The Graduation Contest

USC Programming Contest, Spring 2019

March 30, 2019

Sponsored by VSoE, Electronic Arts, Google, Marconi Foundation, Northrop Grumman

It's almost April, and you know that that means: soon, the jacaranda trees will be blooming, and campus will turn into complete craziness when the next generation of Trojans will graduate. In fact, many of you — seniors and MS students — will probably soon be among the crowd, walking around in caps and gowns, taking pictures, starting to feel nostalgic about your time at USC, or being happy that you finally get out. There are so many things to do when you graduate that you obviously can't be bothered with silly logistics. So we had better prepare and get some programs in place that will take care of the logistics for you.

You can solve or submit the problems in any order you want. When you submit a problem, you submit your source code file using PC^2. 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` (the screen). 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, people, etc.) always starts at 1, not at 0.

And a piece of advice: you will need to print floating point numbers rounded to two decimals. Here is how you do that:

C: `printf ("%.2f", r);`.

C++: `cout.precision (2); cout << fixed << r;`.

Java: `System.out.print ((new java.text.DecimalFormat("#.##")).format(r));`

[Including this page, the problem set should contain 9 pages. If yours doesn't, please contact one of the helpers immediately.]

Problem A: Cap Size

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

Input File: `cap.in`

Description

Picking the right size of your cap is surprisingly difficult. If your cap is too small, it keeps digging into your head, or — even worse — you can't even get it over your computer science filled head. And if it is too large, it keeps wiggling, and you know that it will just fall down or be blown away by the wind at the most solemn and important moment. So before committing to a cap size, you tried on several different caps, and recorded how they fit: some might have been too large, some too small, and if you were lucky, one might even have fit perfectly. After trying on some (but maybe not all) caps, you now want to compute how many caps could possibly still fit you perfectly.

Input

The first line contains a number $1 \leq K \leq 100$, 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 integers n, t with $0 \leq t \leq n \leq 100$: n is the number of caps that are available, and t is the number you tried on. This is followed by a line with n integers $0 \leq s_i \leq 1000$, giving you the size of the i^{th} cap. They will be given in strictly increasing order (which also means that the same cap size never appears twice).

Next are t lines, each with two integers c_i, f_i, where $0 \leq c_i \leq 1000$ and $-1 \leq f_i \leq 1$. Here, c_i is the size of the cap you tried on (which will always be a size occuring in the earlier list of all sizes), and f_i the fit, where $f_i = -1$ means it was too large, $f_i = 0$ that it fit perfectly, and $f_i = 1$ that it was too small.

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 caps that might still fit you perfectly based on the information you received. If the input claimed that a large cap was too small and a smaller cap was too large, or that two different cap sizes fit perfectly, then output "Inconsistent feedback" instead.

Sample Input/Output

Sample input `cap.in`

```
5
4 2
1 4 7 10
1 1
10 -1
3 1
1 4 5
4 0
3 2
1 2 3
1 -1
2 1
3 2
1 5 8
1 1
5 -1
3 2
1 4 9
1 0
9 0
```

Corresponding output

```
Data Set 1:
2

Data Set 2:
1

Data Set 3:
Inconsistent feedback

Data Set 4:
0

Data Set 5:
Inconsistent feedback
```

Problem B: First-Name Basis

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

Input File: `first.in`

Description

In order to schedule who walks across the podium when, one typically uses alphabetical order by last name, breaking ties by first name. In order to do so, you first have to figure out which is a student's first and last name. This gets more difficult because (1) in a number of countries/cultures, the order of family and given name is reversed, and (2) even when that's not the case, students who can effortlessly write 10,000-LoC operating systems seem to have serious difficulty determining which of their first and last name to put into which of the two fields labeled "First Name" and "Last Name". In this problem, you will write a program that helps USC's administrators fix the ordering of first and last names.

