CMSC 754: Fall 2021 Dave Mount

Homework 1: Hulls and Plane Sweep

Handed out Thursday, Sep 16. Due by the start of class on Thursday, Sep 23. (Submissions will be through Gradescope. Submission information will be forthcoming.) Late homeworks are not accepted (unless an extension has been prearranged) so please turn in whatever you have completed by the due date. Unless otherwise specified, you may assume that all inputs are given in *general position.*

Problem 1. Present an algorithm which, given a sequence $P = \langle p_1, \ldots, p_n \rangle$ of $n \geq 3$ points in $\mathbb{R}^2$, determines whether these points constitute the (distinct) sequence of vertices of a convex polygon given either in clockwise or counterclockwise order. The output of your algorithm is either "CW" (for a valid clockwise sequence), "CCW" (for a valid counterclockwise sequence), or "invalid".

Each point p_i is given by its coordinates (x_i, y_i). For full credit, your algorithm should run in $O(n)$ time and should only involve orientation test between the points of P. (You can receive 2/3 credit if you give a correct answer that involves other *exact* computations, such as comparing two coordinates and/or orientation tests involving points that are not part of the input set.)

For this problem, you may *not* assume general position. For example, three or more points might be collinear, and there might be duplicate coordinate values.

Prove that your algorithm is correct and justify its running time. (If you are unsure whether you need to prove a geometric assertion, feel free to check with me.)

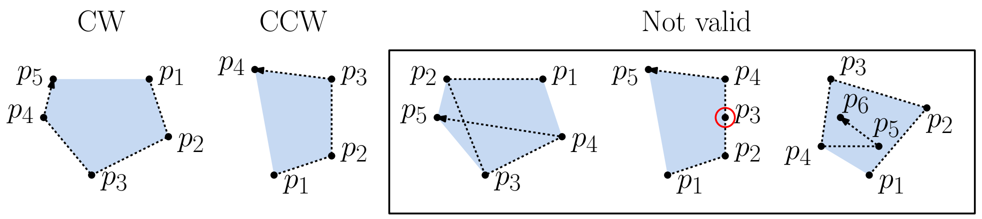

Figure 1: Problem 1: Vertices of a convex polygon?

Problem 2. A popular programming exercise is called the *skyline problem.* You are given a set of n axis-aligned rectangles, all having their bottom edge on the x-axis. The problem is to compute the union of these rectangles, called the *skyline.*

The input is presented as a set of n triples (l_i, r_i, h_i), where l_i is x-coordinate of the ith rectangle's left side, r_i is the x-coordinate of its right side, and h_i is its height (see Fig. 2). You may assume that all these values are positive reals.

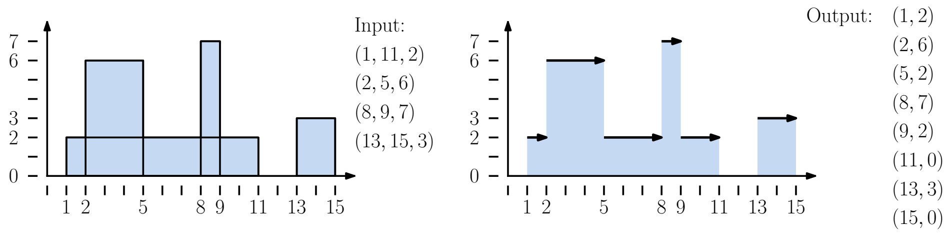

Figure 2: Problem 2: Skyline problem.

The output consists of a *run-length encoding* of the horizontal edges of the boundary of the union of these rectangles. This consists of a sequence of the form $\langle (x_1, h_1), (x_2, h_2), \ldots, (x_m, h_m) \rangle$, where x_j is the start of the jth horizontal edge and h_j is its distance above the x-axis (see Fig. 2). The x values should increase monotonically ($x_j < x_{j+1}$), and consecutive heights should be distinct ($h_j \neq h_{j+1}$). Each horizontal edge runs between x_j and x_{j+1} at $y = h_j$. The first height value h_1 must be nonzero, and the last height value h_m must be zero.

Present an output sensitive algorithm for the skyline problem. For full credit, your algorithm should run in time $O(n \log m)$, where m is the length of the output. (Note that $O(n \log n)$ is easy, and there are many answers on the internet, so you won't get much credit for such an algorithm.)

As always, first explain your strategy, present your algorithm (in English or pseudocode), prove its correctness, and derive its running time. The emphasis here is on getting the desired running time.

Hint: To simplify your description, you may assume that you have access to the following utility function for answering *horizontal ray-shooting queries.* (It's implementation will be left as an exercise.) Given a set of k vertical line segments, each with its lower endpoint on the x-axis, and given a query point $q = (q_x, q_y)$, find the first segment (if any) that is hit by a horizontal ray shot to the right from q. (If not segment is hit, the query returns some special value, like "no-hit".) You may assume that, given any such set of segments in $O(k \log k)$ time it is possible to build a data structure that can answer such queries in time $O(\log k)$.

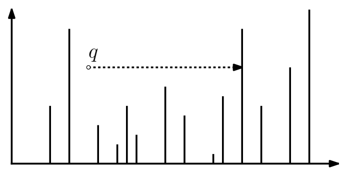

Figure 3: Horizontal ray shooting.

Problem 3. You have m robots that move along a common 1-dimensional track that runs north and south. Each robot provides you with its *motion plan*, which consists of a sequence of k *motion segments.* Each motion segment is given by a pair of reals (Δ, v), where $\Delta > 0$ denotes the duration of movement, and v is the robot's speed during this time interval. A positive speed indicates movement to the north and a negative speed indicates movement to the south. For example, the segment $(5, -13)$ means that the robot moves at 13 units per second to the south for 5 seconds. During this segment, the robot has moved a total of $5 \cdot 13 = 65$ units of distance to the south.

