Solutions to Homework 4

Solution 1: We remove the trapezoids C, D, E, G (highlighted in blue) and replace them with the trapezoids $H, I, \ldots, N$ (see Fig. 1). In the data structure, we replace the removed leaf nodes C and G with a 2-internal-node structure consisting of a segment endpoint (p_3 and t_3, respectively) and its incident edge s_3. We replace leaf nodes D and E with a 1-internal-node structure consisting only of s_3. The leaf nodes J and L are accessible through two distinct paths, and so they are shared.

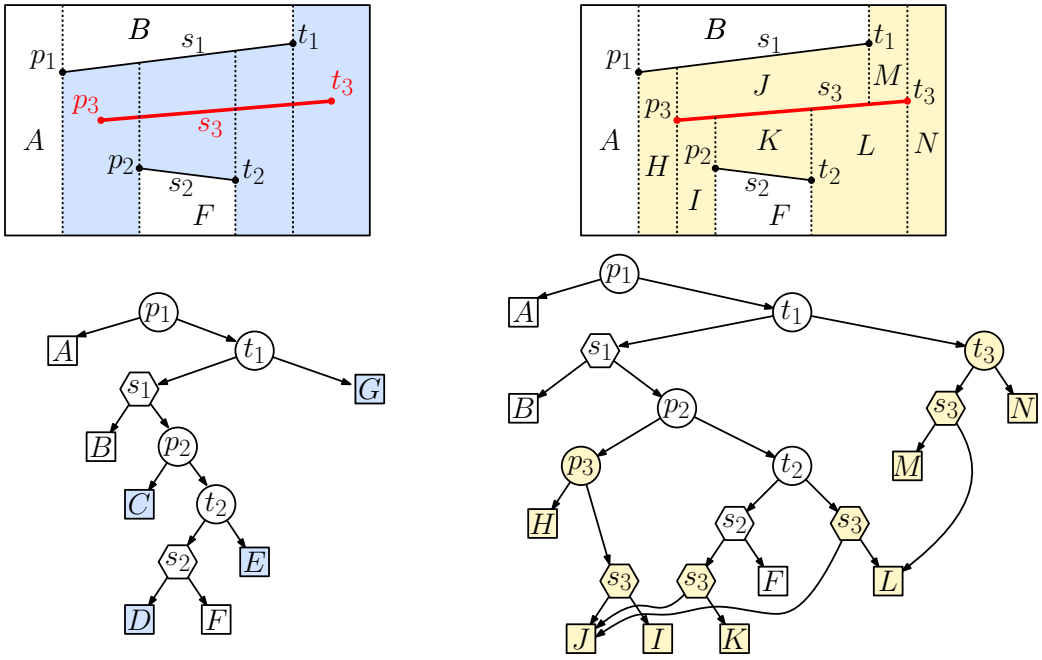


Figure 1: Trapezoidal map insertion and point location data structure.

Solution 2: Given the shot description (c, u_x, u_y), let $s = u_y/u_x$ denote the slope of the line carrying the ball. The ray's line equation is $\ell : y = sx + c$. Through point-line duality, we can treat this as a point $\ell^* = (s, -c)$ in the dual plane.

We will begin with two simplifications, which will not alter the asymptotic space or time requirements.

- We ignore the enclosing rectangle. We imagine that the lower segments extend to $-\infty$ and the upper segments extend to $+\infty$. First, we determine which (infinite) segment is hit. Then, in $O(1)$ additional time, we'll determine whether this intersection occurs within the enclosing rectangle. If not, we'll return the first point where the shot exits the rectangle (see Fig. 2).
- We only consider collisions with A. We can determine collisions with B symmetrically, and we return the closer of the two.

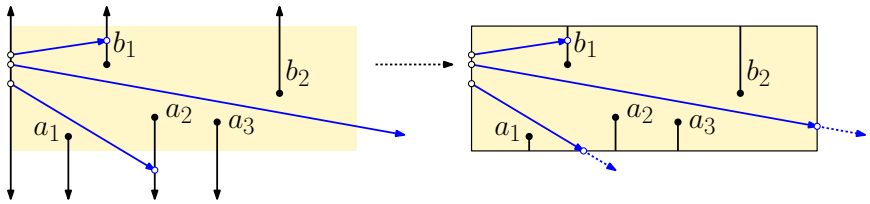


Figure 2: Ray-shooting queries. (Not geometrically accurate.)

We focus on the endpoints a_i of the vertical segments. In the primal plane, we seek the first point a_i (by x-coordinate) such that the line ℓ passes below it (see Fig. 3(a)). By the order-reversing property, this is equivalent to finding the first dual line a_i^* (based on slope) that the dual point ℓ^* lies above. Since the m dual lines intersect, producing $O(n^2)$ intersection points, we must keep the data structure size linear in m.

