15-750 Graduate Algorithms

Homework # 4 **due: March 6, 2001**

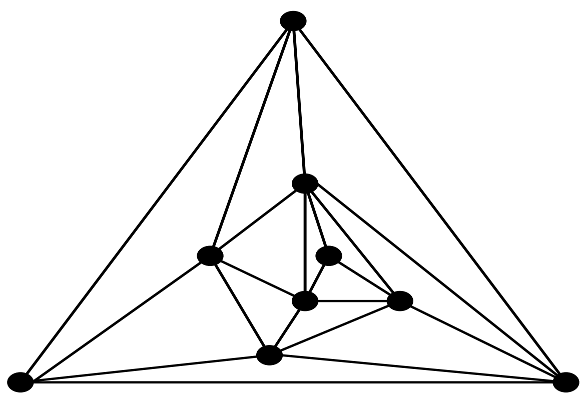
Groundrules:

- Homeworks will generally consist of easier *exercises* and harder *problems*. You should do the exercises by yourself, but you may work with a friend or two on the problems if you want — just write down who you worked with. One exception: no fair working with someone who has already figured out (or already knows) the answer. Of course, your writeup should be in your own words.
- If you've seen a problem before (sometimes we will give problems that are "famous"), then say that in your solution (it won't affect your score, we just want to know). Also, if you use any sources other than the book or those on the web page, write that down too. It's fine to look up a complicated sum or inequality, but don't look up an entire solution.

Problem 1: Point Location

In this problem we will develop another algorithm for planar point location.

Assume we are given as input some triangulation, and for simplicity, let's assume that the convex hull has just three points (we can do this by putting everything inside one large triangle, blowing up the problem by at most a constant factor). E.g., see the figure below. Given a query point q, we want to output which triangle the point is in, or say that q is outside the region. Our goal is to build up a simple data structure to support this.



(a) Prove that the average degree of a vertex (thinking of the triangulation as a graph) is less than 6. To do this, you may wish to use Euler's formula, which states that $v - e + f = 2$, where v is the number of vertices, e is the number of edges, and f is the number of faces (including the "outside"). You proof actually will hold for any planar graph (triangulations are just the worst case).

Note: This is proven as Theorem 14.10 in the book, but please do it yourself.

(b) The first step in the algorithm is to find a large independent set of vertices, all of which have degree at most 6. Give an algorithm to do this: the independent set should have a constant fraction of the nodes.

(c) Say S is the set of nodes found in (b) above. We now remove vertices in S from the graph, and re-triangulate (exception: don't remove any of the boundary vertices – but S could have had at most one of them). We then recursively construct a point location algorithm for the new smaller graph.

The last step is: given the location of the query q in the new graph, we want in *constant* time to produce the location of q in the original graph. Show how to do this. Make sure to argue why the points in S don't interfere with each other. What could go wrong if our input graph wasn't a triangulation?

(d) Putting this all together, what is the running time for point location in our overall datastructure?

Problem 2: Fair Carpooling

There are n people, $p_1, p_2, \ldots, p_n$. Every day, some subset of these people carpool together. Let S_1 be the set that carpool together on day 1. There are m such sets, $S_1, \ldots, S_m$. For each set, one of the people in the set must be chosen to be the driver that day. Driving is not desirable, so the people want the work of driving to be divided as equitably as possible. Your task in this problem is to give an algorithm to do this efficiently and fairly.

The fairness criterion is the following: A person p is in some of the sets. Say the sizes of the sets that she's in are $a_1, a_2, \ldots, a_k$. Person p should really have to drive $\frac{1}{a_1} + \frac{1}{a_2} + \cdots + \frac{1}{a_k}$ times, because this is the amount of resource that this person effectively uses. Of course this number may not be an integer, so let's round it up to an integer. The fairness criterion is simply that she should drive no more times that this many times.

For example, say that on day 1, Alice and Bob carpool together, and on day 2, Alice and Carl carpool together. Alice's fair cost would be $1/2 + 1/2 = 1$. So Alice driving both days would not be fair. Any solution except that one is fair.

Give a polynomial-time algorithm for computing a fair solution. (And thereby proving that a fair solution always exists!)

Hint: Think about modeling the problem using network flow.