

The “Final Exams” Programming Contest

USC Programming Contest, Spring 2007

04/28/2007

Sponsored by VSoE, Electronic Arts, and LanguageWeaver

The semester has just ended, and now is the beginning of study days. That means that in a few days, final exams will be starting. And you are all sitting here, solving programming contest problems instead of studying. Hmmm. To make it worth your while, we'll focus on computational problems that might help you do better on your exams. The operative word is “might”.

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: several problems ask you to print numbers rounded to two decimals. In C, you do this via `printf("%.2f", r);` where `r` is the variable you want to print. And in Java, the syntax is `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: Overlap!

File Name: overlap.cpp|overlap.java

Input File: overlap.in

Description

One problem that seems to crop up fairly frequently in finals week is that students are supposed to be taking multiple final exams at the same time. Now, for some students, this may actually be feasible: they turn in their exams for the first class an hour early, and use the hour to take the other exam. However, for the rest of us . . .

Now, this problem could be easily detected early, and automatically. The default times for each course are posted well in advance, and the courses a student is enrolled in are known early as well. So that we can collect statistics about overlaps in the future, you get to write a little tool that finds out just how many students have overlapping course finals.

Input

The first line contains a number $K \geq 1$, which is the number of input data sets in the file. This is followed by K data sets of the following form:

The first line of a data set contains two numbers m, n between 1 and 1000. m is the number of courses offered, and n the number of students.

This is followed by m lines, each giving the name of a course, and the day and time of the final. The name of each course is a string of 4 upper-case characters (such as "CSCI") followed by three digits (such as "402"). The day is denoted by "M", "T", "W", "TH", or "F", in uppercase. The duration of the final is given in the format "hh:mm-hh:mm", the starting and ending times. The hour is given between 00 and 23, the minute between 00 and 59. Each final exam will finish on the same day it starts. These strings will be separated by a single space each.

Next are n lines, each describing one student. Since we only care about statistics, not about names, each student is simply described by the list of courses he/she is taking. Each list is on one line, consisting of 1–10 of the course identifiers described above, separated by a single space each.

Output

For each data set, first output "Data Set x:" on a line by itself, where x is its number. Then, output the total number of students who have overlap between two or more of their final exams. If one exam ends at the exact time that another starts, that does not count as overlap.

Sample Input/Output

Sample input overlap.in

Corresponding output

```
1
5 3
CSCI402 T 11:00-13:00
CSCI201 F 14:00-16:00
DENT221 T 12:30-14:30
AHIS364 TH 17:00-18:42
ANTH472 F 12:30-14:00
CSCI201 CSCI402 AHIS364
CSCI402 DENT221 ANTH472 AHIS364
ANTH472 CSCI201
```

```
Data Set 1:
1
```

Problem B: Study Days

File Name: studies.cpp|studies.java

Input File: studies.in

Description

In order to prepare for your exams without distractions from lectures etc., the university gives you a few study days, including today. So much for the “no distractions” part . . .

Given that study time is rather limited, it is crucial to divide it up well for studying for various exams. In the end, you just want to maximize your GPA, so maybe it's worth to put up with a C in one course (where a better grade would require a lot of studying) to secure A's (which can be achieved with relatively little studying each) in all other courses. Clearly, this is a crucial optimization problem, and solving it well will probably benefit you tremendously.

Input

The first line contains a number $K \geq 1$, which is the number of input data sets in the file. This is followed by K data sets of the following form:

The first line of a data set contains two numbers n, H , the number of courses you are taking (an integer n between 1 and 10) and the number of hours available for studying (an integer $H \leq 100$).

This is followed by n lines, each containing 10 integers. These integers in line i will describe, in the order given, the number of hours you need to study in order to get an A (4.0), A- (3.7), B+ (3.3), B (3.0), B- (2.7), C+ (2.3), C (2.0), C- (1.7), D+ (1.3), D (1.0) in the i^{th} of your courses. The numbers will be non-increasing. The interpretation is that if you study at least that many hours in the particular subject, you will get the given grade. If you don't even study enough for a D, your grade will be F (0.0).

Output

For each data set, first output “Data Set x :” on a line by itself, where x is its number. Then, output the maximum possible GPA (rounded to two decimals) you can obtain by dividing your H hours of studying between the subjects.

Sample Input/Output

Sample input studies.in

```
1
3 60
40 37 35 33 30 26 20 10 5 1
10 10 10 10 10 10 10 10 10 1
24 23 22 21 20 20 20 20 20 20
```

Corresponding output

```
Data Set 1:
3.43
```

Problem C: Exam Seating

File Name: seating.cpp|seating.java

Input File: seating.in

Description

Sometimes, in exams, the right choice of exam seat can be at least as important as the right preparation. You all know what we are talking about. (Not that anyone at USC would ever actively be involved; but word travels from certain other schools).

