Computational Geometry - Homework II (Solutions)

K. Subramani
LCSEE,
West Virginia University,
Morgantown, WV
{ksmani@csee.wvu.edu}

1 Problems

1. **Linear Programming:** A simple polygon $\mathcal{P}$ is said to be star-shaped, if it contains a point q, such that for all points $p \in \mathcal{P}$, the line segment $\bar{pq}$ is contained in $\mathcal{P}$. Devise a linear time algorithm to decide if a polygon is star-shaped.

   Solution: Without loss of generality, we assume that the simple polygon $\mathcal{P}$ is represented by a counter clockwise chain of its vertices (numbering n) in array form. Given such a chain, it is straightforward to compute the lines representing each edge. Further, these lines can be converted into half-spaces facing the inside of $\mathcal{P}$ using orientation primitives. All the above operations can be accomplished in linear time. Let $\mathcal{K}$ denote the intersection of these half-planes; $\mathcal{K}$ is called the *kernel* of $\mathcal{P}$.

   Lemma 1.1 *If $\mathcal{K}$ is non-empty, then $\mathcal{P}$ is star-shaped.*

   Proof: Let x denote an arbitrary point in $\mathcal{K}$. By construction, $x \in \mathcal{P}$; further, x is visible to the end-points of each edge in $\mathcal{P}$. It follows that every point on every edge of $\mathcal{P}$ is visible to x. (Why?) Hence, all points along the boundary of $\mathcal{P}$ are visible to x. It therefore follows that all points in $\mathcal{P}$ are visible to x. (Why?) Thus $\mathcal{P}$ is star-shaped. □

   Lemma 1.2 *If $\mathcal{P}$ is star-shaped, then $\mathcal{K}$ is non-empty.*

   Proof: Assume that $\mathcal{P}$ is star-shaped. Let x denote a point in $\mathcal{P}$, such that for every point $q \in \mathcal{P}$, the straight line $\bar{xq}$ is completely contained in $\mathcal{P}$. Therefore, the endpoints of each edge are visible to x and hence x is contained in the polygon-facing half-space defining each edge. Thus x is contained in the intersection of these half-spaces and therefore the kernel $\mathcal{K}$ is non-empty. □

   From Lemma (1.1) and Lemma (1.2), we can conclude that

   Theorem 1.1 *A simple polygon $\mathcal{P}$ is star-shaped if and only if its kernel is non-empty.*

   Note that checking the non-emptiness of the kernel is equivalent to solving a linear program and this task can be accomplished in linear time, using the randomized incremental algorithm discussed in class. □

2. **Orthogonal Range Searching:** Let $\mathcal{P}$ denote a set of n axis-parallel rectangles in the plane. Given a query rectangle $[x : x'] \times [y, y']$, the goal is to report all the rectangles in $\mathcal{P}$ that are completely contained in the query rectangle. Describe a data structure for this problem that uses $O(n \cdot \log^3 n)$ storage and answers queries in time $O(\log^4 n + k)$, where k is the number of reported answers.

   Solution: The crucial observation is that any axis-parallel rectangle is uniquely identified by two of its opposing corners. Without loss of generality, we assume that these corners are the northeast and southwest corners respectively. Assume that the rectangles are labeled as: $A_1,\ A_2,\ \ldots, A_n$. Let (s_x^i, s_y^i) and (n_x^i, n_y^i) denote the southwest and northeast corners of rectangle A_i respectively.

   We store each rectangle in a $4D$ range tree. As per Lemma (5.9) in the book, this can be done using $O(n \cdot \log^3 n)$ storage such that the query time is $O(\log^4 n + k)$, where k is the number of reported answers. □

3. **Point Location:** Give an example of a set of n line segments and the order on them that makes the randomized trapezoidal algorithm create a search structure of size $\Theta(n^2)$ and having query time $\Theta(n)$, in the worst case.

   Solution:

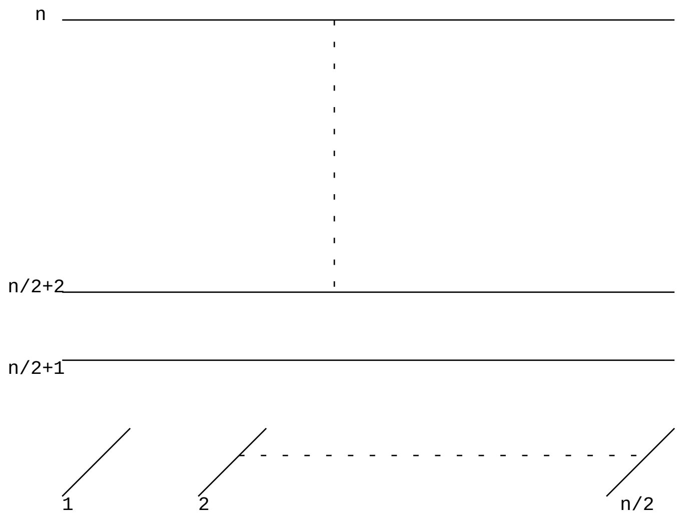

Figure 1: A bad order for the randomized algorithm for trapezoidal map construction

Consider the n non-intersecting segments in Figure (1) and assume that they are added in the order of their labels.

