Computational Geometry (CS60064)

Homework Set 2

Ashutosh Kumar Singh - 19CS30008

Vanshita Garg - 19CS10064

Question 1

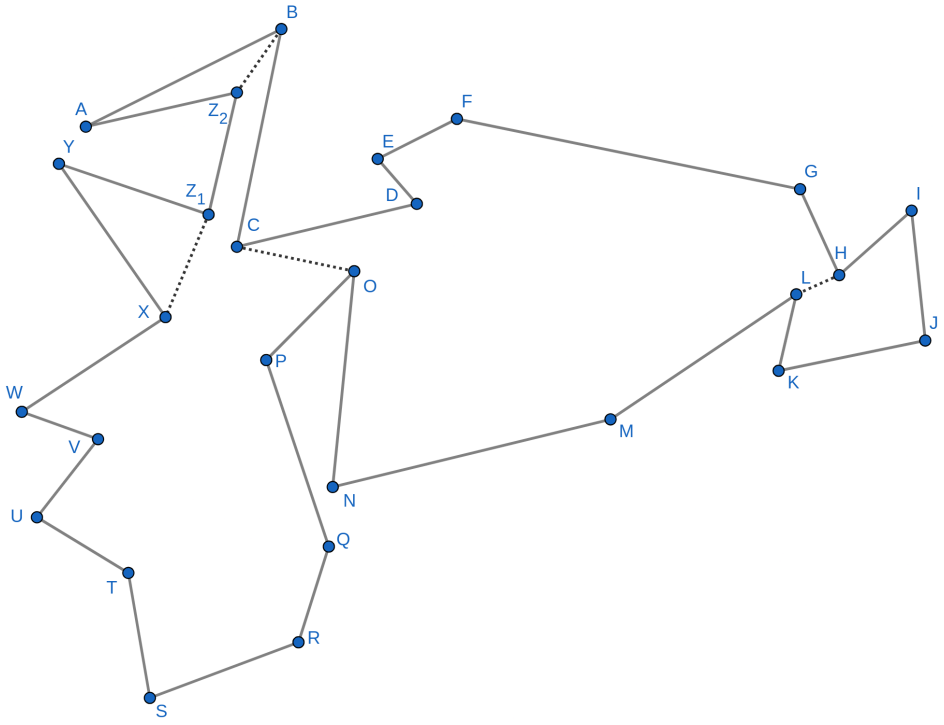

Figure 1: Partitioning into y-monotone polygons

The minimum number of y-monotone polygons that the given polygon can be partitioned into is 5. We have added the 4 diagonals BZ_2, CO, HL and Z_1X.
The given polygon has 4 top vertices (B, F, I, Y) and 4 bottom vertices (A, K, N, S). Every y-monotone polygon can have only one top and only one bottom vertex. Also, a y-monotone polygon cannot have any split or merge vertex. However due to the structure of the polygon, it is not possible to divide the polygon into 4 y-monotone pieces, so we join the merge vertex C with the split vertex O, then we join the merge vertex H with the split vertex L. Then to neutralize the split vertex Z_2, we have to join it to B, and to neutralize the merge vertex Z_1, we join it to X (these are quite intuitive steps from the trapezoid partitioning algorithm). Hence, we need at least 5 partitions to divide the entire polygon into y-monotone pieces.

Question 2

Given n points in general positions in the 2D-plane, we need to determine the Tukey depth of a query point. The Tukey depth of a point q is defined as the smallest number of points in any closed half-plane that contains q.

