

# 15-451 Algorithms, Fall 2005

Homework # 6

due: Mon-Tues, November 21,22, 2005

---

## Ground rules:

- This is an oral presentation assignment. You should work in groups of three. At some point before **Saturday, Nov 19 at midnight** your group should sign up for a 1-hour time slot on the signup sheet on the course web page. *Note:* If you prefer to sign up for an earlier date (e.g., because you will be going out of town) then feel free to contact your TA and arrange a time.

## 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 grade between 0.00 and 1.00, and that requirement  $r_i$  now states “the sum of your grades 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 grades 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. [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 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. 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.

3. [The List-Update Problem] Suppose we have  $n$  data items  $x_1, x_2, \dots, x_n$  that we wish to store in a linked list in some order. Let's say the cost for performing a *lookup*( $x$ ) operation is \$1 if  $x$  is in the head of the list, \$2 if  $x$  is the second element in the list, and so on.

For instance, say there are 4 items and it turns out that we end up accessing  $x_1$  3 times,  $x_2$  5 times,  $x_3$  once, and  $x_4$  twice. In this case, in hindsight, the best ordering for a linked list would have been  $(x_2, x_1, x_4, x_3)$  with a total cost of \$21.

The *Move-to-Front* (MTF) strategy is the following algorithm for organizing the list if we don't know in advance how many times we will access each element. We begin with the elements in their initial order  $(x_1, x_2, \dots, x_n)$ . Then, whenever we perform a *lookup*( $x$ ) operation, we move the item accessed to the front of the list. Let us say that performing the movement is free. For instance, if the first operation was *lookup*( $x_3$ ), then we pay \$3, and afterwards the list will look like  $(x_3, x_1, x_2, x_4 \dots)$ .

- (a) Suppose  $n = 4$  and we use MTF starting from the order  $(x_1, x_2, x_3, x_4)$ . If we perform the following 4 operations:

$$\textit{lookup}(x_4), \textit{lookup}(x_2), \textit{lookup}(x_4), \textit{lookup}(x_2).$$

What does the list look like in the end and what was the total cost?

---

<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).

(b) Your job is to prove that the total cost of the MTF algorithm on a sequence of  $m$  operations (think of  $m$  as much larger than  $n$ ) is at most  $2C_{static} + n^2$  where  $C_{static}$  is the cost of the best static list in hindsight for those  $m$  operations (like in our first example). We will prove this in two steps.

i. First prove the somewhat easier statement that the cost of Move-to-Front is at most  $2C_{initial}$  where  $C_{initial}$  is the cost of the original ordering  $(x_1, x_2, \dots, x_n)$ .

*Hint:* If  $i < j$  but  $x_j$  is in front of  $x_i$  in the MTF list, let's say that  $x_j$  has "cut in line" in front of  $x_i$ . Now, imagine that each element  $x_i$  has a piggy bank with \$1 for everyone that is currently cutting in line in front of it.

ii. Now prove the  $2C_{static} + n^2$  bound.