15-451 Algorithms, Fall 2009

Homework # 6 **due: Tue-Wed, November 17-18, 2009**

Ground rules:

- This is an oral presentation assignment. You should work in groups of three. At some point before **Sunday, November 15 at 11:59pm** your group should sign up for a 1-hour time slot on the signup sheet on the course web page.
- Each person in the group must be able to present every problem. The TA/Professor will select who presents which problem. The other group members may assist the presenter.
- You are not required to hand anything in at your presentation, but you may if you choose.

Problems:

1. [Graduation revisited] Cranberry-Melon University has switched to a less draconian policy for graduation requirements than that used on Homework 5. As in Homework 5, there is a list of requirements $r_1, r_2, \ldots, r_m$, where each requirement r_i is of the form: "you must take at least k_i courses from set S_i". However, unlike the case in Homework 5, a student *may* use the same course to fulfill several requirements. For example, if one requirement stated that a student must take at least one course from $\{A, B, C\}$, another required at least one course from $\{C, D, E\}$, and a third required at least one course from $\{A, F, G\}$, then a student would only have to take A and C to graduate.

   Now, consider an incoming freshman interested in finding the *minimum* number of courses that he (or she) needs to take in order to graduate.

   (a) Prove that the problem faced by this freshman is NP-hard, even if each k_i is equal to 1. Specifically, consider the following decision problem: given n items labeled $1, 2, \ldots, n$, given m subsets of these items $S_1, S_2, \ldots, S_m$, and given an integer k, does there exist a set S of at most k items such that $|S \bigcap S_i| \geq 1$ for all S_i. Prove that this problem is NP-complete (also say why it is in NP).

   (b) Show how you could use a polynomial-time algorithm for the above decision problem to also solve the search-version of the problem (i.e., actually find a minimum-sized set of courses to take).

   (c) We could define a *fractional* version of the graduation problem by imagining that in each course taken, a student can elect to do a fraction of the work between 0.00 and 1.00, and that requirement r_i now states "the sum of your fractions of work in courses taken from set S_i must be at least k_i" (courses not taken count as 0). The student now wants to know the least total work needed to satisfy all requirements and graduate.

   Show how this problem can be solved using *linear programming*. Be sure to specify what the variables are, what the constraints are, and what you are trying to minimize or maximize.

2. [Euler tours] An Euler tour in a graph is a cycle that traverses each edge exactly once (it may visit some vertices multiple times — i.e., it doesn't have to be a *simple* cycle). In this problem we will assume the graph is undirected.

   (a) Suppose the graph has some node of odd degree. Then there cannot be an Euler tour. Why?

   (b) On the other hand, if all nodes have even degree (and the graph is connected) then there always does exist an Euler tour. Prove this by giving a polynomial-time algorithm that finds an Euler tour in any such graph. Your algorithm should work for multigraphs too (multiple edges allowed between any two vertices).

   Hint: Suppose you start at some node x and just arbitrarily take a walk around the graph, never going on any edge you've traversed before. Where will you end up? Now, what about parts of the graph you haven't visited?

3. [TSP approximation] Given a weighted undirected graph G, a *traveling salesman tour* for G is the shortest tour that starts at some node, visits all the vertices of G, and then returns to the start. We will allow the tour to visit vertices multiple times (so, our goal is the shortest cycle, not the shortest simple cycle). This version of the TSP that allows vertices to be visited multiple times is sometimes called the *metric* TSP problem, because we can think of there being an implicit complete graph H defined over the nodes of G, where the length of edge (u, v) in H is the length of the shortest path between u and v in G. (By construction, edge lengths in H satisfy the triangle inequality, so H is a metric. We're assuming that all edge weights in G are positive.)

   (a) Briefly: show why we can get a factor of 2 approximation to the TSP by finding a minimum spanning tree T for H and then performing a depth-first traversal of T. (If you get stuck, the CLRS book does this in a lot more sentences in section 35.2.1.)

   (b) The minimum spanning tree T must have an even number of nodes of odd degree (only considering the edges in T). In fact, *any* (undirected) graph must have an even number of nodes of odd degree. Why?

   (c) Let M be a minimum-cost perfect matching (in H) between the nodes of odd degree in T. I.e., if there are $2k$ nodes of odd degree in T, then M will consist of k edges in H, no two of which share an endpoint. Prove that the total length of edges in M is at most one-half the length of the optimal TSP tour.[1]

   (d) Combine the above facts with your algorithm from 2(b) to get a 1.5 approximation to the TSP. Hint: think about the (multi)graph you get from the union of edges in T and M.

   The above algorithm is due to Christofides [1976]. Extra credit and PhD thesis: Find an algorithm that approximates the TSP to a factor of 1.49.

[1] We didn't prove it in class, but there are efficient algorithms for finding minimum cost perfect matchings in *arbitrary* graphs (not just bipartite graphs).