Assignment 3

Computational Geometry CS60064

Pratyush Jaiswal | 18EE35014
Ram Niwas Sharma | 18CS10044

Question 1

Let S be a set of n disjoint line segments in the plane, and let p be a point not on any of the line segments of S. We wish to determine all line segments of S that p can see, that is, all line segments of S that contain some point q so that the open segment pq does not intersect any line segment of S. Give an $O(n)$ time algorithm for solving this problem. See the example shown on the right.

Answer:

We'll use a sweep line algorithm, in which a line starts at point p and travels in a circle, finding all of the line segments visible from that point.

Algorithm

1. Store all the endpoints in polar coordinate systme with respect to the point P as origin in a list. This list is the set S. Also mark all of the points of S as unvisitied.
2. Sort the above list in ascending order by the polar angles of the points. Create an event queue containing all of the points from S in sorted order, as well as various starting and ending points for each individual line segment, as two different events will occur at both of these locations.
3. Start a ray from p (say R) and rotate it beginning from the horizontal, increasing the angle from 0 to 2π using the sweep line procedure.
4. Sweep status will keep track of all current line segments in S that intersect with R at any given angle, ordered by distance from the point p. Let's name ray R at α an angle $R(\alpha)$. For this, we may consider a binary search tree as a suitable data structure.

5. We extract all the events from the event queue that have the angle α at a specific angle α. As previously stated, there are two types of occurrences that can occur:

    - **Start of a segment**: In the sweep line status, add a new element with a value equal to the distance between this starting point and point p.
    - **End of a segment**: The element that was placed at the beginning of this section is removed from the event queue.

6. We can see that if there are numerous lines running parallel to each other in a given direction, the one nearest to us will be visible while the others will not. We check if there are one or more than items in the event queue at a certain angle α after we've processed all the points with α polar angle. We make the nearest segment visible (the part with the lowest value).

7. We continue to take events from the event queue and repeat the process until the event queue is empty. We return as output all of the line segments that we identified as visible during the procedure as our answer.

Time Complexity Analysis

1. Calculating polar angles for endpoints of n line segments take $O(n)$ time.
2. Sorting all the points (total $2n$ points) and making a event queue takes $O(n \log n)$ time.
3. At most $2n$ events, R must halt, with each event resulting in the addition or deletion of an element from the sweep status. Because deletion and insertion in a binary search tree are $O(nlogn)$ operations, it takes $O(nlogn)$ time for at least $2n$ events.
4. At the end checking for all the segments which were marked as visible also takes $O(n)$ time.

As a result, the entire method has a time complexity of $O(nlogn)$.

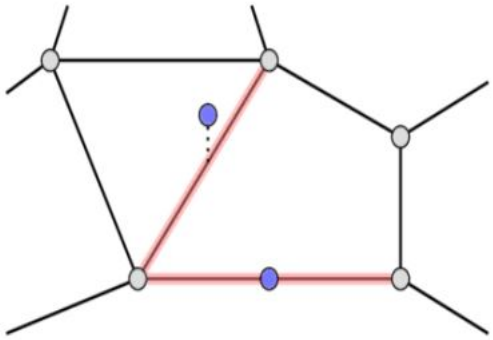
Question 2

Let S be a subdivision of complexity n, represented using DCEL data structure, and let P be a set of m query points. Give an $O((n+m)\log(n+m))$ time algorithm that computes for every point in P in which face of S it is contained.

Answer:

Algorithm

1. We want to identify such an edge for each query point $P(x_0, y_0)$ that if the point belongs to any edge, it lies on the edge we identified; otherwise, this edge must meet the line $x = x_0$ at some unique point $P(x_0, y)$ where $y < y_0$ is the maximum among all such edges. Both cases are depicted in the image below.

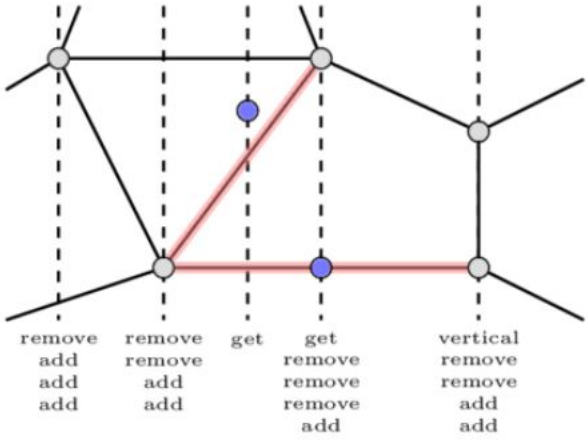


2. We'll use the sweep line algorithm to tackle this problem offline. Let's retain a collection of edges s by iterating through the x-coordinates of query points and edges' endpoints in increasing order. We'll add some events in advance for each x-coordinate.

3. The events we talked about in the previous point will be of four types: **add, remove, vertical, get**:

    - We'll add one vertical event for each x-coordinate for each vertical edge (both ends have the same x-coordinate).
    - For every other edge, we'll add one add event for the endpoints' lowest x-coordinates and one remove event for the endpoints' maximum x-coordinates.
    - Finally, we'll add a get event for each query point's x-coordinate.

4. For each x-coordinate we will sort the events by their types in order (**vertical, get, remove, add**). The following image shows all events in sorted order for each xcoor-



dinate.

