Test 2 Solutions

November 25, 2004

1. **Show by contradiction that the MST of a set of points S is a subgraph of the Gabriel graph of S**

   Proof:
   We will show that every edge ab with $a, b \in S$ belonging to the MST of S also belongs to the Gabriel Graph GG of S.

   Suppose $ab \in MST$ but $ab \notin GG$

   This means that the circle C having ab as diameter is *not* empty. Let $c \in S$ is inside circle C, and that c is not connected to a.

   (a) Suppose that c and a are connected by some path that does not pass through ab. Also, c is connected to b though some other path. This means that we get from a to b though a path passing through c. Therefore, if we remove edge ab, we will still have a connected tree, but with one less edge. Thus, our assumption that $ab \in MST$ was wrong. Contradiction.

   (b) Suppose that c and a are connected by some path that passes through ab. Since c is inside C, then $\bar{ac} < \bar{ab}$. Therefore, if we remove edge ab, and connect a to c we will have replaced a long edge of the MST with a short one. Thus, our assumption that $ab \in MST$ was wrong. Contradiction.

   Therefore, circle C has to be empty $\Rightarrow ab \in GG$.

2. **Show that the complexity for computing the Gabriel Graph GG of a set of points S is $\Omega(n \log n)$ (reduction from sorting).**

   We will reduce the problem of computing the Gabriel Graph of a set of points S to the problem of sorting. That is, we will show how to solve a sorting problem using a black box that computes the Gabriel Graph; and since we know that the lower bound of sorting is at least $\Omega(n \log n)$, then the complexity of computing the Gabriel Graph is at least $\Omega(n \log n)$.

   Suppose we have a set of numbers $x_1, x_2, \ldots, x_n$ that we want to sort using the black box that computes the Gabriel Graph. We map each of these integers on the x-axis - that is, we map each point $x_i \rightarrow (x_i, 0)$ for all $i = 1, 2, \ldots, n$. This operation takes linear time. Now, we compute the Gabriel Graph of these n points. Since these points are on a st. line, then the circle that passes through every consecutive point on this line will be empty, and the circle passing through non-consecutive points will not be empty. Therefore, the Gabriel Graph of a set of points mapped

on the x-axis is just a path that connects these points in a decreasing or increasing order. All we have to do is get the first vertex of this path and follow the edges. This way we will visit all the n points in sorted order. The complexity of this step is $O(n)$. Mapping the points back to the numbers they represented ($(x_i, 0) \rightarrow x_i$) will give us the list of sorted numbers.

Therefore, we have solved the sorting problem using the GG black box and thus, we can conclude that the complexity of computing the Gabriel Graph is at least $\Omega(n \log n)$.

3. **Given a set S of n vertical and horizontal line segments in the plane in general position, describe and algorithm for computing all the intersection points between the vertical and horizontal line segments. The algorithm should run in $O(n \log n + k)$ worst-case time, where k is the number of intersections reported. Argue the complexity and the correctness of the algorithm**

The algorithm runs as follows:
1. Sort the n points by x-coordinate
2. Sweep a vertical line from left to right starting from x_{min}
3. At every vertex v:

(i) If v is the left endpoint of a horizontal line, insert it in a $(2-4)$ tree sorted by its y-coordinate.

(ii) If v is the right endpoint of a horizontal line, remove it from the $(2-4)$ tree.

(iii) If v is the upper endpoint of a vertical line, then find all the horizontal lines in the $(2-4)$ tree that have their y coordinates between the y coordinates of v and that of its lower endpoint. These points are the intersection points.

Complexity:
The first step (sorting) takes $O(n \log n)$ time. Inserting and deleting from a $(2-4)$ tree takes $O(\log n)$ time at each step, thus the insertion/deletion is bounded by $O(n \log n)$. Finding the intersection points is bounded by the number of intersections k. Therefore, the complexity of the algorithm is $O(n \log n + k)$ where k is the number of intersection points.

(Note that you can also use a balanced binary search tree instead of a $(2-4)$ tree.)