Assume the points of $A = \langle a_1, \ldots, a_m \rangle$ are sorted left to right. (The time needed to do this can be absorbed into the preprocessing time for the trapezoidal map.) Let's first consider the line a_1^*. If ℓ^* lies above this line, we are done. We label the region above a_1^* with the index 1. Otherwise, consider the portion of line a_2^* that lies below a_1^*. Again, if ℓ^* lies above this line, we are done. We label the region above a_2^* with the index 2 (see Fig. 3(b)). Continuing inductively, if ℓ^* lies above the lower envelope of $a_1^*, \ldots, a_{i-1}^*$, we already know the answer to the query. Otherwise, consider the portion of the line a_i^* that lies below this lower envelope. The portion of the lower envelope that lies above this line is labeled with index i. It follows that, on termination, every region has been properly labeled with the descending segment that the ray first hits (see Fig. 3(c)). The lower envelope corresponds to a ray that passes above all the points of A, and so is labeled "miss".

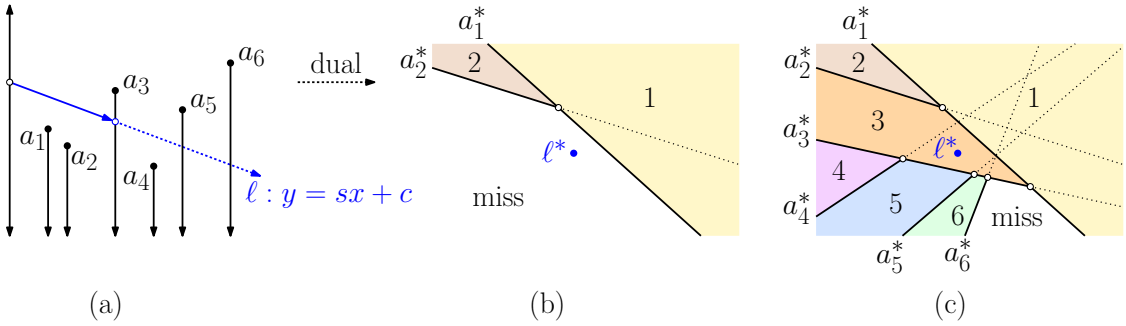


Figure 3: Ray-shooting queries in the dual. (Not geometrically accurate.)

Each line is trimmed when it first hits another line, resulting in n non-intersecting segments. (This is not entirely accurate, since the right endpoint of one segment may intersect the interior of another. If that bothers you, you can easily split the second segment at this intersection point. This creates $n-1$ new vertices, and hence the total number of segments is at most $2n-1$.) Now, we construct the trapezoidal map and associated point-location data structure for the resulting set of segments. The region containing ℓ^* is labeled with the first

The resulting data structure has one segment for each segment of A, for an expected size of $O(n)$. Queries are answered in expected query time $O(\log n)$. We compute this structure for both sets A and B. We determine the first segment of either that is hit by the ray in time

$O(\log n + \log m) = O(\log(n + m))$. Finally, if this intersection point occurs outside the box, we return the last point of intersection with the box in $O(1)$ time.

Solution 3:

(a) We will compute the endpoints of the circular arcs that define the accessible surface of the point set P, and we assume that the length of each arc can be computed in constant time. We begin by computing the Voronoi diagram of the point set P in $O(n \log n)$ time (e.g., by Fortune's algorithm). We assert that two circular arcs on the boundary of $M(P)$ intersect if and only if q lies on an edge of the Voronoi diagram, and q is at unit distance from the two sites bisected by this edge (see Fig. 4(a)). This is because q is at unit distance from the associated sites, and no other site can be closer (otherwise q would lie in the interior of some other circle and would not be on the boundary of $M(P)$).

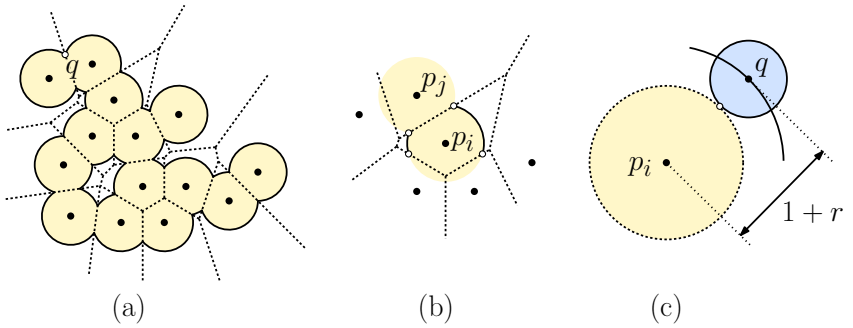


Figure 4: Accessible surface length.

To compute these endpoints, for each Voronoi cell $V(p_i)$ with $p_i \in P$, we walk counterclockwise around its boundary. Each edge intersects the unit ball b_i in at most two points. Thus, we can identify all these intersection points in time proportional to the number of edges incident to $V(p_i)$ (see Fig. 4(b)). After identifying them, we connect alternating pairs with circular arcs (centered at p_i). We compute the lengths of all circular arcs for $V(p_i)$, for $1 \leq i \leq n$, and sum these lengths. Because the diagram has a total of $O(n)$ edges, and each edge can be processed in $O(1)$ time, the total running time is $O(n)$ after computing the Voronoi diagram.

