CSC373H1S 2014 | **Programming Project**
Due: Friday, April 4th, 2013 | Total Score: 15 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 question below, write a program to solve it. Each program can be written in C/C++, Java, or Python. Each problem has 20 test cases associated with it — four are posted along with this assignment, and 16 will remain hidden. The four test cases for each problem are posted on the course website in the assignment section. For the first three problems, you will receive five marks for each test case that your program passes within a time limit. For a test case you will lose all five marks if your program is either incorrect or if it takes much longer than the time limit.

For the fourth (bonus) problem, there are again 20 test cases (four visible and 16 hidden). However, this problem is much harder to solve, and so now an additional parameter will be involved in the marking: *approximation*. Each of the output test cases will be marked based on how fast your algorithm solves the problem, and also how close to the optimal solution it is, where "closeness" will be measured by minimizing

$$\frac{ALG}{OPT},$$

where ALG is the value output by your algorithm and OPT is the value of the optimal solution. Greater weight will be given to approximation than to speed — a correct, but longer, computation will obtain more points than a fast but incorrect algorithm.

Questions

1. **An Odd (Ad-Hoc) Hop**

   Imagine the following scenario: you have n agents, each holding a laptop containing a wireless transmitter. For the ith agent we are given a position (x_i, y_i), and a radius r_i which is the wireless communicating range of the laptop. That is, we imagine that the wireless range of laptop i is a circle centered at (x_i, y_i) with radius r_i. We say that the laptops i and j can communicate if their wireless ranges overlap.

   Of course, not every laptop can communicate with every other laptop, but laptops can send messages by using intermediate laptops as routers. Define the *hop distance* $h(i, j)$ to be the *minimum* number of intermediate laptops used to send a message from laptop i to laptop j. For example, if two laptops can communicate directly, the hop distance between them is 1. In this problem, given a set of n agents with their positions and wireless ranges, you will have to compute the hop-distance from the first laptop to every other reachable laptop. If agent i is not reachable from agent j then the hop distance $h(i, j)$ will be set to 0.

   Input Format The input file format will be as follows. The first line will contain an integer n, which is the number of agents. What follows will be n lines, where the ith line contains three real numbers x_i, y_i, r_i corresponding to the x and y coordinate of the ith agent, along with his or her wireless range.

   Output Format The output format will have n lines. On the ith line will be an integer, representing the hop distance $h(1, i)$ (the minimum number of intermediate laptops necessary for the first agent to communicate with agent i).

2. **Stupid Monkey!**

   You know how it goes — you are out on an hike in the jungle, one thing leads to another, and now you are dealing with a monkey who has stolen all of your belongings and is hurling them forcefully to the ground from a tree top. The monkey is at height h above you, and will run back and forth across the top of the trees to drop your belongings down on top of you! The total distance that the monkey can run back and forth on top of the trees is length ℓ, and if the monkey drops one of your belongings at time t from position x, it will take h time steps to travel from the top of the tree top and land on the ground. If you are standing at position x when your belonging lands on the ground (height 0), you will catch it! Fortunately, the monkeys in the jungle are very predictable, and you know in advance at which times and from what positions the monkey will drop different pieces of your belongings. Unfortunately, you move at a bounded speed: in one time step, you can move at most s positions either to the left or the right. Your job is to catch as many of the items that the monkey will hurl as possible, starting from position $x = 0$ at time $t = 0$.

   Input Format The input file will be formatted as follows. The first line will contain a positive integer ℓ, representing how far the monkey can travel from position 0. To be specific, the monkey can travel to any x-coordinate in the set $\{0, 1, 2, \dots, \ell\}$. The second line contains a positive integer h, which is how high the monkey is above you. The third line will contain a positive integer s which is your top speed: in 1 time step you can move from your current position x to anywhere in

$$\{x - s, x - s + 1, \dots, x - 1, x, x + 1, \dots, x + s - 1, x + s\}\,.$$

   The fourth line contains a positive integer, n, representing how many items the monkey stole from you. The next n lines of the file are defined as follows: Each line will contain two positive integers x_i, t_i. The first integer, x_i, is the x position from which the monkey will drop the ith item. The second integer, t_i, is the time at which the monkey will drop the ith item. Assume that the monkey cannot be in two places at once — he cannot drop an item at the same time from two different positions.

