that at least one label j appears in L exactly once.

We can now prove by induction that L has length at most $2n-1$, where n is the number of distinct labels in L. If no label appears more than once in L (in particular, if $n = 1$), then trivially $|L| \leq n \leq 2n-1$. Otherwise, let j be any label that appears exactly once in L. Remove j from L; if the same label appears immediately before and after j, remove one of those labels as well. The resulting string L' satisfies the same combinatorial conditions as L, but has only $n-1$ distinct labels. The inductive hypothesis implies that $|L'| \leq 2(n-1)-1$, which implies $|L| \leq |L'|+2 \leq 2n-1$, as required. ■

Rubric: 5 points.