Algorithm

- First, calculate the angle made by the line joining the query point q and every other point p with the horizontal, measured in the anticlockwise direction. In this manner, we will get an array of $n-1$ angles. Note that no two values in this array can be equal as that would imply q and two other points are collinear, however, we are given that the points are in general position.
- Sort this array of angles in the non-descending order, let us call this array A (see Figure 2, the points after sorting are $p_1, p_2, ..., p_{10}$ in this example).
- We now scan this array using a two-pointer method. For every element $A[l]$ at index l we find the farthest index r such that $A[r] \leq A[l] + \pi$. We can say that all the points from index l to r (both inclusive) will lie on one side of a line passing through q, and all the remaining points will lie on the other side of this line. To visualize this line, imagine a line passing through q and the point at index l, then rotate this line by a very small angle in a clockwise manner. This will be the line partitioning the plane into these half-planes.
- After we have this line we can say that $r-l+1$ points lie on one side and $(n-1)-(r-l+1)$ points lie on the other side. Doing this for every l (upto the last point where $A[l] < \pi$, because the other half is just symmetric), we can take the minimum of both sides, and add 1 at the end (to account for q itself) to obtain the Tukey depth of q. The two-pointer scan in itself takes $O(n)$ time which is explained in the section on time complexity analysis.

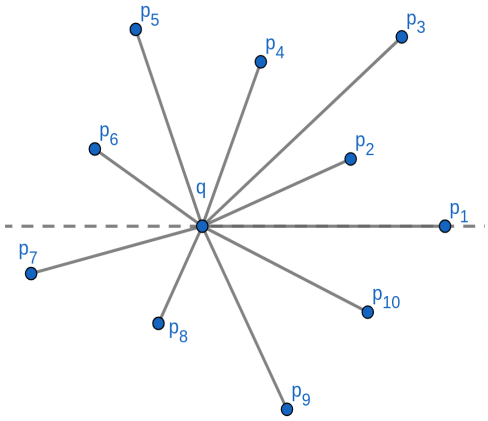

Figure 2: Sorting the points to calculate Tukey depth of q

Algorithm 1 Algorithm to calculate Tukey depth of a query point

1: $P \leftarrow P \setminus \{q\}$ ▷ Remove q from the set of points P
2: $A \leftarrow ()$ ▷ An empty array
3: **for all** $i = 1$ to $n - 1$ **do**
4: Append the anticlockwise angle made by the line joining q and $P[i]$ with horizontal to A
5: **end for**
6: Sort A in non-descending order
7: $d \leftarrow INF$
8: $r \leftarrow 1$
9: $j \leftarrow$ last index such that $A[j] < \pi$
10: **for all** $l = 1$ to j **do**
11: **while** $r < n - 1$ and $A[r + 1] \leq A[l] + \pi$ **do**
12: $r \leftarrow r + 1$
13: **end while**
14: $len \leftarrow min(r - l + 1,\ (n - 1) - (r - l + 1))$
15: **if** $len < d$ **then**
16: $d \leftarrow len$
17: **end if**
18: **end for**
19: $d \leftarrow d + 1$ ▷ To account for q itself
20: **return** d

Time Complexity Analysis
Creating the array of angles A takes $O(n)$ time. Sorting the angles in non-descending order takes $O(n\ log\, n)$ time.
In every iteration of the outer for loop at line 9, l increases by 1. Also, from the $while$ loop at line 10 it is visible that r never decreases. Both l and r move ahead by atmost n. Hence, we can say that the amortized time taken by the two-pointer method is $O(n)$.
Hence, the total time taken by the algorithm is $O(n\ log\, n)$.
Also, the space complexity is $O(n)$.

