15-451 Algorithms, Fall 2012

Homework # 5 **due: Tuesday November 6, 2012**

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:

(25 pts) 1. [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:

$$\begin{aligned} d_{1,in} &= 0, & d_{2,in} &= 1, & d_{3,in} &= 2, & d_{4,in} &= 3 \\ d_{1,out} &= 2, & d_{2,out} &= 2, & d_{3,out} &= 1, & d_{4,out} &= 1 \end{aligned}$$

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?

$$\begin{aligned} d_{1,in} &= 2, & d_{2,in} &= 2, & d_{3,in} &= 1 \\ d_{1,out} &= 2, & d_{2,out} &= 2, & d_{3,out} &= 1 \end{aligned}$$

(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: think network flow.

(25 pts) 2. [Multicommodity Flow] The *multicommodity flow* problem is just like the standard network flow problem except we have p sources $s_1, \ldots, s_p$ and p sinks $t_1, \ldots, t_p$. 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?

(25 pts) 3. [Graduation revisited] Cranberry-Melon University has switched to a less draconian policy for graduation requirements than that used on Homework 4. As in Homework 4, 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 4, 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/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.

(25 pts) 4. [Adversarial Shortest Paths] You're given an undirected graph with non-negative edge lengths. You also have a start vertex and a collection of goal vertices. You're interested

in finding the shortest path from the start vertex to a goal vertex, with one catch. (If there were no catch you could use Dijkstra's or the Bellman-Ford shortest path algorithm, which you should review because they will be useful in solving this problem.)

The catch is that there's an adversary watching your progress through this graph. And when you're at a vertex, the adversary can decide to temporarily block one of the exiting edges from your vertex. When you move along an unblocked edge, the adversary then removes the previous block and sets up a new block on one of the edges from your new vertex.

Give an algorithm that takes such a graph and determines if you can make it to a goal or not, and if so, it tells you how long it will take you to get there (assuming the adversary makes it take as long as he can.) Your algorithm should run in time polynomial in n and m on a graph of n vertices and m edges.