

15-451 Algorithms, Fall 2005

Homework # 5

due: Tuesday November 8, 2005

Please hand in each problem on a separate sheet and put your **name** and **recitation** (time or letter) at the top of each sheet. You will be handing each problem into a separate box, and we will then give homeworks back in recitation.

Remember: written homeworks are to be done individually. Group work is only for the oral-presentation assignments. Also, for any question that asks for an algorithm, you should also argue why the algorithm is correct.

Problems:

- (30 pts) 1. [Fair carpooling] The n employees of Algorithms-R-Us sometimes carpool to work together. Say there are m days, and S_i is the set of people that carpool together on day i . 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 p is in are a_1, a_2, \dots, a_k . Person p should really have to drive $\frac{1}{a_1} + \frac{1}{a_2} + \dots + \frac{1}{a_k}$ 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 than this many times.

For example, say that on day 1, Alice and Bob carpool together, and on day 2, Alice, Carl, and Dilbert carpool together. Alice's fair cost would be $\lceil 1/2 + 1/3 \rceil = 1$. So Alice driving both days would not be fair. Any solution except that one is fair.

- (a) Prove that there always exists a fair solution.
- (b) Give a polynomial-time algorithm for computing a fair solution.

Hint: Try to model the problem using network flow in such a way that part (a) falls out directly from the integrality theorem for network flow, and part (b) just follows from the fact that we can solve max flow in polynomial time. So, it all boils down to coming up with the right flow graph to model the problem.

- (30 pts) 2. [Graph Searching] Let G be a directed graph represented using an adjacency list. So, each node $G[i]$ has a list of all nodes reachable in 1 step from i (all out-neighbors of i). Suppose each node of G also has a value: e.g., node 1 might have value \$100, node 2 might have value \$50, etc.

Give a fast algorithm that computes, for every node, the highest value that can be reached from that node (i.e., that you can get to by some path from that node). For

instance, if G is strongly-connected, then for every node this will be the maximum value in the entire graph. Your algorithm should run in time $O(m + n)$ or $O(m + n \log n)$.

- (20 pts) 3. [Graduation] Cranberry-Melon University has n courses. In order to graduate, a student must satisfy several requirements. Each requirement is of the form “you must take at least k courses from subset S ”. The problem is to determine whether or not a given student can graduate. The tricky part is that any given course cannot be used towards satisfying multiple requirements. For example if one requirement states that you must take at least two courses from $\{A, B, C\}$, and a second requirement states that you must take at least two courses from $\{C, D, E\}$, then a student who had taken just $\{B, C, D\}$ would not yet be able to graduate.

Your job is to give a polynomial-time algorithm for the following problem. Given 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 ”), and given a list L of courses taken by some student, determine if that student can graduate. In particular, show how you can solve this using network flow.

- (20 pts) 4. [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 .)

Show how to solve this using linear programming.