CMSC 754: Fall 2026 Dave Mount

Homework 2

Due by the start of class on Tuesday, Sep 22. (Submissions will be through Gradescope.) 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. Throughout this problem, assume general position, and in particular, whenever two edges intersect, they intersect in a single point.

(a) Consider two integers n and m, both *even*, where $n \geq 3$ and $m \geq 3$. Prove that there exist two simple polygons P_n and P_m, having n and m sides, respectively, such that their boundaries intersect nm times, and, as a function of n and m, this is maximum number of boundary intersections possible. (For the lower bound, a clear drawing is sufficient. Be sure that your drawing is "generic", meaning that it is clear that it can be generalized it to arbitrarily large values of n and m.)

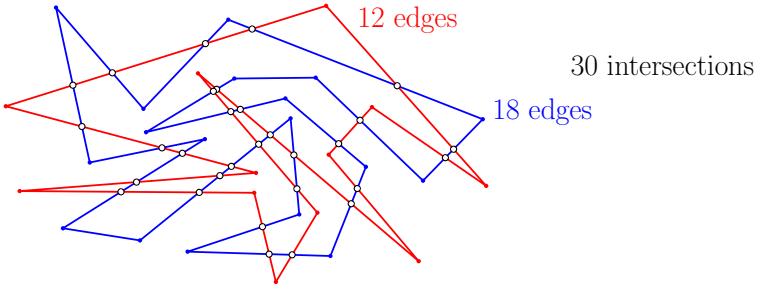

Figure 1: Number of boundary intersections between simple polygons.

(b) Consider problem (a) for the case where n is *even* and m is *odd*. Prove that the number of boundary intersections cannot exceed $n(m-1)$, and present a generic example showing that this is achievable.

(c) Consider problem (a) for the case where both n and m are both *odd*. Present a generic example to show the boundaries may intersect up to $(n-1)(m-1)+2$ times. (Do not expect to be able to prove that the bound is tight. This seems to be an very hard problem, and the current best bound is $(n-1)(m-1)+C$, where $C \leq 2^{2^{67}}$.)

Problem 2. You are given two vertical lines at $x = 0$ and $x = 1$. You are also given two sets of line segments, R and B (for *red* and *blue*) of sizes n and m, respectively. The red segments are pairwise disjoint and the blue segments are pairwise disjoint, but there may be intersections between red segments and blue segments (see Fig. 2).

Devise an efficient algorithm that *counts* all the red-blue intersections. Your algorithm should run in $O((n+m)\log(n+m))$ time, irrespective of the number of intersections (which could be as large as nm).

Hint: Do *not* use plane-sweep. This can all be done based on the relative sorted ordering of points along the left and right vertical sides.

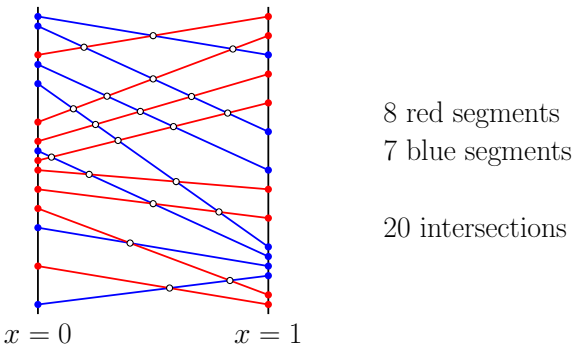

Figure 2: Red-blue intersection counting.

Problem 3. Consider a simple polygon P in $\mathbb{R}^2$, and let s denote its leftmost vertex.

(a) Present an efficient algorithm that outputs the vertex t with the maximum x-coordinate that is reachable from s along an x-monotone path (see Figure 3(a)). Your algorithm need not output the path, just the vertex t. (But see the Challenge Problem.)

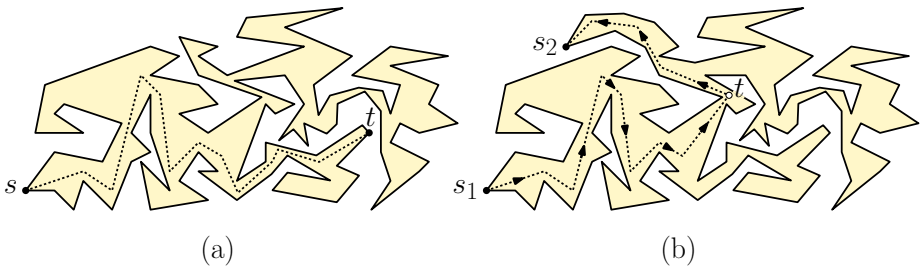

Figure 3: Monotone paths.

(b) Suppose that you are given two starting vertices s_1 and s_2. Present an efficient algorithm that determines whether there exists a point t in the polygon, such that there is an x-monotone path from s_1 to t and a reverse x-monotone (that is, monotonically decreasing) from t to s_2 (see Figure 3(b)). If there is such a vertex t, output such a vertex. If not, indicate this.

You may assume that s_1 and s_2 are "start vertices" in the sense defined in the lecture on polygon triangulation. That is, the interior of P lies locally to the right of each vertex.

Hint: Use plane-sweep. $O(n \log n)$ time and $O(n)$ space is achievable, but partial credit will be given for any polynomial time solution.

