CMSC 754: Spring 2026 Dave Mount

Solutions to Quiz 1

Solution 1:

(a) Diagonals: $n-3$. Triangles: $n-2$. (This was stated in Lecture 5.)

(b) Min: $\lceil k/2 \rceil$ Max: k. Each diagonal can resolve at least one and up to two scan reflex vertices, one for each endpoint. (We will also give credit to $n-3$ as an answer to the "max" part, since a triangulation is a decomposition into monotone pieces, and it maximizes the number of diagonals.)

(c) (This was covered in Lecture 3.)

(i) The first part of Chan's algorithm (Graham's scan) runs in $O(n \log h^*)$ time. If h^* was *extremely large* compared to the true value h (so that $\log h^*$ is asymptotically larger than $\log h$), then this part of the algorithm could take too long.

(ii) There are two possible problems. Either we are true to Chan's original description and stop the Jarvis-march phase algorithm as soon as we see more than h^* points on the hull. In this case, the algorithm terminates before producing the full convex hull. On the other hand, if we run the Jarvis-march phase until the entire hull is finished, this runs in time $O(h \lceil n/h^* \rceil \log h^*)$. If h^* is very small, then h/h^* is asymptotically larger than a constant, and the running time will exceed the allowed $O(n \log h)$.

Solution 2:

(a) The linear extension of $\overline{ab}$ intersects $\overline{st}$ (without the segments intersecting) if an only if s and t lie on opposite sides of the line $\overleftrightarrow{ab}$ but a and b lie on the same side of the line $\overleftrightarrow{st}$, and therefore we have

$$(\text{orient}(a,b,s) \neq \text{orient}(a,b,t)) \quad \text{and} \quad (\text{orient}(s,t,a) = \text{orient}(s,t,b))$$

(b) There are a few ways to test whether the ray $\overrightarrow{ab}$ intersects $\overline{st}$ (without the segments intersecting). One approach is to observe that this is equivalent to the condition that b lies within the triangle $\triangle ast$, which (from Homework 1) can be tested with

$$\text{orient}(a,s,b) \;=\; \text{orient}(s,t,b) \;=\; \text{orient}(t,a,b).$$

Another approach is to take the condition from (a), and combine it with the addition condition that both b and t lie on the same side of the line $\overleftrightarrow{as}$, that is, $\text{orient}(a,s,b) = \text{orient}(a,s,t)$.

Solution 3:

(a) Draw vertical lines through each of the $n+m+2$ vertices of the two polygonal curves (see Fig. 1(a)). This subdivides the plane into $n+m+1$ intervals. Excluding the first and last, each interval can contain at most a single intersection, so the total number of intersections is at most $n+m+1-2 = n+m-1$.

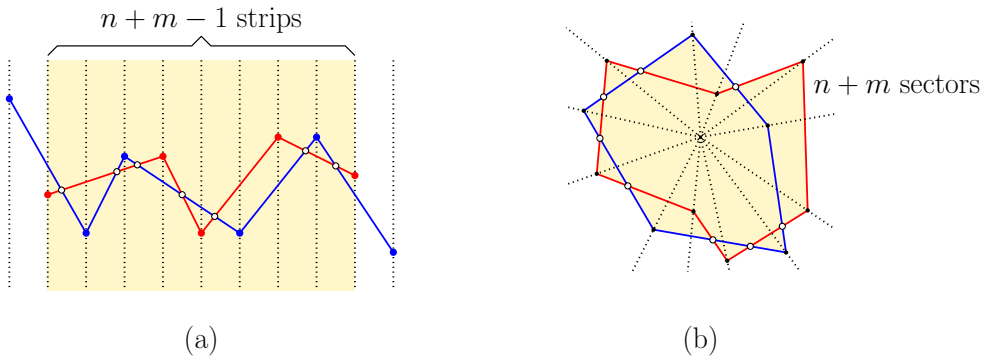

Figure 1: Number of intersections.

(b) Draw rays from the origin through the $n+m$ vertices of the polygonal curves (see Fig. 1(b)). This subdivides the plane into $n + m$ sectors. Each sector can contain at most a single intersection, so the total number of intersections is at most $n + m$. (Note that this is only achievable if $n+m$ is even, since the two polygons need to alternate between inner and outer at each sector boundary. Otherwise, you lose one possible intersection. A more accurate answer would be $n + m - ((n + m) \bmod 2)$.)

