CS 266 Homework 3

Zachary DeStefano, PhD Student, 15247592

Due Date: April 24

Problem 3.11

Give an efcient algorithm to determine whether a polygon P with n vertices is monotone with respect to some line, not necessarily a horizontal or vertical one.

For this algorithm, we will use a version of the plane sweep algorithm. We will sweep a horizontal line at each vertex and if the intersection is more than just two points, a line segment, or empty, then it is not monotone.

Here is the algorithm:
1. Sort the vertices by y-coordinate
2. Make an interval tree data structure for the y_{min} and y_{max} coordinates of each of the segments.

`http://en.wikipedia.org/wiki/Interval_tree#Centered_interval_tree`

3. For each vertex v, do the following:
- Sweep a horizontal line at its y-coordinate
- If the number of other segments or points with that same y-coordinate is more than 1
or there is at least one other intersection but v is part of a horizontal segment, then
declare that P is not monotone and exit.
(TODO: Detail the data structure to be used here)
4. If P has not been declared non-monotone, then P is monotone

Correctness:
All the parts where the it will not be polygon will be at event points, which are the vertices. (TODO: Prove this)

Running time:
Step 1 will take O(n log n) time.
Constructing an interval tree for Step 2 is O(n log n) time.
There are n vertices to test in the worst case.
For each vertex, the query will take O($logn + 2$) time since we are requesting 2 results.
Thus the total running time ends up being O(n log n)

Problem 3.14

Given a simple polygon P with n vertices and a point p inside it, show how to compute the region inside P that is visible from p.

In the following figure, the visible region is the triangles with an X inside them.

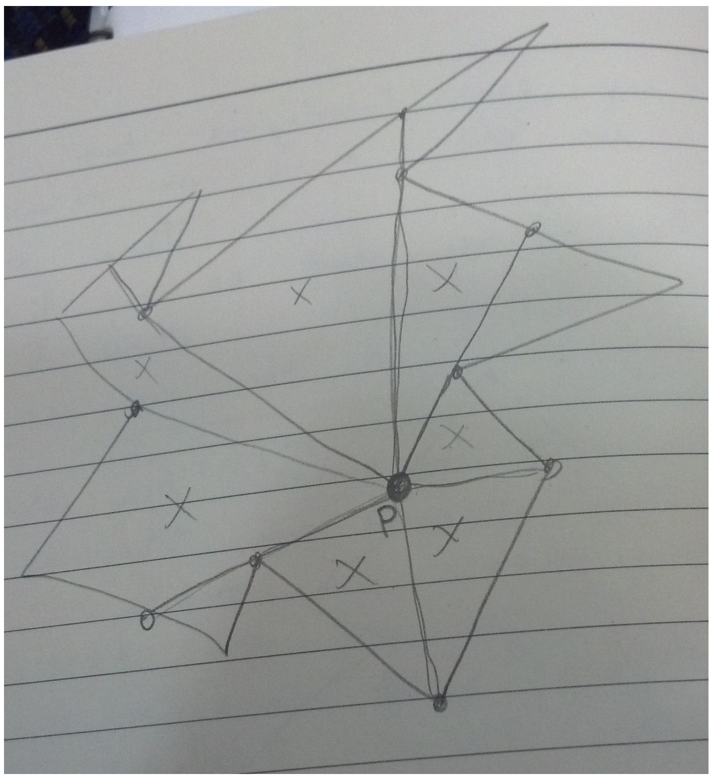

Figure 1: Parts of polygon visible from point p

The following procedure will be used:
For each vertex v with unobstructed view of p:
- Construct line segment from p, then passing through v, and ending at the next line segment in that direction.
- All the triangles that have just been constructed that are around p are the visible region.

Problem 15.2

Algorithm VISIBILITYGRAPH calls algorithm VISIBLEVERTICES with each obstacle vertex. VISIBLEVERTICES sorts all vertices around its input point. This means that n cyclic sortings are done, one around each obstacle vertex. In this chapter we simply did every sort in O(nlogn) time, leading to O(n^2 logn)time for all sortings. Show that this can be improved to O(n^2) time using dualization (see Chapter 8). Does this improve the running time of VISIBILITYGRAPH?

If we take two points a and b and get their duals, $a*$ and $b*$, these lines will intersect assuming that a and b have different x-coordinates.
Denote their intersection point as $p*$
The dual of $p*$, which is the line p in the primal plane will go through a and b.
Thus, the x-coordinate of $p*$ tells us the slope of the line through a and b.

There is a direct relationship between slope and angle.
In the right half-plane the slope increases in counter-clockwise order from $-\infty$ to ∞.
In the left half-plane, the slope increases in clockwise order from $-\infty$ to ∞.
This means that if we split up the points into left and right half and then order them by the slopes, we will be able to get the radial order of the points.

We can now translate these facts into a new algorithm:
1. Dualize the points
2. Compute the arrangement of the duals.
3. For each vertex v in the primal plane:
- Find the line v* in the dual and get its intersection points in the dual in x-coordinate order.
- Go through the points and put the ones corresponding to the left points into set_left and right points into set_right.
- concatenate set_left in increasing order with set_right in decreasing order to get $vertices_in_radial_order$.

Running time:
Computing the arrangement takes O(n^2).
Since the arrangement was computed, there is no need to sort when finding the intersection points for $v*$ thus that step is O(n).
This means that step 3 take O(n^2).
The total running time is thus O(n^2) which is an improvement over the last algorithm.

Problem 15.4

What is the maximal number of shortest paths connecting two xed points among a set of n triangles in the plane?

The visibility graph will have $V = (3n + 2)$.
For each vertex v we have $|v| \leq 3n + 2$ for the number of vertices it is connected to.
The max number of paths is the max number of sequences of such vertices.
We need to start with s and end with t but the vertices in the middle do not matter.
We end up with $(3n)!$ paths possible.