Problem 4. (Optional - Ungraded) A friend of yours from the civil engineering department wants to analyze whether a dangerous portion of a river will flood. He presents you with the following model of the river. The portion of the river of interest is modeled as an x-monotone polygon P that is bounded between two vertical lines at $x = x^-$ and $x = x^+$ (see Fig. 4). The river is bounded on its left and right ends by two vertical line segments of lengths w^- and w^+, respectively. Inside the polygon are some number of disjoint x-monotone polygons

that represent islands in the river. Let n denote the total number of vertices, including both the outer banks of the river and the islands.

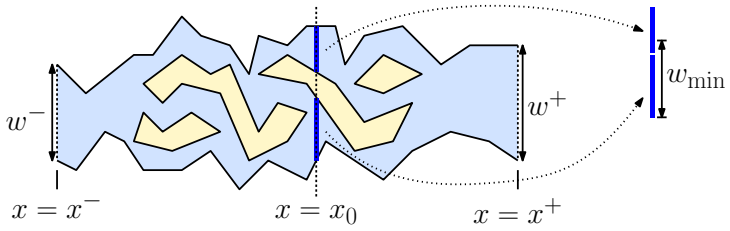

Figure 4: River flooding

Your friend tells you that the in order to avoid a flood, the width of the river (not counting islands) at *every* vertical cut must be at least some minimum value $w_{\min}$. For example, in the figure, the sum of the two blue vertical segments at $x = x_0$ must be at least $w_{\min}$ in order to avoid a flood.

Given the polygon P and the value $w_{\min}$, present an $O(n \log n)$ time algorithm that determines whether the river will flood, that is, whether there is a vertical cut whose total width is smaller than $w_{\min}$. If it will flood, your algorithm should output the value x_0 of the *bottleneck*, that is, the location where the sum of vertical lengths (excluding islands) is the smallest.

Hint 1: There is an (uncountably) infinite number of possible vertical cuts to consider. Prove that it suffices to check the width at a discrete set of locations, whose number is $O(n)$.

Hint 2: There is a bit of a trick to updating the vertical widths (excluding the islands). For partial credit, explain how to do it under the assumption that the sweep line can only intersect a constant number of islands at any time. (For example, in the figure, the sweep line never hits more than two islands at a time.) For full credit, explain how to do it even if the number of islands hit by the sweep line at any time could be as high as $\Omega(n)$.

Note: Challenge problems are not graded as part of the homework. The grades are recorded separately. After final grades have been computed, I may "bump-up" a grade that is slightly below a cutoff threshold based on these extra points. (But there is no formal rule for this.)

Challenge Problem 1: Augment your algorithm from Problem 3(a) to output the monotone path from s to t.

Challenge Problem 2: The following problem takes place on a square integer grid in the plane. You are given a box of height h and width w. A number of squares in the box contain mirrors that are angled at $\pm 45°$ (see Figure 5(a)). A laser beam enters the box horizontally in the top-left corner, and it is reflected whenever it hits a mirror (see Figure 5(b)). The experiment is considered *successful* if the laser beam eventually exits horizontally through the lower right corner, and otherwise it is a *failure*.

Given such a box of dimensions $h \times w$, and given the directions are orientations of n mirrors in the box, there are three possibilities:

(i) The laser shot from s successfully arrives at t, following the existing set of mirrors.

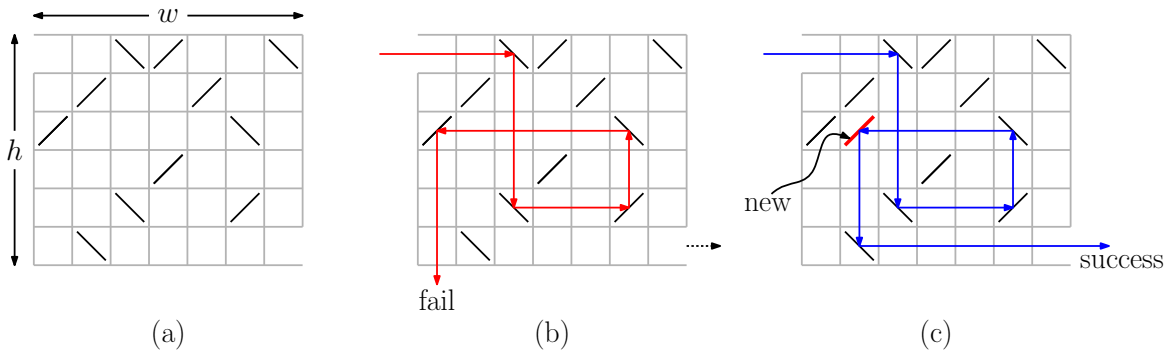

Figure 5: (a) Shooting a laser beam (unsuccessful) and (b) adding a mirror to make it successful.

(ii) (i) does not hold, but it is possible to insert a *single* new mirror within the box so that the laser beam now hits t (see Figure 5(b)).

(iii) (i) does not hold, but it is *not* possible to reach t by the insertion of a *single* mirror.

Let's assume that case-(i) does not occur. Design an efficient algorithm to determine whether (ii) or (iii) applies. If (ii) applies, determine the square and the orientation of a mirror that leads to success. (There may be many, and your algorithm can return any of them.)

As an aid to your algorithm, you may assume that you have access to a data structure which, given any horizontal or vertical ray, determines the first mirror that the ray strikes in $O(\log n)$ time. Given this data structure, there is a solution that runs in $O(n \log n)$ time.