# Theory of Algorithms. Homework 4

Peter Racioppo

**Computational Geometry – Algorithms and Applications**

**1.) 6.8 & 6.13**

**6.8.) Design a deterministic algorithm to compute the trapezoidal map of a set of non-crossing line segments in O($n$log$n$) time using the plane sweep paradigm.**

Scan from left to right. When the leftmost vertex of a line segment is reached, add it to a balanced binary search tree on its $x$-coordinate. Draw vertical lines upward and downward from the vertex, until reaching the first line segment that contains any of the vertices in the BST. When the rightmost vertex of a line segment is reached, remove this line segment from the BST.

**6.13.) Use a plane sweep argument to prove that the trapezoidal map of $n$ line segments in general position has at most 3$n$+1 trapezoids.**

Suppose that $k$ line segments intersect at a point. We can perturb the endpoint of one of the segments on the vertical line drawn through this vertex without changing the number of trapezoids in the map. So, without loss of generality, we can consider the case that no two segment endpoints are identical. In this case, scanning a vertical sweep line from left to right, it's clear that the left endpoint of every segment corresponds to a single trapezoid to the left of this point, and the right endpoint of every segment corresponds to two trapezoids to its left. So in total there are no more than 3$n$ trapezoids to the left of the rightmost right endpoint. There is exactly one trapezoid to the right of this point, so altogether there are no more than 3$n$+1 trapezoids.

**2.) Given two sets of points $A$ and $B$ with $n$ points each, give an algorithm to compute, among all pairs of points $a$ in $A$ and $b$ in $B$, the pair that has the smallest cost. What is the running time of your algorithm? *Costs are Euclidean distances.***

The Delaunay triangulation of a set of points in the plane contains the minimum spanning tree of the points by the following argument: In order for a particular edge $\underline{pq}$ not to be in the Delaunay triangulation, it is a necessary (though not sufficient) condition that the circle intersecting $p$ and $q$ with diameter $|\underline{pq}|$ contain another point. To see why this is the case, note that this circle is the smallest possible circle that can be drawn through both $p$ and $q$, so either its left or right side is contained by any circumcircle passing through $p$ and $q$. If this circle is empty, $p$ and $q$ can always form an empty circumcircle with the point nearest to either of the vertices. Thus, if the minimum spanning tree contained an edge that were not in the Delaunay triangulation, a point would exist inside such a circle on this edge, and this point would form an edge of lower distance to either of the points in the original edge.

So we can do the following: Compute the Delaunay triangulation in $O(n\log n)$ time. There are $O(n)$ edges in the Delaunay triangulation, since there are exactly $3n–3–k$ edges in any triangulation, where $k$ is the number of vertices on the boundary of the convex hull. We then find a minimum spanning tree on this set of edges in $O(n\log n)$ time using Prim's algorithm or Kruskal's algorithm. The minimum spanning tree contains $O(n)$ edges and must contain the minimum cost edge, so we can find the minimum cost edge in $O(n)$ time. Altogether, it's $O(n\log n)$.

**3.) 9.14**

**The relative neighborhood graph (RNG) of a set $P$ of points in the plane is defined as follows: Two points $p$ and $q$ are connected by an edge of the RNG iff**

$$d(p,q) \le \min_{r \in P, r \neq p,q} \max\big(d(p,r), d(q,r)\big)$$

**a.) Define lune(*p*,*q*) as the intersection of the two circles around *p* and *q* of radius *d*(*p*,*q*). Prove that *p* and *q* are connected in the rel. neighbor graph iff lune(*p*,*q*) doesn't contain any other point in *P*.**

According to the definition, in order to form an edge, $p$ and $q$ must be closer to each other than either is to any other point in $P$. If there is another point $r$ inside lune($p$,$q$), this point is closer to both $p$ and $q$ than $p$ and $q$ are to each other, that is $\max(d(p,r),d(q,r)) < d(p,q)$, so $\underline{pq}$ cannot be in the relative neighbor graph (RNG). On the other hand, if there is no point inside lune($p$,$q$), then for all $r$, $\max(d(p,r),d(q,r))$ is greater than or equal to the distance from $p$ (resp. $q$) to the boundary of lune($p$,$q$) intersected by $q$ (resp. $p$) which is exactly $d(p,q)$, so $\underline{pq}$ is in the RNG.

**b.) Prove that *DG*(*P*) contains the RNG of *P*.**

Define the diameter circle of $\underline{pq}$ as the circle intersecting $p$ and $q$ with diameter $|\underline{pq}|$. Any edge for which the diameter circle is empty must be in the Delaunay triangulation (see Qs. 2). The region lune($p$,$q$) contains the diameter circle of $\underline{pq}$, so every edge with empty lune($p$,$q$) also has empty diameter circle. Thus, all edges with empty lune($p$,$q$) must be in the Delaunay triangulation, which implies that $DG(P)$ contains the RNG of $P$.

