CSC373H1S 2014 **Assignment 3**
Due: Thursday, March 20th, 10PM Total Score: 10 points

Please follow the instructions provided on the course website to submit your assignment. You may submit the assignments in pairs. Also, if you use *any* sources (textbooks, online notes, friends) please cite them for your own safety.

For each of the problems below you will be asked either to write a linear program or prove that the problem is NP-Complete.

Questions

1. **Oh hi Mark!**

   Here is a generalization of the roommate-chore matching question from the previous assignment. As before, there are n roommates and m chores, and we want to distribute the work from each of the chores across the roommates. For simplicity, we will label each of the m chores with a unique integer from $\{1, 2, \ldots, m\}$. This time, however, for each chore i you will receive a positive real number c_i representing how many hours of work this chore will require each week.

   For any chore i the n roommates can split up the work arbitrarily as long as the c_i hours of work will be completely finished. For each j, $1 \leq j \leq n$, the jth roommate is defined by $r_j = (P_j, z_j, h_j)$. The set $P_j \subseteq \{1, 2, \ldots, m\}$ is a collection of chores, representing the chores that this roommate prefers to do. However, each roommate *is willing* to do the chores not in his set of preferred chores, but you must sweeten the deal with some hard cash: for each hour of work roommate j spends on a chore not in P_j, you must pay him z_j dollars (for some positive real number z_j). You do not have to pay a roommate any amount of money for them to do a preferred chore. Finally, h_j is a positive real number representing how many hours of work roommate r_j is willing to do in total (that is, including the contributions from every chore that they are working on).

   Greedy Roommates

   Input: Two positive integers m, n. For each i, $1 \leq i \leq m$, a positive integer c_i representing how many hours of work the ith chore will require this week. For each j, $1 \leq j \leq n$, a subset of preferred chores P_j for roommate r_j, the roommate's hourly price z_j for doing chores they do not prefer, and the number of hours h_j they are able to spend on doing chores in total.

   Output: The minimum amount of money that you need to pay each of your roommates in order to complete all of the hours of work for each chore, or infeasible if it is impossible to assign your roommates to chores within the given constraints.

   Give a linear programming formulation which solves the Greedy Roommates problem.

2. **Talk about MY generation**

   The *weighted generation* problem is defined as follows. You are given a finite set of n points P, with two distinguished points s and t in P (and note that $s \neq t$). As well, you are given a set of *triples* $T \subseteq P^3$, where a *triple* consists of any three (ordered) points drawn from P: for example, the triple (x, y, z) is distinct from the triple (x, z, y). There is also a weight function $w : T \to \mathbb{R}^+$, which weights each triple with a positive real number.

   Given P and T, we say that a point $z \in P$ can be *generated* from T if either:

   (a) $z = s$ (that is, s is always generated), or

   (b) there is a triple $(x, y, z) \in T$, and both x and y can be generated.

The *cost* of generating a point z from x, y using the triple $\ell = (x, y, z)$ is

$$cost(z) := \min\{cost(x), cost(y)\} + w(\ell).$$

The cost of generating s is 0. There may be many ways of generating each point in P, and in this problem you will be interested in finding the way of generating the point t with the minimum overall cost.

Weighted Generation

Input: A set P of n points, with two distinguished points s and t. A set of triples $T \subseteq P^3$, along with a function $w : T \to \mathbb{R}^+$ weighting each triple with a positive real number.

Output: The minimum cost necessary for generating the point t.

Give a linear programming formulation which solves the Weighted Generation problem.

3. **Take that, you stupid string!**

In this problem we will be considering strings of bits. If y is a string of bits (for example, $y = 00001$) then we will use the notation $y[i]$ to denote the ith character in y. (In the example, $y[1] = 0$ and $y[5] = 1$.) We define the *Hamming weight* of a bit string y, denoted $|y|$, to be the number of 1s in the string y. In the example above the Hamming weight of the string y is $|y| = 1$.

Hitting String

Input: A positive integer k. A set A of strings of bits, where each string in A has length n.

Output: Return 1 if and only if there is a string x of n bits, with $|x| \leq k$, such that for every string $y \in A$ there exists an index i where $x[i] = y[i] = 1$. (Such a string x is called a *hitting string*.)

Part a.

Show that this problem is in NP by a reduction to one of the NP-Complete problems discussed in the lectures or tutorials. Then show that it is NP-Hard by a reduction from the Vertex Cover problem.

Part b.

The *search version* of the Hitting String problem instead asks for you to actually return the hitting string x, if it exists (rather than simply return 1). Give a polynomial time search-to-decision reduction for the Hitting String problem. That is, assuming that you have access to a polynomial time algorithm H which solves the Hitting String problem, give a polynomial time algorithm which solves the search version of the Hitting String problem.

4. **Puzzles are fun!**

Let m and n be positive integers and consider the following game – called Pebble Up – played on an $n \times n$ grid. We will denote the square in the ith row of the grid and the jth column of the grid by the pair (i, j). For each $1 \leq i, j \leq n$, the square (i, j) will be given a *colour* (either black or white) and a *number* (some positive integer $k \leq m$). The grid, along with a given colour and integer for each square, will be referred to as a *game board*.

Pebble Up is played on the game board as follows. You have a bag of black and white pebbles, and for each integer k between 1 and m, you must choose to place either a black or a white pebble on all of the squares labelled with the integer k (so, squares with different numbers can have different coloured pebbles, but if two squares have the same number then they must have the same coloured pebble). After choosing a coloured pebble for each integer k, the game moves on to the next phase.

For each square (i, j), with $1 \leq i, j \leq n - 1$, examine the set of four squares

$$S_{ij} = \{(i, j), (i+1, j), (i, j+1), (i+1, j+1)\}.$$

This set of four squares is said to be *winning* if at least one of squares has the same colour as the pebble lying on it. You *win* the game if every set S_{ij} is winning.

Figure 1 depicts a game board (with $n = 3, m = 5$) along with two configurations of pebbles. The middle configuration is a winning configuration: any set S_{ij} of four squares has at least one pebble which has the same colour as its square. If we switch the colour of the "5" pebble from white to black, we get the configuration on the right. This is not a winning configuration since the set S_{12}, consisting of the four squares in the top-right corner of the game board, has each square and pebble differing in colour.

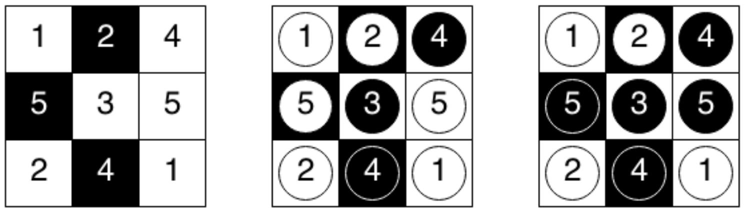

Figure 1: A game board and two possible configurations of pebbles

Pebble Up

Input: Two positive integers m, n, and for every (i, j) a colour b_{ij} (either black or white) and a positive integer $k_{ij} \leq m$.

Output: 1 if and only if it is possible to win the Pebble Up game on the given game board.

Prove that Pebble Up is NP-Complete.