Computational Geometry - Homework III (Solutions)

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

1 Problems

1. **Duality:** Let R be a set of n red points in the plane and B denote a set of n blue points. A line l is said to be a *separator* for R and B, if l has all the red points on one side and all the blue points on the other. Describe a randomized, linear-time algorithm that decides whether the given sets R and B have a separator.

   Solution: If there exists a separator line l_s for the two sets R and B, then clearly, all the points in R must lie above l_s and all the points in B must lie below l_s or vice versa. Now, consider the duals of the points in R and B, under the duality transform discussed in class; the corresponding line sets are called R' and B' respectively. Let R'_l denote the set of half-planes that are obtained by directing each line in R' downwards; likewise, let R'_u denote the set of half-planes that are obtained by directing each line in R' upwards. B'_l and B'_u are defined similarly. On account of the order-reversing property of the duality transform, either the point corresponding to l_s is simultaneously below all the lines in R' and above all the lines in B' or vice versa. Thus, either the half-plane set $R'_l \cup B'_u$ has a non-empty intersection or the set $R'_u \cup B'_l$ does.

   Not surprisingly, the converse holds as well.

   Lemma 1.1 *If the half-planes in the set $R'_l \cup B'_u$ have a non-empty intersection, then there exists a separator for the given set of points.*

   Proof: Pick any point p in the non-empty intersection of $R'_l \cup B'_u$ and focus on its dual p^* in the primal plane. Since $p \in R'_l$, it lies below all the lines in R' and by the order-reversing property of the duality transform, line p^* must lie above the duals of all the lines in R'. Likewise, since $p \in B'_u$, it lies above all the lines in B' and by the order-reversing property of the duality transform, line p^* must lie below the duals of all the lines in B'. In other words, the line p^* is a separator for the points in R and B. □

   In identical fashion, we can prove that

   Lemma 1.2 *If the half-planes in the set $R'_u \cup B'_l$ have a non-empty intersection, then there exists a separator for the given set of points.*

   From the above discussion, it is clear that the problem of determining the existence of a separator can be reduced to two linear programming problems, which in turn can be solved by the randomized linear-time algorithm discussed in class.

   □

2. **Line Arrangements:** Let L denote a set of n lines in the plane and let $\mathcal{A}(L)$ denote their arrangement. Describe an $O(n \cdot \log n)$ algorithm to compute an axis-parallel rectangle that contains all the vertices of $\mathcal{A}(L)$.

   Solution: Let us focus on finding the x-coordinate of the left-bounding line of the axis-parallel rectangle bounding all the vertices of $\mathcal{A}(L)$, i.e., the leftmost vertex. Clearly, we could compute all the vertices of $\mathcal{A}(L)$ in $O(n^2)$ time and find the intersection point having the smallest x-coordinate.

Let us study the intersection of the n lines with the vertical line $l_m : x = -M$, for a large, positive M. If M is sufficiently large, all the vertices of $\mathcal{A}(L)$ would lie to the right of M. Secondly the lines in L intersect l_m in slope-ordered fashion, i.e., the line with the smallest slope has the largest intercept, the line with the next smallest slope has the second largest slope and so on. If this were not the case, then two lines would have an intersection point to the left of l_m, which is a clear contradiction. (I strongly urge you to study the intersection of non-parallel lines with the y-axis to convince yourselves of the above property.) This immediately implies that the leftmost vertex has to result from the intersection of two lines which are adjacent to each other on l_m. This suggests the following strategy to locate the leftmost vertex: Sort the lines in L, based on slope; let $l_1, l_2, \ldots, l_n$, denote the slope-ordered set. This takes $O(n \cdot \log n)$ time. Compute intersections between pairs of adjacent lines, i.e., between line l_i and line l_{i+1}, for $i = 1, 2, \ldots, n-1$ and select the vertex with the smallest x-coordinate. This takes $O(n)$ time and hence the leftmost vertex can be computed in $O(n \cdot \log n)$ time.