In class, we proved that the time required to insert the i^{th} segment is proportional to the number of newly created trapezoids. For each of the first $\frac{n}{2}$ segments, only $O(1)$ new trapezoids are created; however, the depth of the search structure is $\Omega(n)$ (Verify this!). Figure (2) displays the trapezoidal map after the first $\frac{n}{2}$ segments have been added.

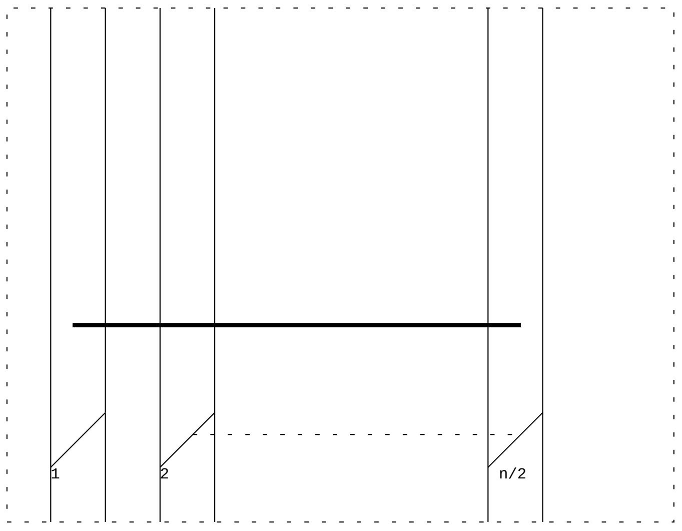

Figure 2: Adding a segment after the first $\frac{n}{2}$ segments have been added

Let Δ denote the set of trapezoids that are bounded below by the segments numbered 1 through $\frac{n}{2}$. For each horizontal segment that is added, the $\Omega(n)$ trapezoids in Δ are replaced and thus the storage increases by $\Theta(n)$. It follows that the total storage requirements after inserting the segment numbered n, is $\Theta(n^2)$. Additionally, the depth of the search structure is $\Omega(n)$ and hence in the worst case, the query time is $\Theta(n)$. $\square$

4. **Point Location:** Describe a deterministic algorithm to compute the trapezoidal map of a set of n non-crossing segments in time $O(n \cdot \log n)$. *Hint: Use a variant of plane sweep.*

   Solution: Let $S = \{s_1, s_2, \ldots, s_n\}$ denote the set of segments; associated with each segment s_i, is its left end-point (x_i^l, y_i, l) and its right end-point (x_i^r, y_i^r). The plane sweep that we employ is very similar to the one used for the line intersection problem. We assume that the initial bounding box is stored in DCEL form; this bounding box encloses all the n segments. We maintain an event queue $\mathcal{Q}$ as a sorted list of the end-points of the segments and the sweep status $\mathcal{T}$ as a balanced binary search segment tree (cf. the line intersection algorithm). We initialize the sweep status with the four segments defining the bounding box. The events are extracted one by one from $\mathcal{Q}$ and handled as follows:

   (a) Left end-point of s_i - Insert the segment into $\mathcal{T}$. Find the predecessor and successor segments of s_i in $\mathcal{T}$. Using these segments update the DCEL structure with the appropriate trapezoids. Observe that the total work done is $O(1)$, per left end-point.

   (b) Right end-point of s_i - Once again find the predecessor and successor segments of s_i in $\mathcal{T}$. Using these segments update the DCEL structure with the appropriate trapezoids. Convince yourself that the work done is $O(1)$. Delete s_i from $\mathcal{T}$.

   Clearly the time taken by the algorithm is dominated by the sorting procedure for the $2 \cdot n$ end-points; this is accomplished in $O(n \cdot \log n)$ time. Each of the $2 \cdot n$ events is handled in $O(1)$ time and hence the total time taken to handle all the event points is $O(n)$. $\square$

5. **Voronoi Diagrams:** Let $\mathcal{P}$ denote a set of n points in the plane. Give an $O(n \cdot \log n)$ time algorithm to find for each point p in $\mathcal{P}$, another point in $\mathcal{P}$, that is closest to it.

   Solution: We first compute the Voronoi Diagram $VD(\mathcal{P})$ of $\mathcal{P}$; this is accomplished in $O(n \cdot \log n)$ time using the algorithm discussed in class. Note that $VD(\mathcal{P})$ is a planar subdivision induced by $\mathcal{P}$ and we may assume that it is stored in a DCEL. Let p be a point in $\mathcal{P}$ and let q be the point in $\mathcal{P}$ that is closest to p. We claim that p and q share an edge in $VD(\mathcal{P})$. Consider the circle drawn with segment $\bar{pq}$ as diameter. This circle contains only the points p and q; for if it were to contain any other point $r \in \mathcal{P}$, then r is closer to p than q is. By Theorem (7.4) in the book, the bisector between points p and q must be an edge of the Voronoi Diagram of $\mathcal{P}$. Thus, in order to find the point closest to p, we need only examine its proximity to its neighbours in $VD(\mathcal{P})$. This operation may cost as much as $O(n)$ for a single site; however, it costs $O(n)$ over all sites $q \in \mathcal{P}$. (Why?)

   $\square$