The full motion of the ith robot is specified as a sequence of k such segments, $\langle(\Delta_{i,1}, v_{i,1}), \ldots, (\Delta_{i,k}, v_{i,k})\rangle$. For example, for $k = 3$, given the sequence $\langle(2, 5), (3, -2), (1, 4)\rangle$, this robot will move at velocity 5 for 2 seconds (moving 10 units north), then at velocity -2 for 3 seconds (moving 6 units to the south), and finally at velocity 4 for 1 second (moving 4 units north). Thus, over the span of 6 seconds, this robot has moved a net distance of $10 - 6 + 4 = 8$ units north.

The input to your program consists of the starting positions $\langle z_1, \ldots, z_m\rangle$ of the m robots along the track, together with m motion plans, one for each robot, each of length k. Once a robot reaches the end of its motion plan, it remains at its final position until all the robots have stopped moving. Let $n = km$ denote the entire size of the input.

Given this input, provide answers to the following questions:

(a) We say that two robots *collide* if they occupy the same location at the same time. As a function of k and m, what is the maximum number of collisions that there might be among a collection of m robots with k-element plans? Justify your answer. (For full credit, give an exact bound. Partial credit will be given for an asymptotic bound.)

(b) Present an efficient algorithm which, given the starting positions and motion plans, determines whether any of the robots ever collide. If there is a collision, your algorithm should output the first time that any two robots collide and indicate the indices of these two robots. If there are no collisions, indicate the closest that any two robots ever get to one another in the course of the entire motion process. For full credit, your algorithm should run in time $O(n \log n)$.

(c) Suppose that when robots collide, they simply pass through each other. Present an efficient algorithm which, given the starting positions and motion plans, reports all the collisions. Letting c denote the total number of collisions, your algorithm should run in time $O((c + n) \log n)$.

In all three cases, justify your algorithm's correctness and derive its running time.

Problem 4. Given a simple polygon P with n vertices, recall that the addition of any diagonal (an internal line segment joining two visible vertices of P) splits P into two simple polygons with n_1 and n_2 vertices respectively, where $n_1 + n_2 = n + 2$.

(a) Show that given any simple polygon P with $n \geq 4$ there exists a diagonal that splits P such that $\min(n_1, n_2) \geq \lfloor n/3 \rfloor$. (Hint: It may help to consider the dual graph of any triangulation.)

(b) Show that the constant 1/3 is the best possible, in that for any $c > 1/3$, there exists a polygon such that *any* diagonal chosen results in a split such that $\min(n_1, n_2) < cn$. (You can provide a drawing, but it should be clear how your drawing can be generalized to all sufficiently large values of n.)

Challenge Problem 1: Present a solution to the horizontal ray-shooting problem stated in Problem 1. (Given a set of vertical line segments with their lower endpoints on the x-axis, determine the first segment hit by a horizontal ray shot to the right from any query point.)

Challenge Problem 2: Present an efficient algorithm which, given a set $P = \{p_1, \ldots, p_n\}$ of n points on the integer grid, computes the polygon of minimum perimeter that encloses these points, subject to the condition that the sides of this enclosure can be horizontal, vertical, or sloped at $\pm 45°$ (see Fig. 4). (Hint: $O(n)$ time is possible, but $O(n \log n)$ time acceptable. Prove that the enclosure produced by your algorithm has the minimum perimeter. Note that there may generally be many enclosures with the same perimeter, and your algorithm may output any of them.)

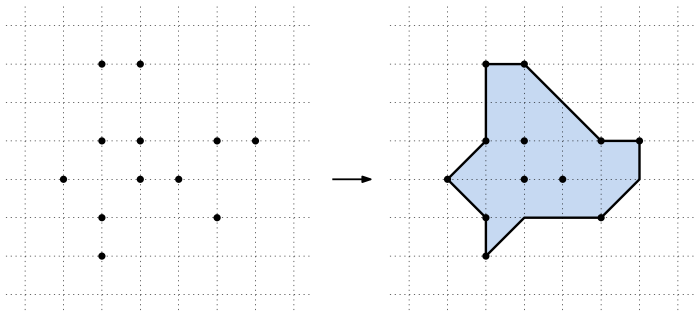

Figure 4: Minimum perimeter enclosure.

Some tips about writing algorithms: Throughout the semester, whenever you are asked to present an "algorithm," you should present three things: the algorithm, an informal proof of its correctness, and a derivation of its running time. Remember that your description is intended to be read by a human, not a compiler, so conciseness and clarity are preferred over technical details. Unless otherwise stated, you may use any results from class, or results from any standard textbook on algorithms and data structures. (If the source is from outside of class, you must cite your sources.) Also, you may use results from geometry that: (1) have been mentioned in class, (2) would be known to someone who knows basic geometry or linear algebra, or (3) is intuitively obvious. If you are unsure, please feel free to check with me.

Giving careful and rigorous proofs can be quite cumbersome in geometry, and so you are encouraged to use intuition and give illustrations whenever appropriate. Beware, however, that a poorly drawn figure can make certain erroneous hypotheses appear to be "obviously correct."

Throughout the semester, unless otherwise stated, you may assume that input objects are in *general position*. For example, you may assume that no two points have the same x-coordinate, no three points are collinear, no four points are cocircular. Also, unless otherwise stated, you may assume that any geometric primitive involving a constant number of objects each of constant complexity can be computed in $O(1)$ time.