   Output Format The output will consist of a single integer, which is the maximum number of items that you can catch.

3. **Stacking Boxes**

A *shipping manifest* of size $m \times n$ is an $m \times n$ array M in which each entry is drawn from $\{A, B, -\}$, with the following properties:

(a) If $M[i, j] \neq -$ then $M[k, j] \neq -$ for all $i < k \leq m$.

(b) For all $1 \leq j \leq n$ there is an i such that $M[i, j] \neq -$.

We imagine a shipping manifest as encoding a way of stacking some collection of boxes, where each box is labelled with either A or B (and a $-$ denotes that the space does not have a box). If you are given an $m \times n$ shipping manifiest M, you can perform the following manipulations (which respectively correspond to adding, removing, or swapping boxes):

Add Boxes You may add a box to the top of a pile: If $M[i, j] = A$ or B and $M[i - 1, j] = -$ then set $M[i - 1, j] = A$ or B. This has a cost of 1.

Remove Boxes You may remove a box from any pile: If $M[i, j] = A$ or B then you can *remove* the box, and all of the boxes above it will fall down one block. This has a cost of 1.

Swap Boxes If $M[i, j] = A$ or B, you may swap $M[i, j]$ for the other box. This has a cost of 1.

Input Format The input format will be a file defined as follows. The first line will contain m, the maximum number of rows. The second line will contain n, the maximum number of columns. What follows will be $2m$ lines, each with n characters. The first m lines will encode an $m \times n$ shipping manifest S, and the next m lines will encode another $m \times n$ shipping manifest T.

Output Format The output format will be a single integer, k, corresponding to the minimum number of operations needed to transform the shipping manifest for S into the manifest T.

4. **(BONUS) Business! (It's like a really neat video game!)**

   As the heartless CEO of FacelessCorp, you are seeking to expand your company's profile into various exploitative goods and services. Of course, there is only one way to do so — the ruthless takeover and liquidation of other companies!

   To be specific, there are a set $\mathcal{S}$ of n services that you would like to be in FacelessCorp's profile. Each service $s \in S$ has a weight w — this represents the cost to the company if we do not acquire service S.

   There is also a collection $\mathcal{C}$ of m companies, where each company $C \in \mathcal{C}$ offers a subset $C \subseteq S$ of services. Moreover, associated with each company C is a cost $c(C)$ (a positive real), representing the cost of purchasing the company C.

   The goal is simple: you will be given a set of services $\mathcal{S}$, as well as a collection of companies $\mathcal{C}$ (each represented by a list of their offered services), along with the weights of each service and the cost of purchasing each company. You will choose a subset $\mathcal{C}' \subseteq \mathcal{C}$ of companies to purchase, with the purpose of minimizing

   $$cost(\mathcal{C}') = \sum_{C \in \mathcal{C}'} c(C) + \sum_{\substack{u \in S \\ \forall C \in \mathcal{C}', u \notin C}} w(u).$$

   In other words, this is the cost of purchasing the companies in $\mathcal{C}'$, plus the weight of each service not offered by any company in $\mathcal{C}'$.

   Input Format The input file will be given in the following format. The first line of the file will contain a single integer n, which is the number of services in $\mathcal{S}$. The second line of the file will contain an integer m, which is the number of companies in $\mathcal{C}$. The next n lines of the file each contain a single positive real number w_i, corresponding to the cost of not obtaining any company which provides service i. The final m lines of the file will contain between 1 and $n+1$ numbers: the first number is a positive real c_i, corresponding to the cost of purchasing company i, and the rest of the numbers on the line consist of distinct positive integers less than or equal to n, corresponding to which services the ith company provides.

   Output Format The output format is a single positive real number $cost(\mathcal{C}')$, which is the cost of your purchasing profile.