

It's a Zoo

USC Programming Contest, Fall 2008

September 20, 2008

Sponsored by VSoE, LanguageWeaver, Google, Electronic Arts, Sun Microsystems,
and Zero G Games

If you think that we humans are the only ones having computational problems, think again. Hedgehogs, monkeys, kangaroos, and many others, all have to solve very complex algorithmic tasks. And we aren't even talking about all the graphic design that butterflies seem to be doing. The difference between animals and us? We humans can actually learn how to program and write programs that solve our computational problems. Hedgehogs really haven't figured out the whole "typing" thing yet. So in an act of inter-species altruism, we're going to spend some time solving animals' computational problems. Be forewarned! Some of your programs will decide life-or-death situations.

You can solve or submit the problems in any order you want. When you submit a problem, you submit your source code file using `PC2`. Make sure to follow the naming conventions for your source code and the input file you read. Remember that all input must be read from the file of the given name, and all output written to `stdout`. You may not use any electronic devices besides the lab computer and may not open any web browser for whatever reason.

One thing you should know about this contest is that the judging is done essentially by a `diff` of the files. That means that it is really important that your output follow the format we describe — if you have the wrong number of spaces or such, an otherwise correct solution may be judged incorrect. Consider yourselves warned!

Another warning that seems to be in place according to our experience: all of our numbering (input data sets, classes, etc.) always starts at 1, not at 0.

And a piece of advice: you will need to print floating point numbers both in scientific notation with four digits after the decimal dot, and rounded to two decimals. Here is how you do that in C:

- Scientific Notation: `printf ("%E", r);` where `r` is the variable you want to print.
- Two Decimals: `printf "%.2f", r);`

In Java, the syntax is:

- Scientific Notation: `System.out.print ((new java.text.DecimalFormat("0.####E0")).format(r));`
- Two Decimals: `System.out.print ((new java.text.DecimalFormat("#.##")).format(r));`

[Including this page, the problem set should contain 8 pages. If yours doesn't, please contact one of the helpers immediately.]

Problem A: Isn't It Funny How a Bear Likes Honey?

File Name: bear.cpp|bear.java

Input File: bear.in

Description

One of the famous stories in A. A. Milne's "The Tales of Winnie-the-Pooh" revolves around Pooh's intent to steal some honey from a beehive high up in a tree. After failed attempts at climbing the tree, he decides to float up to the beehive with a balloon. Never mind the fact that the story involves a balloon inflated with regular air, we are going to calculate here whether a collection of helium-filled balloons is going to be enough to lift up Pooh.

In order to calculate that, we will give you Pooh's weight as well as the radius of the balloons. We assume the balloons are perfectly circular. We will ignore the weight of the balloons themselves and strings. Here is what you need to know about flotation: each liter of helium (1000cm^3) is enough to lift one gram of weight. As a courtesy reminder, the volume of a sphere of radius r is $\frac{4}{3}\pi r^3$.¹ (We will make sure that in the inputs, there will be no case where the buoyancy *exactly* compensates for the weight, i.e., no "ties". To avoid rounding problems, we will guarantee that either the buoyancy will fall short of the required one by at least 0.001, or will exceed the required one by at least 0.001.)

Input

The first line is the number K of input data sets, followed by the K data sets, each of the following form:

The first line contains an integer b and a floating point number w . $b \geq 0$ is the number of balloons Pooh is using, and w is Pooh's weight (in grams).

This is followed by b lines, each containing one floating point number $r_i \geq 0$. This is the radius of the i^{th} balloon (in cm).

Output

For each data set, output "Data Set x :" on a line by itself, where x is its number. On the next line, output "Yes" or "No", depending on whether the balloons together will be able to lift Pooh. Each data set should be followed by a blank line.

Sample Input/Output

Sample input bear.in

```
2
2 5000.0
0.0
126.31
5 10000.0
78.0
78.0
78.0
78.0
78.0
```

Corresponding output

```
Data Set 1:
Yes

Data Set 2:
No
```

¹Hint: Make sure you use an accurate approximation of π . Plugging in 3.14 might not be a good idea.

Problem B: The Hare and the Hedgehogs

File Name: hare.cpp|hare.java

Input File: hare.in

Description

There is a famous German fable about the hare and the hedgehog. The hare keeps boasting about how fast he is, so the hare and hedgehog agree to a race across a field. As soon as the race starts, the hare takes off racing, but when he gets to the end, the hedgehog is already there. So he races back even faster, but the hedgehog is already there. He keeps racing back and forth at increasing speeds, and meets the hedgehog at the other end every time. Finally, he collapses and dies. Of course, the hedgehog won because the one at the other end of the field was actually his wife. When the story is told to children, the implication usually seems to be either that if you boast, you deserve to be shown up, or how effective teamwork can be. Unfortunately, the real message the story conveys is that it's easy to win if you cheat.