Using similar arguments, we can compute the rightmost, topmost and bottommost vertices in $O(n \cdot \log n)$ time each. It follows that the axis-parallel rectangle containing all the vertices of $\mathcal{A}(L)$ in its interior can be computed in $O(n \cdot \log n)$ time. □

3. **Delaunay Triangulation:** Given a set P of n points in the plane, the Euclidean Minimum Spanning Tree (EMST) is the tree of total edge length connecting all the points in P. Describe an $O(n \cdot \log n)$ algorithm to compute the EMST for P.

   Solution: A few observations are in order:

   (a) There exist a number of efficient algorithms for the Minimum Spanning Tree (MST) problem on arbitrarily weighted graphs. The algorithms which are most popular are those of Kruskal and Prim [CLR92].

   (b) Kruskal's algorithm takes time $\Theta(m \log m)$ on a graph having n vertices and m edges, with appropriate implementations of data structures to support cycle checking conditions [CLR92].

   (c) The crucial step in Kruskal's algorithm, is the sorting of all the edges by their weights. It is this sorting step that takes $\Theta(m \cdot \log m)$ time.

   Observe that if the number of edges in the graph is $O(n)$, then Kruskal's algorithm in fact, runs in time $O(n \cdot \log n)$.

   The naive method for computing the EMST would be as follows:

   (a) Compute the point to point distances for each pair of points in $\Theta(n^2)$ time and create the weighted graph G_P.

   (b) Run Kruskal's algorithm on G_P.

   Unfortunately, G_P has $\Omega(n^2)$ edges and the strategy therefore takes $\Theta(n^2 \cdot \log n)$ time.

   Let $D(P)$ denote the Delaunay triangulation of the point set P.

   Lemma 1.3 *Let $\mathcal{T}(P)$ denote the EMST of the point set P. Then $\mathcal{T}(P) \subseteq D(P)$.*

   Proof: We prove the above lemma by contradiction. Assume that there exist two sites a and b in the point set P, such that $\bar{ab}$ is an edge of $\mathcal{T}(P)$, but not of $D(P)$. This immediately implies that the circle drawn with $\bar{ab}$ as diameter includes another site $c \in P$. (See Theorem 9.6, part (ii) in [dBvKOS00].) Remove the edge $\bar{ab}$ from $\mathcal{T}(P)$; this breaks $\mathcal{T}(P)$ into two components. Without loss of generality, assume that a and c are in the same component. We can now reunite the two components by adding the edge $\bar{cb}$ to get another spanning tree $\mathcal{T}'(\mathcal{P})$. However, observe that the weight of $\mathcal{T}'(P)$ is:

$$w(\mathcal{T}'(\mathcal{P})) \quad = \quad w(\mathcal{T}(\mathcal{P})) - w(\bar{ab}) + w(\bar{cb})$$

   But, $\bar{ab}$ is a diameter and therefore is heavier than $\bar{cb}$, from which it follows that $w(\mathcal{T}'(\mathcal{P})) < w(\mathcal{T}(P))$. This directly contradicts the hypothesis that $\mathcal{T}(P)$ was the EMST of the point set P; we therefore conclude that $\mathcal{T}(P) \subseteq D(P)$. □

   Based on Lemma 1.3, it is clear that we need to run Kruskal's algorithm on $D(P)$ and since $D(P)$ has $O(n)$ edges and can be computed in $O(n \cdot \log n)$ time, we conclude that the EMST of P can be computed in $O(n \cdot \log n)$ time.

   □

4. **Delaunay Triangulation:** Given a set P of n points, the Gabriel graph $\mathcal{G}(P)$ of P is defined as follows: Two points p and q are connected by an edge in $\mathcal{G}(P)$ if and only if the circle with diameter $\bar{pq}$ does not contain any other point of P in its interior.

   (a) Prove that $\mathcal{DG}(P)$ contains $\mathcal{G}(P)$, where $\mathcal{DG}(P)$ denotes the Delaunay graph of P.

   (b) Describe an $O(n \cdot \log n)$ algorithm to compute $\mathcal{G}(P)$.