Hypothetically speaking, if you wanted to copy from classmates, you would want a seat where you can see the exams of (1) many students, who are (2) doing well in the class. Clearly, there is a tradeoff. You can see a student's exam better if it is closer, and sometimes, your view is entirely blocked by other students. So finding the best seat at an exam is clearly a challenging problem.

We will assume that all seats are located at integer points on a $d \times d$ grid, numbered from $(1, 1)$ to (d, d) . At each location (x, y) is a student who is characterized by his/her skill $s_{x,y} \geq 0$ and shoulder width $w_{x,y} \in [0, \frac{1}{2}]$. (If a seat is unoccupied, we model this as $s_{x,y} = 0, w_{x,y} = 0$.) Sitting at seat (x, y) , you can only see "forward", which we interpret to mean that you can only possibly copy from seats (x', y') with $y' < y$. The shoulder width can be interpreted as follows: a student at (x, y) is modeled as a straight line from $(x - w_{x,y}, y)$ to $(x + w_{x,y}, y)$, and you can never see through any student/line. We assume that each student keeps the exam in the middle, at (x, y) . Formally, sitting at (x, y) , you can "possibly see an exam" at (x', y') if $y' < y$, and the straight line from (x, y) to (x', y') does not go through any student other than the one at (x', y') . (If the straight line goes through the student "right at the edge", it counts as going through the student.)

Now you also have limited eye sight, meaning that farther away students' exams will be hard to decipher. Specifically, if your eye sight is E , and a student sits at distance $D > E$ from you, you won't be able to see anything. At distance $D \leq E$, you will be able to see a fraction $1 - D/E$ of the other student's exam. Your total benefit is then the sum, over all students with visible exams, of their skills, weighted by the fraction you can see. As a final constraint, you can of course only sit in an empty seat.

Input

The first line contains a number $K \geq 1$, which is the number of input data sets in the file. This is followed by K data sets of the following form:

The first line contains two numbers, the integer size $d \leq 100$ of the classroom and your eye sight value $E > 0$, a floating point number.

This is followed by d^2 lines, where line $d(y - 1) + x$ describes the student sitting at position (x, y) . Each such line contains two numbers s, w , the student's skill and width. Both will be non-negative floating points, and the width will be at most $\frac{1}{2}$. We will guarantee that there will be at least one empty seat in each data set.

Output

For each data set, first output "Data Set x:" on a line by itself, where x is its number. Then output the maximum total benefit you can get at the best open seat in the room, rounded to two decimals.

Sample Input/Output

(see next page)

Sample input `seating.in`

Corresponding output

```
1
3 2.2
0 0
4 0.4
2.1 0.2
6.0 0.2
0.2 0.1
0.0 0.0
10.5 0.5
0.0 0.0
0.0 0.0
```

```
Data Set 1:
2.57
```

Problem D: Yes or No?

File Name: `yesorno.cpp|yesorno.java`

Input File: `yesorno.in`

Description

Multiple choice tests are rather easy to grade, and therefore very popular among some teachers. Some students like them (“you have a chance to get the answer right even if you have no clue”), some students hate them (“there must be a hidden catch somewhere”). The ultimate form of multiple choice questions are true/false ones.

Now assume that you are taking a true/false test. For each question, you have an a priori estimate of how likely each of the two answers is right. For question i , you think that the answer is “Yes” with probability y_i , and “No” with probability $1 - y_i$. So of course, you should just pick whichever is higher between y_i and $1 - y_i$ (if $y_i = \frac{1}{2}$, you could go either way).

The catch? If your estimate of “Yes” is higher than for “No” for *all* questions, you may suspect something is fishy. Most teachers don’t like clear patterns in their answer key, and try to mix things up. Suppose that you know that this particular teacher always has a number of “Yes” answers between ℓ and r (inclusive), for some $\ell \leq r$. Then, picking the set of questions to answer “Yes”, maximizing your total expected number of correct answers is not quite trivial any more. So you had better write a program to do it. Specifically, you want to find the answers to give to all questions so as to maximize the expected number of questions you get right, but giving the answer “Yes” at least ℓ and at most r times.

Input

The first line contains a number $K \geq 1$, which is the number of input data sets in the file. This is followed by K data sets of the following form:

The first line of each data set contains three integers $\ell \leq r \leq n$, where $n \leq 200$ is the total number of questions on the exam, ℓ is the minimum number of questions on which the answer will be “Yes”, and r the maximum number.