Here, we will look at a more complicated race between a hare and a number of hedgehogs. The course will consist of several "legs", each a straight line. This will be given by a sequence of points in the plane $(x_1, y_1), \dots, (x_n, y_n)$. The starting point of the race is the origin $(0, 0)$. The hare will run at a constant speed u along the straight line to (x_1, y_1) , then along a straight line to (x_2, y_2) , and so forth. The hedgehogs can move at a (different) given speed v , and can walk around any way they like. Points are scored as follows: If, at the moment that the hare reaches (x_i, y_i) ($1 \leq i \leq n$), there is a hedgehog at (x_i, y_i) , then the hedgehogs win a point. Otherwise, the hare earns a point. If they arrive at exactly the same time, the hare also wins the point. Your goal is to compute the best score that the hedgehogs as a team can accomplish.

Input

The first line contains the number K of data sets. This is followed by K data sets, each of the following form:

The first line of each data set contains two integer numbers n, h , and two floating point numbers u, v . $1 \leq n \leq 10$ is the number of legs to be run, $1 \leq h \leq 5$ is the number of hedgehogs on the field, u is the hare's speed, and v the hedgehogs' speed, both in m/s .

This is followed by n lines, each giving the coordinates for a point (x_i, y_i) as two floating point numbers. This is followed by $h - 1$ lines, each giving the initial coordinates (x'_i, y'_i) of hedgehog $i = 2, \dots, h$. Hedgehog 1 always starts at the origin. All coordinates will be in m .

Output

For each data set, output "Data Set x :" on a line by itself, where x is its number. On the next line, output the maximum number of legs the hedgehogs together can win. Each data set should be followed by one empty line.

Sample Input/Output

Sample input hare.in

```
2
4 2 1.0 0.1
1.0 0.0
0.0 0.0
1.0 0.0
0.0 0.0
0.75 0.0
3 1 1.0 0.2
1.0 0.0
-0.5 -3
0.2 0.2
```

Corresponding output

```
Data Set 1:
3
Data Set 2:
1
```

Problem C: Hermit Crabs

File Name: crabs.cpp|crabs.java

Input File: crabs.in

Description

On many beaches, you can see hermit crabs, which are quite interesting creatures. They are related to crabs, but have a soft belly, so they need to live inside empty shells from sea snails for protection. So what you see is a sea snail shell walking along the beach at brisk pace. When you approach it, the crab withdraws inside, and the shell looks just like a shell, except it has a big claw closing off the entrance. Like most animals, hermit crabs grow over time, which means that they have to move shells. If they don't find a suitable shell, they usually get eaten pretty quickly. If shells are scarce, hermit crabs sometimes even fight for them with each other. Here, we will see which hermit crabs survive over time as they outgrow their shells.

To simplify things a little bit, we assume the following. Each hermit crab i has an integer size $c_i \geq 0$. Because the crab grows, c_i increases by one every time unit. Each shell j also has an integer size $s_j \geq 0$, which of course does not change over time. A hermit crab i can live in shell j at time t if the hermit crab's size at time t lies between $s_j - D$ and s_j , for a given constant D . (If the shell is too large, then the crab cannot protect the entrance; if it is too small, it does not fit inside.) Initially, at time 0, each crab i lives in shell i . (We will make sure that the size fits.) The moment a hermit crab becomes too large for its current shell, it leaves the shell, and moves into the *biggest* currently unoccupied shell into which it can fit. If there is no such shell at that moment, it gets eaten by a bird. If multiple crabs are trying to move into the same shell at the exact same time, they fight, and only the largest crab interested in that shell survives. If another crab leaves a shell j exactly at the same time as crab i is looking for a shell, crab i can move into shell j . All inputs will be such that no two crabs will have exactly the same size, and no two shells will have exactly the same size.

Input

The first line contains the number K of data sets. This is followed by K data sets, each of the following form:

The first line contains four integers n, m, D, T . n is the number of hermit crabs, and m the number of shells, and they will satisfy $1 \leq n \leq m \leq 1000$. D is the maximum time a crab can spend in a shell (see description above), and T is the length of the observation period.

This is followed by n lines, each containing one integer c_i , the initial size of crab i (all c_i will be different). Next are m lines, each containing one integer s_j , the size of the j^{th} shell (all s_j will be different). We will make sure that $s_i - D \leq c_i \leq s_i$ for $i = 1, \dots, n$, i.e., shell i has the right size for crab i initially.

