Homework 2

Sam Windham *with Lucio Franco*

10/17/17

1.

```
PROCEDURE LeftTangent(Polygon p, Point q):
    // first point is left tangent
    if isLeft(q, p[n-1], p[0]) AND !isLeft(q, p[0], p[1]):
        return (q, p[0])

    let i = 0, offset = 1
    loop:
        // direction of current edge
        let edgeDir = isLeft(q, p[i], p[i+1])

        // current edge is the left tangent point
        if isLeft(q, p[i-1], p[i]) AND !edgeDir:
            return (q, p[i])

        // check directions and continue search

        let prevEdgeDir = isLeft(q, p[i - offset], p[i - offset + 1])

        // current edge is left
        if edgeDir:
            // previous edge is left AND current point left of previous point
            if prevEdgeDir AND !isLeft(q, p[i-offset], p[i]):
                i -= offset
                offset = 1
            // prev edge is right OR current edge left of previous edge
            else:
                offset *= 2
        else:
            // previous edge is left
            if prevEdgeDir:
                i -= offset
                offset = 1
            else:
                offset *= 2

        i = min(i + offset, n - 1)
```

The algorithm performs an exponential search on the points of P and returns the line segment between q and the tangent point. The `isLeft(a, b, c)` function returns the polarity of `cross(<a,b>, <b,c>)`. Exponential search runs in O(log n).

3.

It is possible to report all positive slope lines using a line-sweep in the dual plane. Each point can be transformed to the dual plane as lines. These lines will produce a total of $\binom{n}{2}$ intersections (or the induced lines in the primal). We want to report the total number of these dual-plane intersections that have x-coordinates greater than zero (which will be all induce lines with positive slope).

(a.) Starting at $x = 0$ in the dual plane, we add all lines to the status. Iterating from top to bottom on this status, we can check each pair of lines. Two adjacent lines, top with positive slope, and bottom with negative slope can be ignored, since they will not intersect to the right of the status position. All other adjacent pairs need to be checked and intersections added to the event queue. At each event, swap lines and check against their neighbors.

(b.) Do the same as **a.**, but start at $x = m^{'}$ and stop at $x = m^{"}$ when sweeping through the dual plane.

So for n points, we have k induced lines with positive slope (or k intersections in the dual with $x > 0$). Each event takes O(log$n$) using a balanced binary tree. Initially adding all the points to the status takes O($n$log$n$). Therefore our total runtime will be O($n$log$n$ + $k$log$n$) = O(($n + k$)log$n$). This is an improvement on the naive brute-force method, checking all $\binom{n}{2}$ pairs in $\theta(n^2)$.

4.

Visible points can be reported using a rotating sweep line. Each endpoint lies on a ray starting from p at an angle of α. Let such a ray be denoted as $R(\alpha)$.

Sort each endpoint on α to form a schedule (ties split on distance from p). Starting with the first endpoint in the schedule, add the corresponding segment to a min-heap ordered on distance from p. At each event, a segment is either added or removed from the heap since a line segment either begins or ends at endpoints. If a segment has been added already, it gets removed. Since the segments are disjoint (do not cross), we can assume that the position in the heap will not change between events.

At the end of each heap operation, the minimum element is peeked at and reported as visible. This is the closest element to p at this point in the schedule and therefore visible from p.

There are a total of $2n$ events in the schedule, since each segment has two endpoints. At each event, a heap insertion or deletion is performed in O(log$n$) and a minimum is peeked at O(1). Therefore, the algorithm requires O($n$log$n$) for initially sorting the endpoints, O($n$log$n$) for processing each event in the schedule, and O($n$log$n$) to sort and remove duplicate reports from the result, giving us a total runtime of O($n$log$n$).

5.

Note:

This problem can be solved similarly to the Widest Empty Corridor problem. The Widest Empty Corridor problem searches for the two parallel lines that have no points between them with the maximum orthogonal distance d between the lines. Our solution would be the line parallel to these two lines and a distance $d/2$ from them.

Houle and Maciel gave a O(n^2)-time and O(n)-space algorithm for solving this problem. This consists of constructing an arrangement on S in O(n^2) and then processing each face f in the arrangement to construct a widest empty corridor. Processing each face f takes O($|f|$), therefore the time to process all faces is O(n^2) since the arrangement has O(n^2) vertices.

Finding our line will be fairly straight-forward, needing only to transform one of the widest corridor lines orthogonally by $d/2$.

Since these sort of problems require the construction of an arrangement of S, other solutions will require $O(n^2)$ as well.

Source: Widest-Corridor Problems - Janardan and Preparata, 1994

My approach:

Another approach would be to transform all n points of S to lines in the dual plane. We create an arrangement on these lines in order to create a schedule for the line sweep. Each event would be the intersection of dual-lines. At each event, we check the distance to the intersection's neighboring lines in the vertical direction. While doing the sweep, we keep track of the maximum distance and which three points are involved. At the end, the maximum distance will give us the three points that form a widest corridor.

A degenerate case would be that the widest corridor is perpendicular to the line-segment between two points and its distance being the Euclidean distance between those points. This case occurs when an intersection in the dual is either the maximum or minimum at its event. In that case, also compare the distance between the points.

The solution to our problem, again, is easily determined from this output. The construction of the dual-arrangement takes $O(n^2)$ time, resulting in $O(n^2)$ vertices as events. The ordering of the lines would take $O(nlogn)$ and operations on this ordering will take $O(logn)$. Therefore:

$$T(n) = O(n^2) + O(nlogn) + O(n^2)O(logn) = O(n^2logn)$$

This is not as efficient as the first proposed algorithm, but one that I came up with on my own.