(b) The only difference with part (a) is that we use the radius $1 + r$ instead of 1. To see why, consider a point q that is the center of some water molecule that "kisses" the boundary of a unit disk centered at some point $p_i \in P$ (see Fig. 4(c)). Because the water molecule ball is tangent to the molecule ball, we have $\|q - p_i\| = 1 + r$. Thus, the solvent-accessible surface is the same as the accessible surface of a collection of balls of radius $1 + r$.

Solution 4: Recall that the edge e is vertical. Through an appropriate translation, we may assume that e lies on the y-axis. Letting p_1 and p_2 denote the upper and lower endpoints of e, their duals, p_1^* and p_2^*, are horizontal lines. By the order-reversing property of the dual transform, the set of lines passing through e dualizes to the set of points in the horizontal strip bounded by these two lines.

To answer the query, we first dualize the line ℓ passing through r to a point in the dual plane. Because we know that the ray must start on e and enter into P, each possible query is mapped to

a unique dual point in this way. We will be able to produce a subdivision of the strip into $O(n)$ polygonal regions, such that the region containing the dual query point ℓ^* uniquely determines the edge that is hit by the query ray. Thus, the answer to the ray shooting query can be solved by point location.

We first triangulate P. Consider the other vertex p_i of the triangle incident to edge e (p_3 in Fig. 5). Partition the strip by inserting the dual line p_i^*. This subdivides the strip into two convex polygonal regions. The region lying above p_i^* corresponds to rays that pass below p_i, and vice versa. We recursively split these two regions by applying the same process. If the edge below p_i is an edge e of P (i.e., e_2 in Fig. 5), we label this region with e and stop the recursion. Otherwise, we consider the vertex p_j that is incident to the neighboring triangle. We split this region using the dual line p_j^*, into two subregions. The subregion lying above p_j^* corresponds to rays passing below p_j and vice versa.

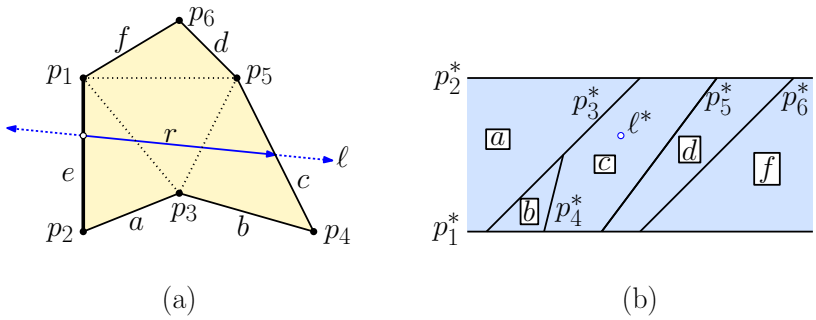


Figure 5: Ray-shooting queries.

The final result subdivides the strip into at most n convex polygonal regions. (At most n, because it is easy to see that each edge of P can contribute at most one region to the subdivision.) Since there are n faces and no vertices of degree two, we can use Euler's formula to show that the subdivision has $O(n)$ vertices. Thus, we can build a point-location data structure of size $O(n)$ to answer ray-shooting queries in $O(\log n)$ time.

Solution to the Challenge Problem:

(a) Recall that the arcs of the beach line are parabolic arcs. We claim that any two parabolas intersect in at most two points. To see this, let $y = P(x)$ and $y = Q(x)$ be the equations for two parabolas. The x-coordinates of their common intersection are the roots of $P(x) - Q(x) = 0$. This is a quadratic equation in x, and hence it has at most two real roots.

If such an alternation were to occur, then by the continuity of parabolas, between each pair of alternating arcs there must an intersection point. Three alternations would imply the existence of three intersection points, a contradiction.

(b) Let $L(n)$ denote the maximum length of a sequence of n distinct characters that satisfy our constraints. We prove that $L(n) \leq 2n-1$ by induction on n. This is easily verified for $n = 1$.

Assume the hypothesis holds for any number of distinct symbols strictly less than n. Let the symbols be $\{a_1, \ldots, a_n\}$. Consider the first and last occurrences of the symbol a_n. They subdivide the string into three strings as shown below

$$w_1 \; a_n \; u \; a_n \; w_2,$$

where w_1, u, and w_2 are strings, and w_1 and w_2 contain no occurrences of a_n. Let $w = w_1 w_2$. Observe that if symbol $a_i \neq a_n$ occurs in w, then it cannot occur in u, and vice versa, since otherwise we would have a forbidden alternation. Thus, if k denotes the number of distinct characters in u then there are $n-k$ distinct characters in w. Clearly there can be no forbidden alternations in u nor in w. Also, u cannot have any repeated symbols. It is possible to have a single repeated symbol in w (where w_1 and w_2 come together), so if we count this repeated character and the lengths of the strings u and w, the total number of symbols in the string is at most

$$L(n) \;\leq\; L(k) + L(n-k) + 1 \;\leq\; (2k-1) + (2(n-k)-1) + 1 \;\leq\; 2n-1,$$

completing the proof.