Question 3

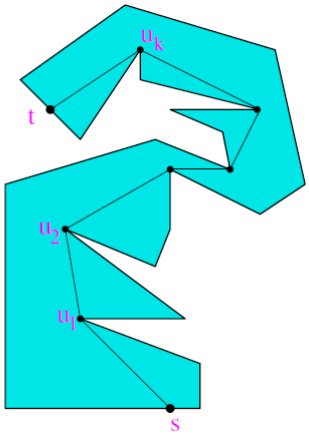

Figure 3: ESP between s and t

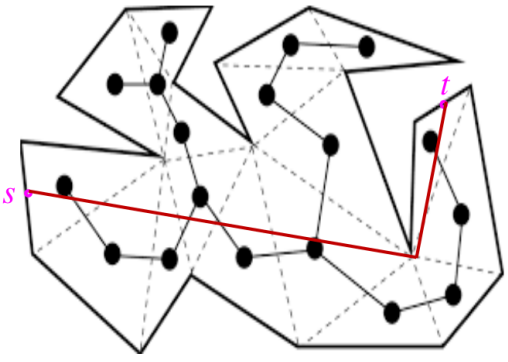

Figure 4: ESP with the dual tree

We are given a simple polygon P with n sides and two points s and t in P, and let T denote a triangulation of P. The Euclidean Shortest Path (ESP) between s and t is defined as the shortest path inside P connecting s and t. Let us denote the ESP between s and t as $\delta(s,t)$.

Claim 1: $\delta(s,t)$ is unique.

Proof:
We shall try to prove this claim by contradiction. If $\delta(s,t)$ is not unique, then let $\delta_1(s,t)$ and $\delta_2(s,t)$ be two distinct shortest paths from s to t. Let α and β be two points of $\delta_1(s,t) \cap \delta_2(s,t)$ such that the two paths are entirely disjoint between α and β, i.e., $\delta_1(s,t) \cap \delta_2(s,t) = \{\alpha, \beta\}$. The two paths $\delta_1(s,t)$ and $\delta_2(s,t)$ enclose some region of P's interior, free of obstacles, since P is simple. At least one of the two paths has a convex corner (since if we have 2 paths between two points consisting of straight line segments, one of them is always bound to have a convex corner), cutting off this convex corner shortens the path, or at least, we can reduce the convexity of one of them to reduce its length, which is a contradiction, since the path was chosen to have the shortest length.

Claim 2: The minimal set of triangles containing the ESP forms a path in the dual tree of T.

Proof:
The diagonals on the path from s to t each split P into two parts, one containing s and the other containing t. The shortest path from p to q crosses only these diagonals of the triangulation, and each of them exactly once. If it crossed any diagonal more than once, the path could definitely be shortened. Also, note that these are the only diagonals that are crossed, because if some other diagonal were being crossed, then we could again reduce the length of the path and this fact can be proven by the triangle inequality (see Figure 5 for this case, going from $s \to t$ is optimal than traversing the path $s \to E \to t$). In fact, the shortest path between the points must cross these separating diagonals in a particular order. When the shortest path from s to s crosses a diagonal d, it must have already crossed all the separating diagonals on the side of d nearer s and none of those on the side nearer t. Hence, we can say that the minimal set of triangles containing the ESP forms a path in the dual tree of T, since we only cross the diagonals separating s and t and pass through each of them exactly once, thus forming a path in the dual tree.

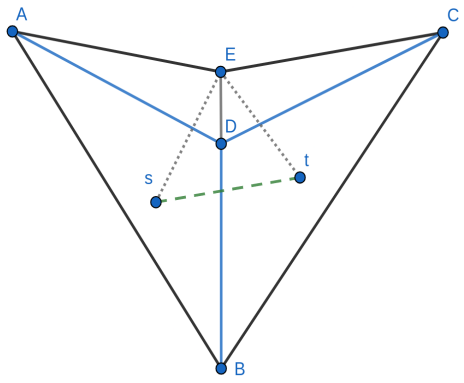

Figure 5: Demonstrating why the ESP will be form a path in the dual tree

Question 4

Part (a)

We have an arbitrary line segment L interior to a convex polygon P with n vertices. Yes, there exists a triangulation such that the number of intersections of L with all diagonals becomes $O(log\ n)$.
We propose an algorithm for the same below.
Here we consider P as an anticlockwise or clockwise sorted sequence of edges of the polygon.

