15-451 Algorithms, Spring 2007

Homework # 5 **due: Tuesday April 10, 200**

Please hand in each problem on a **separate** sheet and put your **name**, **andrew id** and **recitation** (time or letter) at the top of each page. You will be handing each problem into a separate box, and we will then give homeworks back in lecture. If a problem takes up more than one sheet of paper, you must **staple** all sheets of paper for that problem together.

If you DO NOT follow these instructions exactly, you will lose up to 30pts.

Remember: Group work is allowed, however each student must hand in a separate write-up. Moreover, you must explicitly state where you got your ideas from. The answers should always be in your own words.

Note: For every algorithm that you give briefly explain its correctness and runtime. For every NP-hardness reduction, you can only reduce from problems given in class or the class notes (this does not include the textbooks).

---

1 CMU Graduation (50 pts)

Central Metropolitan University has a list of m graduation 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". A student is allowed to 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.

Suppose Lazy is an incoming freshman interested in finding the *minimum* number of courses that he (or she) needs to take in order to graduate. Specifically, consider the following decision problem: given n courses labeled $1, 2, \ldots, n$, given m subsets of these courses $S_1, S_2, \ldots, S_m$, and given an integer k, does there exist a set S of at most k courses (for Lazy to take) such that $|S \bigcap S_i| \geq k_i$ for all S_i.

Problem 1. Show how you could use a polynomial-time algorithm for the above decision problem to also solve the search-version of the problem in polynomial time (i.e., actually find a minimum-sized set of courses to take).

Problem 2. Using the knowledge you have gained during your time at Central Metropolitan University prove that the decision problem above is NP-complete. Recall that you can only reduce from problems given in class or the class notes. **Hint:** consider reducing from Vertex Cover.

Problem 3. We could define a *fractional* version of the graduation problem by imagining that in each course taken, a student gets a score between 0.00 and 1.00, and that requirement r_i now states "the sum of your scores 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 scores 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.

Now suppose that the new dean Dr. Evil of Central Metropolitan University changes the graduation policy such that now the same course *cannot* satisfy more than one requirement. 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.

Problem 4. All the students are now desperate to work out if they can still graduate! More formally, the problem is as follows: 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, decide whether the student is eligible to graduate. Either give a polynomial time algorithm to solve this problem, or show that the problem is NP-complete.

2 (Generalized) Shortest Supersequence (25 pts)

Problem 5. In homework 3 we introduced the Shortest Supersequence problem for 2 strings. Here we generalize the problem to n strings as follows.

Given n strings $\{x^i = x^i_1 x^i_2 \dots x^i_{m_i} \mid m_i \geq 1,\ i = 1, \dots, n\}$ over some alphabet, a *common supersequence* of x^i (for all $i = 1, \dots, n$) is a string z such that each x^i appears in z as a subsequence (as defined in homework 3).

For example, given the strings $\{abc, bca, dbf\}$ possible supersequences include $abcadbf$ and $adbcaf$.

The (generalized) shortest supersequence problem is, given a set of n strings $X = \{x^i = x^i_1 x^i_2 \dots x^i_{m_i} \mid m_i \geq 1,\ i = 1, \dots, n\}$ and an integer $K > 0$, to decide whether there is a supersequence of X of length at most K.

(a) Consider the following reduction from the Vertex Cover problem. Given a graph $G = (V, E)$ and integer L (the Vertex Cover instance), construct an instance of (generalized) Shortest Supersequence as follows: for each edge $(u, v) \in E$, create two strings, uv and vu and add them to the set X of strings in the Shortest Supersequence instance. Set $K = L + |V|$ and let the final instance be (X, K). Using this reduction, show that the (generalized) Shortest Supersequence problem is NP-complete.

Recall, the Vertex Cover problem is as follows: given an undirected connected graph $G = (V, E)$ and an integer $L > 0$, you want to decide whether there is a subset $S \subseteq V$ of size $|S| \leq L$ such that for every edge $(u, v) \in E$, either $u \in S$ or $v \in S$, or both.

Hint: draw a small example

(b) Suppose that in the reduction above instead of including both uv and vu, you include only uv in the Shortest Supersequence instance. For example, suppose that you only include the direction preserving the alphabetical order. Show why you can't use this reduction to show that (generalized) Shortest Supersequence is NP-complete.

3 More Flow Problems (25 pts)

Here are some variations of the Maximum Flow problem. Please give short answers.

Problem 6. Suppose each vertex has a capacity on the maximum flow that can enter it. Show that we can reduce this to the original Max Flow problem to solve it.

Problem 7. Suppose that each edge e also has a lower bound $l(e)$ on the flow it must carry. Show how to find a Maximum Flow with these restrictions by reducing the problem to Linear Programming.

Problem 8. Suppose the outgoing flow from each node u is not the same as the incoming flow but is smaller by a factor of $1 - \varepsilon_u$ where ε_u is the loss coefficient associated with node u. Show how to find a Maximum Flow in this scenario by reducing the problem to Linear Programming.