5. During the sweep-line procedure, we'll keep two sets. A set s for all non-vertical edges, and a set t for vertical edges alone. At the start of each x-coordinate processing, we'll empty the set.

6. Now let's process the events for a fixed x-coordinate.

   - If a vertical event occurs, we simply add the appropriate edge's endpoints' minimum y-coordinate to t.
   - If we got a remove or add event, we will remove the corresponding edge from **s** or add it to **s**.
   - We must execute a binary search in t for each get event to see if the point is on any vertical edge. We must determine the answer to this inquiry in s if the point does not reside on any vertical edge. To accomplish so, we use a binary search once more.

7. We now have the edge that intersects the line $x = x_0$ at some unique point $P(x_0, y)$ for each query point. In this scenario, we know that our query point is on one of the edge-pair corresponding to this edge's incidence faces. Because we know that $y < y_0$, we will always select the face that is at the top section of this edge. A simple orientation test may be used to identify it.

8. We also have certain edge situations to deal with individually. If no edge is identified, we can conclude that the point is located on the outer face. Also, if a point is on an edge, that point will be a part of two faces, and we will output both of them.

Time Complexity

1. There are m query points and $2n$ endpoints of n edges in total, however two edges share an endpoint, resulting in a total of n endpoints. Sorting them according to their x-coordinates and kinds will be required in steps 2-4. It takes $O((n+m)\log(n+m))$ time to complete this procedure.

2. The deletion and insertion of elements in s and t takes $O(\log(n+m))$ time since the sorted order must be maintained.

3. We have $m+n$ events at most, therefore it takes $O((n+m)\log(n+m))$ time.

4. Steps 6-7 require $O(1)$ time for each query point since we have to access the DCEL and locate the incidence face for specific half-edges, as well as handle any edge situations. As a result, it takes $O(m)$ time in total.

As a result, the entire procedure has a time complexity of $O((n+m)\log(n+m))$.

REFERENCE: https://github.com/e-maxx-eng/e-maxx-eng/tree/master/src/geometry

Question 3

Write a formal proof for the following claim: Any polygon with h holes and a total of n vertices (including those defining the polygon and holes), can always be guarded by $\lfloor (n+2h)/3 \rfloor$ vertex guards. Note that a hole may be surrounded by other holes, and thus it may not be always visible from the boundary of the polygon.

Answer:

- We are going to prove it using triangulation and then further dividing the polygon along the diagonals of the triangulation, converting the polygon into a simple polygon and further making an argument by using *Chvatal's Art Gallery Theorem*

PROOF

1. We triangulate the polygon P into triangles, the result being T.

2. Now, divide the polygon along the diagonals of the triangulation, so that each hole is removed by connecting it to the outside boundary of P.

3. **Cutting along any such diagonal either merges the hole with another, or connects it to the outside**, that leads to reducing the number of holes by one in both the cases.

4. We carefully make h such cuts, one cut on any of the diagonal originating from all the holes, and not resulting in several disconnected pieces. To check it, we just make a dual of the triangulation and remove edges from it shared with the exterior face.

5. Above operation ensures that the result is a single polygon.

6. The resulting polygon with no holes is P', which has $n+2h$ vertices, *(2 vertices added per cut)*, since cuts do not add new triangles, the number of triangles is unchanged.

7. By *Art Gallery* theorem, we can say a polygon with $n + 2h$ vertices can be guarded by $\lfloor (n + 2h)/3 \rfloor$ vertex guards

Question 4

Consider an implementation of Hertel-Melhorn (HM) Algorithm for convex-partitioning of a simple polygon P with n vertices. Assume that a triangulation of P is given. Suggest a data structure and the required procedure so that HM-Algorithm can be implemented in $O(n)$-time.

Answer:

1. As asked in the question, we have to implement the Hertel-Melhorn (HM) Algorithm in $O(n)$-time.
2. It can be accomplished by storing the edges and the diagonals obtained from triangulation in a DCEL *(as explained in lectures)*, along with a parity for a diagonal checker, *for each pair of half-edges corresponding to a edge, check whether the edge is a diagonal or not*.
3. An empty array *arr* is also initialised which stores the list of diagonals involved in convex partitioning, call them D'
4. To check whether a diagonal is in D' or not, we use next and previous half-edges of the pair of half-edges, call them e_1 and e_2.
5. Now, calculate two angles, first between the next edge of e_1 and previous edge of e_2 and second between previous edge of e_1 and next edge of e_2. If any of these angles are $\geq \pi$, then the diagonal belongs to D'

ALGORITHM:

1. Iterate over all the half-edges that make up a diagonal and see if it belongs to D' or not, as discussed previously. If *Yes*, add it to the array; otherwise, delete that half-edge and its pair from the DCEL.
2. We ultimately acquire a list of D' which will split the polygon into smaller convex polygons after performing this for pairs of half-edges corresponding to all the diagonals.

TIME COMPLEXITY:

1. Storing which edge is part of a diagonal for all the half-edges and iterating over them to check if they belong to D' takes $O(n)$ time.

2. Checking whether the two angles subtended are convex and deleting a pair of half-edge from DCEL takes $O(1)$ time.

OVERALL TIME COMPLEXITY : $O(n)$