This is followed by n lines, each giving a fractional number $y_i \in [0, 1]$ for the corresponding question i .

Output

For each data set, first output “Data Set x :” on a line by itself, where x is its number. Then, output the maximum total number of questions you can get right subject to all the constraints, rounded to two decimals.

Sample Input/Output

Sample input `yesorno.in`

```
1
2 4 5
0.2
0.4
0.35
0.8
0.4
```

Corresponding output

```
Data Set 1:
3.25
```

Problem E: Essay Writing

File Name: essays.cpp|essays.java

Input File: essays.in

Description

It's one thing to get the computer to speed up your math homework by solving that linear system for you, but writing essays? Well, if your teacher isn't looking all that closely, one can actually do something with computers, based on Markov Models. You can learn which words follow which other words in the English language (by analyzing examples), and then generate random text using those rules. It will look surprisingly similar to English, in particular if you look at which words follow which sequences of two previous words.¹

Suppose you had already made your computer analyze the English language for you. Now, you have to write a final essay. Obviously, you won't want to generate completely random text, as you would rather have the text contain certain key terms related to the essay topic. Here, you are going to write a program that will test whether you can generate an essay of a given length with two given keywords.

More formally, you will be given sample text, two keywords, and a length target w . Your goal is to generate exactly w words of text, containing both of the keywords at least once each (in any order). In the sequence you generate, each word can only follow a word it follows at least once in the sample text, and the first word of your sequence can be arbitrary. You are to decide if it is possible to generate any text matching those specifications.

Input

The first line contains a number $K \geq 1$, which is the number of input data sets in the file. This is followed by K data sets of the following form:

The first line of a data set contains two numbers n, w , and two words s_1, s_2 . Here, $w \leq 100$ is the number of words your essay is supposed to contain, s_1 and s_2 are the two required words, and $1 \leq n \leq 10$ is the number of lines of example English text your program is supposed to learn from.

This is followed by n lines, each containing a sequence of 1–20 words. Each word (here and above) is a sequence of 1–20 lower-case characters. Words will be separated by one or more white spaces. There will not be any punctuation or other characters.

Output

For each data set, first output "Data Set x:" on a line by itself, where x is its number. Then, output whether or not a corresponding essay can be generated.

Sample Input/Output

Sample input essays.in

Corresponding output

```
2
4 5 solution answers
when you program your solution correctly the
program will work and
it will write right answers for you but when you are wrong
it will not
4 6 solution answers
when you program your solution correctly the
program will work and
it will write right answers for you but when you are wrong
it will not
```

```
Data Set 1:
No
Data Set 2:
Yes
```

¹You can also apply that to music in principle. The results for both language and music can be quite hilarious.

Problem F: Party!

File Name: party.cpp|party.java

Input File: party.in

Description

Having passed all of your exams (despite spending an entire Saturday afternoon at a programming contest, no less), you decide to throw a little party, and invite your 500 or so best friends. Naturally, the beverages are not restricted to soy milk and soda water. Once the party draws to a close, being the responsible host, you want to ensure that everyone will make it home safely. In particular, this involves keeping anyone from driving intoxicated. You might end up with a few people sleeping in your living room as a result.

We assume that some of your friends are coming from the same location. If they live in the same location, they can drive together. However, each car can only fit so many people. You will be given, for each person, where they live, and whether they are intoxicated. In addition, you will be given for each car where it “lives”, and how many people it seats. A car can be driven by anyone living at its location, so long as he/she is sober. You are to determine how many people will not be able to get home.

Input

The first line contains a number $K \geq 1$, which is the number of input data sets in the file. This is followed by K data sets of the following form:

The first line of a data set contains three numbers n, c, ℓ . $n \leq 500$ is the number of friends, $c \leq n$ the number of cars, and $\ell \leq c$ the number of locations where people live.

Next are n lines, each describing a friend. For each friend, you will be given a number between 1 and ℓ , the location where he/she lives, followed by the letter ‘I’ (for intoxicated) or ‘S’ (for sober).

This is followed by c lines, each describing a car, by giving its location (an integer between 1 and ℓ) and its capacity (an integer between 2 and 8).

Output

For each data set, first output “Data Set x:” on a line by itself, where x is its number. Then, output the number of friends who will need to stay at your apartment (you try to have as few as possible stay).

Sample Input/Output

Sample input party.in

```
1
8 3 2
1 I
1 I
1 S
2 I
1 I
1 I
2 S
2 S
1 3
1 3
2 4
```

Corresponding output

```
Data Set 1:
2
```