Algorithm 2 Algorithm to construct a triangulation with $O(log\ n)$ intersections with an arbitrary line segment L interior to a convex polygon

```
1: Q ← P
2: T ← φ
3: while Q is not a triangle do
4:     n ← |Q|
5:     Q' ← ()                                          ▷ An empty list
6:     for all i = 1 to ⌊n/2⌋ do
7:         d ← diagonal(Q[2i − 1], Q[2i])
8:         T ← T ∪ {d}
9:         Append d to the list Q'
10:    end for
11:    if n is odd then
12:        Append the edge/diagonal (Q[n − 1], Q[n]) to Q'
13:    end if
14:    Q ← Q'
15: end while
16: return T
```

Q is first initialized to the input polygon P. In each iteration of the *while* loop on line 3, we add a diagonal between every second vertex of Q, and if the number of vertices is odd, we add the last edge too. So, in each iteration we can say that we construct a new polygon using the diagonals of the current polygon using the method stated above.

The loop invariant is that Q is a convex polygon. P is given as a convex polygon, and the invariant is re-established because at each iteration of the inner for loop on line 6, we are simply cutting an "ear" off the polygon; this is equivalent to intersecting the convex polygon with the half-plane consisting of the diagonal extended into an infinite line and the other side of that line from the ear being cut off. Intersections of convex regions results in a convex region, and thus the invariant is maintained.

Also, note that in each step, the number of edges/vertices in Q is almost halved, hence it decreases in each iteration, and so we can be sure that the algorithm will always terminate.

Consider all the diagonals/edges added to Q' during a particular iteration of the while loop to be part of the same level. Suppose we have an arbitrary line L. The diagonals belonging to a particular line form a convex region, hence the line L can intersect the diagonals of a particular level at most twice as a straight line cannot re-enter a convex region once it has exited it. Now, for any level i, we can say that $|Q'|_i = \lceil |Q|_i/2 \rceil$. Thus, there are $O(log\ n)$ levels, and so, L will intersect with the diagonals $O(log\ n)$ times.

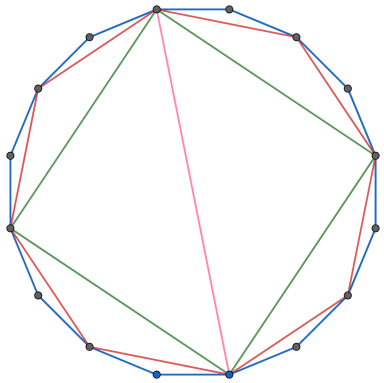

Figure 6: Triangulating a polygon with 16 vertices

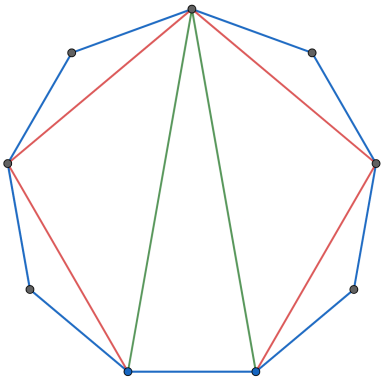

Figure 7: Triangulating a polygon with 9 vertices

Part (b)

No, there cannot be any polygon such that for any triangulation, such a line segment L that is interior to the polygon will have $\Omega(n)$ intersections with diagonals.
The reason for this is that for any triangulation, we can always choose a line segment that lies entirely inside any one triangle. Such a line will have zero intersections with the diagonals, and zero cannot be said to belong to $\Omega(n)$, and hence there cannot exist any polygon such that for any triangulation a line segment interior to the polygon has $\Omega(n)$ intersections with the diagonals.
For illustration, consider the line segment AB in the polygon below.

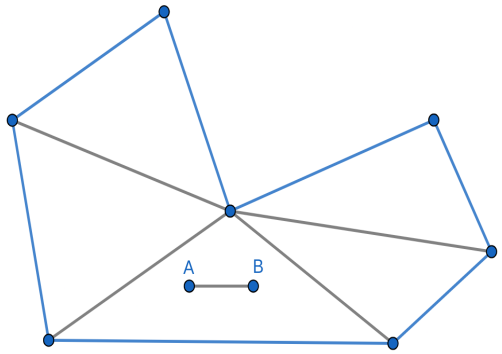

Figure 8: The entire line segment can lie inside a triangle