Output

For each data set, first output "Data Set x:" on a line by itself, where x is its number. Then, output, one number per line, and in increasing order, the list of all crabs still alive at time t . Each data set should be followed by a blank line.

Sample Input/Output

See next page.

Sample input crabs.in

```
2
3 4 100 133
300
310
120
350
360
155
450
2 3 10 17
13
21
16
23
32
```

Corresponding output

```
Data Set 1:
2
Data Set 2:
```

Problem D: Monkeys at Typewriters

File Name: monkeys.cpp|monkeys.java

Input File: monkeys.in

Description

You've probably all heard at some point that if you were to sit down enough monkeys at typewriters, and let them type for long enough, they would eventually type all of Shakespeare's works. The assumption is that the monkeys will hit a completely random sequence of keys. It's usually not clear what point people are trying to make when they bring the "monkeys at typewriters" up; sometimes, it is about the fact that randomness contains structure, or that there isn't really that much artistic about writing poetry and plays. People who are skilled at math instead use it to argue just how *extremely unlikely* it is that the monkeys would produce anything meaningful, or just how many monkeys, typewriters, and how much time one would need.

Here, we'll solve a related question: given a collection of words, and monkeys typing randomly until they hit the space bar for the first time, what's the probability that they'll accidentally write one of the given words? All words will consist only of lowercase letters. You will be given the probabilities of hitting each of the lowercase letters, as well as that of hitting the space bar. Those probabilities will sum up to 1. The assumption is that the monkeys will keep hitting random keys according to this distribution until they hit the space bar, and then stop typing. You will also be given the list of words, and your goal is to compute the probability of the monkey typing one of the given words. A monkey is considered as having typed a word if the sequence up to the space bar matches that word exactly.

Input

The first line contains the number K of data sets. This is followed by K data sets, each of the following form:

The first line contains three numbers n, m, s ($1 \leq n \leq 100, 1 \leq m \leq 26, 0 \leq s \leq 1$). n is the number of words we want the monkeys to produce, m is the number of letter keys on the typewriter, and s is the probability of hitting the space bar. This is followed by m lines, each containing a letter and a floating point number, specifying the probability of the monkey hitting that letter. The probabilities, including the space bar probability, will add up to 1. Finally, there will be n lines, each containing a word w_i . Each word w_i consists only of lower-case letters which are also available on the typewriter, and has between 1 and 20 characters. All the words will be distinct.

Output

For each data set, output "Data Set x :" on a line by itself, where x is its number. Then, on the next line, output the probability of typing at least one of the given words. Since these probabilities can be very small, you should output them using scientific notation with four decimals after the dot, e.g., $3.7602E-13$. See the front page of this problem set for a hint on how to do that. Each data set should be followed by an empty line.

Sample Input/Output

Sample input monkeys.in

```
2
2 2 0.5
a 0.5
b 0.0
abba
baa
2 2 0.4
a 0.5
b 0.1
abba
babbbb
```

Corresponding output

```
Data Set 1:
0.0000E+00

Data Set 2:
1.0020E-03
```

Problem E: Swamp Kangaroo

File Name: kangaroo.cpp|kangaroo.java

Input File: kangaroo.in

Description

Kangaroos are fascinating creatures. For one, the idea of carrying around their offspring in a pouch is very cute, and it reminds us of our own ways of transporting babies. For another, they can jump really far. That can be quite useful, in particular if you're a kangaroo stuck in a swamp with only small islands of land, and crocodiles swimming around. In that case, you'd rather not land in the water. Besides mere jumping strength, it's also useful to have some computational power to compute how exactly to use that jumping strength to get where you need to go. That's where — if you're a swamped Kangaroo — your USC programming buddies come in.

Here's how we model Kangaroo movement. Kangaroos can only move North-South or East-West; no other directions (such as diagonals). Kangaroos can jump any integer distance between 1 and 5 in one hop, which takes them one unit of time. However, after a longer jump, they have to rest before being able to jump again. Specifically, after jumping distance d , the Kangaroo has to rest $(d - 1)^2$ time units before being able to jump again. Also, if the next jump is in a different direction from the previous one, the Kangaroo takes an extra one time unit between the jumps to turn around.

The swamp will be described by a two-dimensional grid. Each entry is either water, denoted by a dot '.', or land, denoted by 'X'. Two locations will be marked with special symbols. 'K' denotes the initial position of the Kangaroo, and 'G' is the goal the Kangaroo wants to reach (both of these are of course land). You are to find the shortest time in which the Kangaroo could get to the goal (if at all). It does not matter if the kangaroo is tired when it arrives at its destination; it doesn't have to rest.