**c.) Design an algorithm to compute the RNG of a given point set.**

Compute $DG(P)$ in $O(n\log n)$ expected time. For each of the $O(n)$ edges in $DG(P)$, check whether the corresponding lune region is empty. At each of the $O(n)$ lune regions, it takes $O(n)$ time to check whether each of the $O(n)$ other vertices are inside, so the algorithm is $O(n^2)$. The performance can be improved with a sweep line approach. Each lune region has two endpoints where the two circles that form it meet. Sweep the plane top to bottom, adding a lune to a balanced binary search tree on its $x$-coordinate when you reach its first endpoint. If two lunes intersect, reverse their order in the BST. When a vertex is encountered, check it against the adjacent lunes to its left and right. Remove a lune from the BST when its second endpoint is encountered. There are $O(n)$ lunes, and a pair of lunes can intersect no more than 4 times, so there are $O(n)$ events. Inserting and deleting takes $O(\log n)$ at each event. Altogether, the algorithm is $O(n\log n)$ expected time.

**4.) Let S be a set of *n* real numbers. Design a binary search tree data structure that allows for insertions (without any rebalancing) so that when S is inserted in a random order (as given by a random permutation of S), the height of the tree is, in expectation, *O*(log*n*). Use the backward analysis method for bounding the height.**

In a sorted list of the points, for a pair of adjacent points, one must be the ancestor of the other, since otherwise they would share a common ancestor of intermediate value. In order for a pair of points of distance $k$ away from each other in the sorted list to be related, either of the points has to have been chosen before every other intermediate point, with probability $2/k$. Thus, the probability that a point is an ancestor of another point $k$ away from it in the sorted list is $1/k$. The expected number of ancestors of the first or last element of the sorted list is then $\sum_{i=1}^{n} 1/i$. In general, the expected number of ancestors of the $m$th element of the sorted list is $\sum_{i=1}^{n-(m-1)} 1/i + \sum_{i=1}^{m-1} 1/i$ which is bounded by the value for the median, which is bounded by the case where there are an odd number of elements: $\sum_{i=1}^{n/2} 2/i = 2H_{n/2}$. But $2H_{n/2} \le 2\log(n/2) + C = O(\log n)$. The expected value for the whole set is bounded above by the maximum expected value, so the expected number of ancestors of a random point is $O(\log n)$. So I don't think it's necessary to design a particular data structure. Just walk through the random permutation and add vertices to a BST as you come to them, and it will have expected height $O(\log n)$ by the preceding argument.

**5.) 7.4 & 7.5.**

**7.4.) Prove that the breakpoints of the beach line, as defined in Sec. 7.2, trace out the edges of the Voronoi diagram while the sweep line moves from top to bottom.**

Define $parab_i$ as the distance between the $i$th vertex and the $i$th parabola and $sweep_i$ as the distance between the $i$th vertex and the sweep line. The $i$th parabola is defined as being the set of points such that $parab_i = sweep_i$. At an intersection point of the $i$th and $j$th parabolas, $parab_i = sweep_i = sweep_j = parab_j$ , so this point is equidistant from the $i$th and $j$th vertices, which is the definition of the $ij$th boundary of the Voronoi region.

**7.5.) Give an example where the parabola defined by some site $p_i$ contributes more than one arc to the beach line. Can you give an example where it contributes a linear number of arcs?**

When the sweep-line reaches a new point, a new parabola of zero width is added, which becomes less narrow as the line continues moving down. The parabola that had existed prior to the addition of this new point and which is intersected by the new parabola now contributes two arcs to the beach line, corresponding to the sections of the old parabola to the left and right of the intersection(s). A linear number of arcs would occur if a linear number of new vertices were to appear at the sweep-line in the range in which the old parabola is in the beach line. In particular, $n$ such vertices would result in the old parabola contributing $n$+1 arcs to the beach line.

**6.) Let K be a triangulation of a set of $n \geq 3$ points in the plane. Let L be a line that avoids all the points. Prove that L intersects at most $2n–4$ edges in K.**

Any triangulation has $n_t = 2n–2–k$ triangles, where $k$ is the number of points on the boundary of the convex hull. But $k$ can be no less than 3, so $n_t \leq 2n–5$.

Between any pair of triangles through which we pass, we have to pass across a shared edge, and we can only pass through one of the remaining edges of the next triangle. The only exception to this is the first triangle, through which we must pass through two edges. So the number of edges through which we pass is no greater than one more than the number of triangles. That is, $n_e \leq n_t + 1 \leq 2n–4$.