15-451 Algorithms, Fall 2009

Homework # 5 **due: Tuesday November 3, 2009**

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.

Problems:

(35 pts) 1. [Fair carpooling] The n employees of the Algo Rhythms music downloading service 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. Since people would rather not drive, they want the work of driving to be divided as fairly as possible. Your task in this problem is to give an algorithm to do this efficiently.

The fairness criterion is the following: Say that person p is in some k of the sets, which have sizes $n_1, n_2, \ldots, n_k$, respectively. Person p should really have to drive $\frac{1}{n_1} + \frac{1}{n_2} + \cdots + \frac{1}{n_k}$ 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 from the sets S_i.

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. [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_i courses from subset S_i". 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 $r_1, r_2, \ldots, r_m$ (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.

(35 pts) 3. [Realizing degree sequences] You are the chief engineer for Graphs-R-Us, a company that makes graphs to meet all sorts of specifications.

(a) A client comes in and says he needs a 4-node directed graph in which the nodes have the following in-degrees and out-degrees:

$$d_{1,in} = 0, d_{1,out} = 2$$
$$d_{2,in} = 1, d_{2,out} = 2$$
$$d_{3,in} = 2, d_{3,out} = 1$$
$$d_{4,in} = 3, d_{4,out} = 1$$

Is there a directed graph, with no multi-edges or self loops, that meets this specification? If so, what is it?

(b) What about a 3-node graph (again with no multi-edges or self loops) with these in-degrees and out-degrees?

$$d_{1,in} = 2, d_{1,out} = 2$$
$$d_{2,in} = 2, d_{2,out} = 2$$
$$d_{3,in} = 1, d_{3,out} = 1$$

(c) This type of specification, in which the in-degrees and out-degrees of each node are given, is called a *degree sequence.* The question above is asking whether a given degree sequence is *realizable* — that is, whether there exists a directed graph having those degrees.

Find an efficient algorithm that, given a degree sequence, will determine whether this sequence is realizable, and if so will produce a directed graph with those degrees. The graph should not have any self-loops, and should not have any multi-edges (i.e., for each directed pair (i, j) there can be at most one edge from i to j, though it is fine if there is also an edge from j to i). Hint: as if you couldn't have guessed - think network flow!