15-451 Algorithms, Fall 2004

Homework # 5 **due: Tuesday November 9, 2004**

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:

(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 $a_1, a_2, \ldots, a_k$. Person p should really have to drive $\frac{1}{a_1} + \frac{1}{a_2} + \cdots + \frac{1}{a_k}$ 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 and Carl carpool together. Alice's fair cost would be $1/2 + 1/2 = 1$. 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.

(Note: this is not an online problem. We are assuming we know all the sets $S_1, \ldots, S_m$ up front).

(25 pts) 2. [Options pricing] An *option* (specifically, an "American call option") gives the holder the right to purchase some underlying asset (e.g., one share of IBM) at some specified exercise price (e.g., \$100) within some specified time period (e.g., 1 month). The value of an option depends on the current price of the asset, the exercise price of the option, the length of the time period, and one's beliefs about how the asset's price is likely to behave in the future.

For example, to take an easy case, suppose we have an option to buy one share of IBM at \$100 that expires right now. If IBM is currently going for \$105, then the value of this option is \$5. If IBM is currently going for \$95, then the value of this option is \$0 (we wouldn't want to exercise it). However, suppose the option expires tomorrow, and suppose we have some simple model of how IBM shares behave. E.g., perhaps our model says that each day, with probability 1/4 the share price goes up by \$10, and with probability 3/4 the share price goes down by \$5. In that case, if IBM is currently worth \$95, then the value of this option is \$1.25 because that is our expected gain if we use the optimal strategy "wait until tomorrow, and then exercise the option if IBM went up" (our expected gain is $\frac{1}{4} \times 5 + \frac{3}{4} \times 0$). If IBM is currently worth \$105, then the value of the option is \$5 (and our optimal strategy is to exercise the option right away).

Let us assume stock prices are integers between 0 and B. Suppose we also have have a probabilistic model p_{ij} for how the stock behaves: specifically, if the stock has value i on day t, then p_{ij} is the probability that the stock will have value j on day $t+1$. So, for each i, $\sum_j p_{ij} = 1$.

Describe a dynamic-programming algorithm to calculate the value of an option of exercise price X that expires T days in the future, given that the current price of the stock is S. The running time of your algorithm should be $O(B^2T)$.

(30 pts) 3. [Set Cover] The *set-cover* problem is the following: Given n points labeled $1, 2, \ldots, n$, and m subsets of these points $s_1, s_2, \ldots, s_m$, and an integer k, is it possible to cover all the points using only k of the sets s_i? (I.e., every point should be in at least one of these k sets). For instance, if $n = 6$ and the sets are $s_1 = \{1, 2, 3\}$, $s_2 = \{1, 4\}$, $s_3 = \{2, 5\}$, and $s_4 = \{3, 6\}$, then it is possible to cover all the points with three sets (namely, s_2, s_3, and s_4) but not with two of them.

(a) Prove that the set-cover problem is NP-complete by reducing from the vertex-cover problem (vertex cover is defined in Chapter 34). Also, say why set-cover is in NP.

(b) Show how to reduce the search version of the set-cover problem to the decision version. That is, show how you can use an oracle for the set-cover decision problem defined above to actually *find* an optimal set cover (a cover that uses the fewest sets).

(c) The *fractional set cover* problem is like the set-cover problem, except rather than choosing or not choosing each set, you instead assign each set a fraction between 0 and 1. The requirement is that for each point i, if you add up the fractions assigned to sets that cover that point, you must get a total of ≥ 1. The goal is to minimize the sum of all the fractions.

For example, if there are 3 points and the sets are $s_1 = \{1, 2\}$, $s_2 = \{2, 3\}$, and $s_3 = \{3, 1\}$, then the best fractional set cover is to assign each set the fraction 1/2; this covers all the points and the total sum is $1/2 + 1/2 + 1/2 = 3/2$. In contrast, the best standard set cover requires 2 sets. (Note that a standard set cover is also a legal fractional cover, but not necessarily vice-versa.)

Show how to solve the fractional set cover problem using *linear programming*. Be sure to specify what the variables are, what the constraints are, and what you are trying to minimize or maximize.

(15 pts) 4. [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.

Show how to solve this using linear programming.