Input

The first line contains the number K of data sets. This is followed by K data sets, each of the following form:

The first line contains two number h, w ($1 \leq h, w \leq 30$). These give the height and width of the swamp map. This is followed by h lines, each containing exactly w characters. Each character is either a '.', an 'X', a 'K', or a 'G'. Together, these $h \times w$ characters describe the swamp.

Output

For each data set, output "Data Set x :" on a line by itself, where x is its number. Then, on the next line, output the minimum amount of time in which the Kangaroo can reach the goal position. If there is no way to reach the goal, then output "Impossible" instead. Each data set should be followed by an empty line.

Sample input kangaroo.in

Corresponding output

```
1
12 30
.....XXX.....XX.....X.X.XX.
XK....XXXXXXXXX..XX.....
X.....XXXXX....XX.X.X.X...X.
.....XXX.....XX.....
.....XX.....XX.
.....XX.
...XX...XX.....XX.
..XXXX.....X..
..XXXX.....
...XX.....XX...X...XXX..
.....XX...XX.X.X.X.XGX..
.....XX.....XXX..
```

```
Data Set 1:
64
```

Problem F: Zebra Herd

File Name: zebras.cpp|zebras.java

Input File: zebras.in

Description

Zebras are very social animals. Like other members of the horse family, they form groups that tend to stick together and hang out fairly regularly, though not exclusively. (Humans also come to mind in this respect.) Lately, researchers have been trying to understand just how the communities of zebras evolve over time, what triggers changes, and so forth. Of course, all they have to go by is observations of *where* the zebras are over time. From that, we'd like to figure out what are the most natural groups. The assumptions are that (a) if a zebra is part of a group, it tends to spend time close to others in that group, (b) if a zebra is not part of a group, it tends to spend time further away from others in that group, and (c) zebras don't change their group membership very often.

Let's make this more precise. You will be given a sequence of observations of zebras. For each observation time, you will have the exact location of each zebra. The distance between two zebras is exactly their Euclidean (straight-line) distance. We assume that there are exactly two groups of zebras in the herd, and will denote them by two colors. What we want to do is color each zebra either red or blue for each time step, expressing membership to one or the other group. To express assumption (c) above, we will assess a penalty of some given number c every time a zebra changes colors. To express assumptions (a) and (b), we look at the distance $d(i, j)$ between every pair i, j of zebras. If i and j are of the same color, then we assess a penalty of $a \cdot d(i, j)$ for this pair. If i and j are of opposite colors, then we assess a penalty of $-b \cdot d(i, j)$ (i.e., we give a bonus).

Thus, if you are given a proposed labeling of all zebras with either red or blue for each time step, you can compute how good an explanation of zebra activity it is. Your goal is to find the *best* possible labeling, in the sense that it has the smallest possible total penalty. But you'll only need to output the total penalty of the labeling, not the labeling itself.

Input

The first line is the number K of input data sets, followed by the K data sets, each of the following form:

The first line of each data set contains two integers z, t , the number of zebras $2 \leq z \leq 10$, and the number of time steps $2 \leq t \leq 50$. Next comes a line with three floating point numbers $a, b, c \geq 0$, the penalty multipliers. This is followed by t lines, describing zebra positions. Each line contains $2z$ floating point numbers, giving the positions of the zebras in the form $x_1 y_1 x_2 y_2 \dots x_z y_z$. The first line contains the positions at time 1, the second line at time 2, and so forth.

Output

For each data set, output "Data Set x :" on a line by itself, where x is its number. On the next line, output the minimum penalty that can be achieved by any grouping over time of the zebras, rounded to two decimals. Each data set should be followed by a blank line.

Sample Input/Output

Sample input zebras.in

Corresponding output

```
1
5 10
1.0 1.0 20.0
0.0 0.0 0.0 0.5 0.0 1.0 10.0 0.0 10.0 0.5
0.0 0.0 0.0 0.5 10.0 1.0 10.0 0.0 10.0 0.5
0.0 0.0 0.0 0.5 0.0 1.0 10.0 0.0 10.0 0.5
0.0 0.0 0.0 0.5 8.0 1.0 10.0 0.0 10.0 0.5
0.0 0.0 0.0 0.5 0.0 1.0 10.0 0.0 10.0 0.5
0.0 0.0 0.0 0.5 0.0 1.0 10.0 0.0 10.0 0.5
0.0 0.0 0.0 0.5 9.0 1.0 10.0 0.0 10.0 0.5
0.0 0.0 0.0 0.5 9.0 1.0 10.0 0.0 10.0 0.5
0.0 0.0 0.0 0.5 9.0 1.0 10.0 0.0 10.0 0.5
0.0 0.0 0.0 0.5 9.0 1.0 10.0 0.0 10.0 0.5
```

```
Data Set 1:
-476.30
```