   Solution:

   (a) Let a and b denote two points in P and let $\bar{ab}$ denote an edge of $\mathcal{G}(P)$. By definition, the circle with diameter $\bar{ab}$, viz., C_{ab}, does not contain any point $r \in P$, in its interior. We need to consider precisely two possibilities, viz.,

      (i) There exists a point r on the circle C_{ab}. This forces the triangle $\triangle abc$ to be a face of the Delaunay Triangulation of of $\mathcal{DG}(P)$, as per part (i) of Theorem 9.6 in [dBvKOS00] and hence edge $\bar{ab}$ is part of $\mathcal{DG}(P)$.

      (ii) There does not exist a point of P on the circle C_{ab}. In this case, there is closed disk with a and b on its boundary, and containing no other points of P. As per part (ii) of Theorem 9.6 in [dBvKOS00], it must be the case that $\bar{ab}$ is an edge of the Delaunay Graph $\mathcal{DG}(P)$.

   In either case, we see that an edge of the Gabriel graph is an edge of the Delaunay graph and hence $\mathcal{G}(P) \subseteq \mathcal{DG}(P)$.

   (b) The Gabriel graph of the point set P can be computed as follows:

      (i) First compute the Delaunay Graph $\mathcal{DG}(P)$ of the point set P.

      (ii) For each edge $\bar{pq} \in \mathcal{DG}(P)$, construct a circle C_{pq}, with $\bar{pq}$ as diameter. If any site of P lies in C_{pq}, the edge $\bar{pq}$ is deleted.

   Clearly, the above procedure delivers the Gabriel graph of P, since all the offending edges are deleted. Now observe that $\mathcal{DG}(P)$ can be computed in $O(n \cdot \log n)$ time. Secondly, for each edge $\bar{pq}$ in $\mathcal{DG}(P)$, we only need to look at the neighbours of p and q for the circle violation property. This implies that a vertex v participates in $deg(v)$ circle violation tests. Since $\mathcal{DG}(P)$ is a planar graph, the total number of circle violation tests is $O(n)$; from which it follows that $\mathcal{G}(P)$ can be computed in $O(n \cdot \log n)$ time.

   □

5. **Geometric Data Structures:** Let I denote a set of n intervals in the plane. Given an interval $r = [x : x']$, we are interested in all the intervals $p \in I$, such that p is completely contained in r. Describe a data structure that uses $O(n \cdot \log n)$ storage and answers these queries in $O(\log n + k)$ time, where k is the number of answers.

   Solution: There are a number of different ways to approach this problem. I prefer to use the 2-dimensional range tree, since this structure reflects my perspective.

   Each interval $p \in I$ is defined by its left coordinate and its right coordinate. We construct a 2-dimensional range tree T as follows:

   (a) The main tree is a balanced binary search tree $\mathcal{T}$, built on the left coordinate of each interval.

   (b) For any internal or leaf node v, the canonical subset $P(v)$ is stored in a balanced binary search tree $\mathcal{T}_{assoc}(v)$ on the right coordinate of the points. Recall that the canonical subset of a node v is the subset of points stored in the leaves of the subtree rooted at v. Note that $\mathcal{T}_{assoc}(v)$ stores not only the right coordinate of intervals but the intervals themselves, for reporting purposes.

   As per the discussion on range trees, $\mathcal{T}$ can be built in $O(n \cdot \log n)$ time and takes $O(n \cdot \log n)$ space. Queries with respect to a given interval $[x : x']$ are answered as follows:

   (a) First the split node v_{split} is located in $\mathcal{T}$.

   (b) If v_{split} is a leaf, then check if the interval corresponding to this left endpoint must be reported, and if so report the corresponding interval.

   (c) If v_{split} is not a leaf, then recursively traverse the left and right subtrees of v_{split} calling 1-dimensional range queries on the appropriate subtrees. (See Pg. 108 of [dBvKOS00].)

As described, the query processing takes $O(\log^2 n + k)$ time, where k is the number of reported answers; however, we can use fractional cascading to reduce the query time to $O(\log n + k)$. □

References

[CLR92] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill Book Company, Boston, Massachusetts, $2nd$ edition, 1992.

[dBvKOS00] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry - Algorithms and Applications*. Springer-Verlag, 2^{nd} edition, 2000.