Solution 4: The algorithm uses a top-down plane-sweep approach. For any location of the horizontal sweep line $y = y_0$, the sweep-line status consists of all the line segments that intersect this line, sorted from left to right based on their intersection points. As with line-segment intersection, we do not store the actual x-coordinates of the intersections, but rather the line equations so we can derive this information.

As the algorithm proceeds, it can be in one of two modes. We say the drop is *falling* when it is falling along some vertical line, and we say it is *sliding*, when it is traveling along a segment (see Fig. 2(a)). When the drop is falling, we store the x-coordinate of the water-drop line in the sweep-line status. We check for intersections between this vertical line and the segments that lie immediately to its left and right (predecessor and successor in the sweep-line status). When the drop is sliding, we record which segment that it is currently sliding along.

Since the segments do not intersect, the only intersection events we need to consider are between the (falling) vertical rain drop and the two segments on either side of it on the sweep line. Here are the main events:

Initialization: We start at the horizontal line through q, that is, $y = q_y$. For simplicity, let's assume that q is so far above all the segments that this horizontal line does not intersect any segments. We create an empty sweep-line status, except for the water drop at $x = q_x$ and we set the mode to *falling*.

Upper endpoint: On seeing the upper endpoint of any segment, we insert it into the sweep-line status. If the drop is *falling*, and this segment becomes the predecessor or successor of the fall line, we check whether we intersect this segment at a point higher than any prior intersection point, and if so we, create an intersection event and remove any prior intersection event (see Fig. 2(b)).

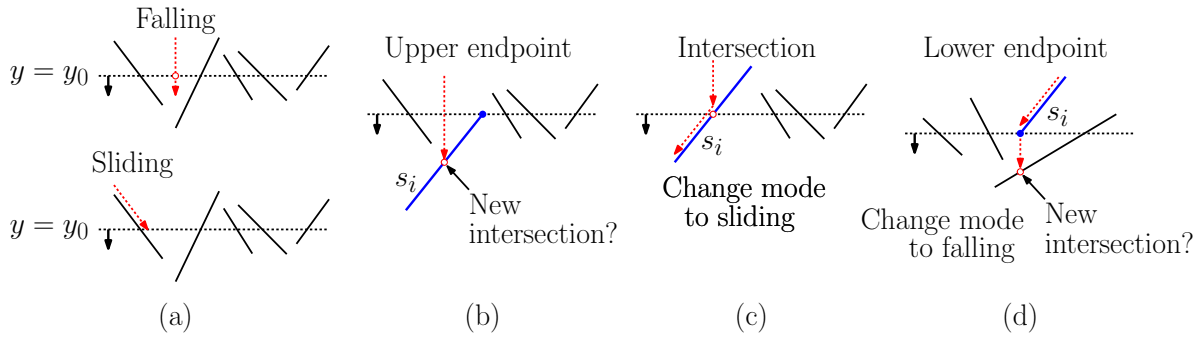

Figure 2: Water drop problem.

Water drop intersection: When the falling water drop hits some segment, we set the mode to *sliding*, and save this segment as the one carrying the water drop (see Fig. 2(c)).

Segment lower endpoint: First, we remove this segment from the sweep-line status. If the current mode is *sliding* and this is the segment that currently carries the water drop, we switch the mode to *falling* at the x-coordinate of this endpoint (see Fig. 2(d)). As before, we need to determine the predecessor and successor of the fall line in the sweep line status, and check each for possible new intersection events.

Otherwise, if the mode is *falling*, the removal of this line may have affected the immediate predecessor or successor of the falling line, and if so, we update and check for intersections.

End: On hitting the x-axis, the mode must be *falling*. We output the current water-drop line x-coordinate as the final answer.

In $O(n \log n)$ time we can sort the segment endpoints by y-coordinates. Each time we check for intersections with the falling water drop, we only consider its immediate predecessor and successor on the sweep line, which can be determined in $O(\log n)$ time. Since there are $O(n)$ events (endpoints and drop-segment intersections), and each event involves a constant number of dictionary operations, the overall running time is $O(n \log n)$.