

# 15-451 Algorithms, Fall 2006

Homework # 6

due: Mon-Tue, November 20-21, 2006

---

## Ground rules:

- This is an oral presentation assignment. You should work in groups of three. At some point before **Saturday, November 18 at 11:59pm** your group should sign up for a 1-hour time slot on the signup sheet on the course web page.
- You are not required to hand anything in at your presentation, but you may if you choose. If you do hand something in, it will be taken into consideration (in a non-negative way) in the grading.

## 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  $r_1, r_2, \dots, r_m$ , 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  $1, 2, \dots, n$ , given  $m$  subsets of these items  $S_1, S_2, \dots, S_m$ , and given an integer  $k$ , does there exist a set  $S$  of at most  $k$  items such that  $|S \cap S_i| \geq 1$  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 gets a score between 0.00 and 1.00, and that requirement  $r_i$  now states “the sum of your scores 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 graduate, defined as the the minimum sum of all scores needed to satisfy all the requirements.  
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. [Multicommodity Flow] The *multicommodity flow* problem is just like the standard network flow problem except we have  $p$  sources  $s_1, \dots, s_p$  and  $p$  sinks  $t_1, \dots, t_p$ . The stuff flowing from  $s_1$  has to go to  $t_1$ , the stuff from  $s_2$  has to go to  $t_2$ , and so on. For each sink  $t_i$  we have

a demand  $d_i$ . (E.g., we need to get  $d_1$  trucks from  $s_1$  to  $t_1$ ,  $d_2$  trucks from  $s_2$  to  $t_2$ , and so on.) Our goal is to solve for a *feasible solution* — a solution satisfying the demands — if one exists. (Just like with standard network flow, the total amount of stuff going on some edge  $(u, v)$  cannot exceed its capacity  $c_{uv}$ . However, our “flow-in = flow-out” constraints must hold separately for each commodity. That is, for every commodity  $i$ , and every vertex  $v \notin \{s_i, t_i\}$ , the amount of type- $i$  stuff going into  $v$  must equal the amount of type- $i$  stuff going out from  $v$ .)

- (a) Show how to solve this using linear programming.
  - (b) The above problem assumes all edges are directed. E.g., if you had a highway with 3 lanes going one way and two lanes going the other, that would be a directed edge of capacity 3 in one direction and a directed edge of capacity 2 in the other. Suppose we wanted to also allow undirected edges  $e$  with capacities  $c_e$  (like a highway with 5 lanes where part of your job is to decide how many will go one way and how many will go the other). How can you modify your LP formulation to handle this as well?
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.<sup>1</sup>
  - (d) It turns out that in any connected graph (or multigraph) in which each node has even degree, there must exist an *Euler tour*, which is a tour that travels on each *edge* exactly once. Furthermore, this can be found efficiently by a simple algorithm. (You have probably seen this fact in a prior class, but in any case you are not being asked to prove this fact for this assignment.) Use this together with part (c) to get a 1.5 approximation to the TSP.

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.

---

<sup>1</sup>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).