Homework 1

Vu Tuan Phong Pham

September 2019

1. (a) Let's consider P as a polygon.
   We already proved that if P is a simple polygon, there exist a maximal triangulation of P that consist of $n-2$ triangles. We also know that these triangles' vertices are made from the polygon vertices, so for every interior angles of the polygon we traverse is composed of the interior of some of these triangles. Hence, the total sum of interior angle of P is:

   $$\pi(n-2)$$

   The turn angle at a vertex of the polygon is defined to be (π - interior angle at that vertex), so:

   $$\text{sum turn angle} = \sum_{i=1}^{n} (\pi - x_i) \text{ (for } x_i = \text{interior angle at vertex } i^{th})$$

   $$\implies \text{sum turn angle} = \pi n - \pi(n-2) = 2\pi$$

   I don't think we can construct a constant time solution for the sum of turn angle where P is only a closed polyline based on n. It has to be a formula based on all the turn angles. For $n = 4$, I can construct 2 closed polyline with different sum of turn angles.

   (b) for (b), (c), (d), I cannot yet provide a persuasive argument, but I believe the sum of turn angles will strictly differ from 2π. Less or more than 2π is not consistent. I think the sum of turn angles is sufficient to determine if a polyline is a simple polygon or not.

2. Consider the point P with the lowest y coordinate (in case of tie y, take the one with highest x coordinate) and the tangent line through P of the polygon that is parallel to x-axis.
   I claimed that the angle made by 2 edges that share P as the common end-point is less than π and is in the upper half-plane divided by the tangent line. If not, then P is no longer the point with the lowest y coordinate, hence a contradiction.
   Base on this observation, the algorithm is as follow. Consider the point P with the mentioned property, let P_0, P_1 be the points right before and after P in the ordered list. If turn(P_0, P, P_1) is a right turn, then the polygon is given in CW order, else if it is a left turn, then the polygon is given in CCW order. Case of a straight line is dismissed, since I'm taking P with the highest x coordinate.

3. (a) We partition the plane into a grid with each cell of size $r \times r$. A point in the input set that has coordinate (x, y) will be put in the cell $(\frac{x}{r}, \frac{y}{r})$.
   For a point $P(x, y)$, a near neighbor of P can only lie in either of these 3 cells: $(\frac{x}{r}+1, \frac{y}{r}), (\frac{x}{r}, \frac{y}{r}+1), (\frac{x}{r}+1, \frac{y}{r}+1)$ (We are only going forward, so we don't have to look at the other 5 cells next to point P).
   The hash map might be a bit troublesome to program, but there's only n keys at max, so the space complexity should still be O(n).
   We have to be careful of 2 points in the same bucket now, since they can still have distance more than r (We can use bucket of size $\frac{r}{\sqrt{2}} \times \frac{r}{\sqrt{2}}$ so we don't have to check, but I would rather not use floating point operation), but since we are outputing at max all the pairs in the same bucket, and comparing takes constant time (assuming d is small), running time should not change, so it is O($n+k$), for k being the output size.

(b) The only thing that should be adapted for the solution to work under L_1 and L_2 metric is the distance function. For L_1, we are not dealing with any floating points, and the distance is pretty much the same with L_∞ except we take the sum instead of max of all dimensions, so we can use the same solution with the adapted distance function.
For L_2, we don't want to deal with the troublesome and expensive square root floating point operation, we can instead partition the plane to grid with cell of size $r^2 \times r^2$ and keep the distance squared. A point $P(x, y)$ is now bucketed into cell $(\frac{x}{r^2}, \frac{y}{r^2})$. From this point, the analysis should be the same. The space complexity is $O(n)$ and the time complexity is $O(n + k)$ for k being the output size.

4. (a) We will prove this by induction. The base case is $n = 3$. Given a triangle with 3 points $P_0(x_0, y_0), P_1(x_1, y_1), P_2(x_2, y_2)$, using the area of triangle formula $S = \frac{1}{2}ha$ and draw the rectangle with sides parallel to the x and y axis that has P_0, P_1, P_2 on its sides, we can prove that:

$$\text{area} = \tfrac{1}{2}((x_0y_1 - x_1y_0) + (x_1y_2 - x_2y_1) + (x_2y_0 - x_0y_2))$$

and given that the set of points is in CCW order, the area is positive. In CW order, the area will be negative.
Now assume this is correct for all polygons with less than n vertices. Consider a polygon P with n vertices namely $P_0P_1P_2 \dots P_{n-1}$. We already prove that P can be partitioned into smaller polygons by using diagonals. For simplicity, I'll partition P into $P_0P_1P_2$ and $P_0P_2P_3 \dots P_{n-1}$ (a triangle and a polygon with $n - 1$ vertices).
By the induction hypothesis:

$$S(P_0P_1P_2) = \tfrac{1}{2}((x_0y_1 - x_1y_0) + (x_1y_2 - x_2y_1) + (x_2y_0 - x_0y_2))$$

$$S(P_0P_2P_3 \dots P_{n-1}) = \tfrac{1}{2}((x_0y_2 - x_2y_0) + (x_2y_3 - x_3y_2) + \dots + (x_{n-1}y_0 - x_0y_{n-1}))$$

Therefore:

$$S(P) = S(P_0P_1P_2) + S(P_0P_2P_3 \dots P_{n-1}) = \tfrac{1}{2}((x_0y_1 - x_1y_0) + (x_1y_2 - x_2y_1) + \dots + (x_{n-1}y_0 - x_0y_{n-1}))$$

(x_2y_0 and x_0y_2 canceled)

$$\implies S(P) = \tfrac{1}{2}\sum_{i=0}^{n-1}(x_iy_{i+1} - x_{i+1}y_i) = \tfrac{1}{2}|\sum_{i=0}^{n-1}(x_iy_{i+1} - x_{i+1}y_i)| \text{ (if in CW order)}$$

(b) i. Define a function f with meaning $f(i) = \sum_{j=0}^{i-1}(x_jy_{j+1} - x_{j+1}y_j)$.
We can pre-process the problem like the determine if a point is inside a convex polygon problem (based at P_0). Given a chord C with endpoints C_0, c_1, we can binary search and determine in between which 2 consecutive vertices of P are C_0 and C_1. Let's assume, without loss of generality, that C_0 appears first in the ordered list of P. Assume that C_0 is between P_i, P_{i+1} and C_1 is between P_j, P_{j+1} $(i < j)$. Area of half polygon divided by chord C that does not contain P_0 is:

$$S' = \tfrac{1}{2}|f(j) - f(i) + x_jy_i - x_iy_j|$$

For the other half-polygon, we take the area of the polygon subtract by S'.
The pre-process time is $O(n)$, and for every query, time complexity is $O(\log n)$.

ii. We can use the same definition of f function from the previous question.
Using the same idea like the previous question, except now we have to scan the whole list to find the positions of C_0, C_1. The time complexity is now $O(n)$ per query. I cannot yet think of a way to reduce this to $O(\log n)$.