Specifically, for each of the students, you will be given what he/she *entered* as their first and last name. Furthermore, you get to assume[1] that each string can serve either as a first name or a last name, but not both. That is, you cannot have one person named "Aretha Franklin" and another named "Franklin Roosevelt", because this would use "Franklin" as both a first and last name. If two students claimed to have these names, then one of them would have to have reversed his/her names. Given the list of names, you should output the smallest number of students whose claimed names need to be reversed to make all names consistent (i.e., each string is only used as a first name, or only used as a second name), or otherwise output "Impossible".

Input

The first line is the number $1 \leq K \leq 100$ of input data sets, followed by K data sets, each of the following form:

The first line is the number $0 \leq n \leq 1,000$ of students whose names were included. This is followed by n lines, each containing two strings $s_{i,1}, s_{i,2}$, separated by some kind of spaces. Each string consists of 1–30 lowercase letters.

Output

For each data set, output "`Data Set x:`" on a line by itself, where `x` is its number. Then, output the smallest number of students whose reported first and last names need to be reversed to make the input consistent, or output "Impossible" if there is no way to accomplish this.

Each data set should be followed by a blank line.

Sample Input/Output

Sample input `first.in`

```
2
5
a b
a c
c d
b d
b e
3
alice bob
bob carol
alice carol
```

Corresponding output

```
Data Set 1:
2

Data Set 2:
Impossible
```

[1]not completely realistically

Problem C: Mangling Names

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

Input File: `names.in`

Description

Having figured out everyone's first and last name, it's time for everyone to walk across that stage, to cheers of friends and family, receive a degree, and hear one's name pronounced/mangled. One has to have some pity, though, with the poor officials who have to pronounce hundreds or thousands of names, from dozens of countries, some of them very long. Getting the name right can take a while, and delay the whole ceremony. In this question, you will figure out just how much.

You will be given a table that tells you, as a function of the number of vowels ('a', 'e', 'i', 'o', 'u', 'y'; we will treat 'y' as only a vowel, never a consonant) and consonants in the name, how long it will take to pronounce a name correctly. Then, for each graduating student, you will be given the student's name. You are to output the total length of the ceremony.

Input

The first line contains a number $1 \leq K \leq 100$, 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 two integers $0 \leq L \leq 30, 0 \leq n \leq 100,000$. Here L is the (larger of the) maximum number of vowels/consonants in any name, and n is the number of students marching.[2] This is followed by $L+1$ lines, each with $L+1$ integers. Entry j in row i contains an integer $0 \leq p_{i,j} \leq 100$, which tells you how many seconds it takes to pronounce a name with i vowels and j consonants; here, $0 \leq i, j \leq L$.

This is followed by n lines, each containing a string of length at most $2L$, consisting only of lowercase letters. This string is the student's name, and it will contain at most L vowels and at most L consonants.

Output

For each data set, first output "`Data Set x:`" on a line by itself, where `x` is its number.

Then, output the total duration of the ceremony in seconds.

Each data set should be followed by a blank line.

Sample Input/Output

Sample input `names.in`

```
2
4 4
1 2 8 25 50
1 1 1 3  8
7 6 4  8  10
6  5  5  4 4
15  12 10 10 10
cthulu

amoeba
czsz
1 2
0 5
1 10
a
b
```

Corresponding output

```
Data Set 1:
71

Data Set 2:
6

```

[2] If you are wondering about allowing n up to 100,000: have you seen our graduation ceremonies lately?!?

Problem D: GPA

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

Input File: `gpa.in`

Description

For many students, at graduation, all that matters is one number: the GPA.[3] Except, then, many employers seem to be not so interested in how a CSCI or EE student did in "Introduction to Origami;" somehow, they seem to care more about performance in CSCI104 or CSCI270 or CSCI360. Strange! As a result, the "Major GPA" has become a thing, which only includes classes "in the major." But if you are double- or triple-majoring, then computing all your multiple major GPAs becomes rather time-consuming. Isn't that what computers are for? In fact, it is — thanks for pointing this out!

You will be given the complete course catalog of a university (course names and units), as well as the major requirements for each of the majors. Then, you will be given a list of students, and for each student all the classes they have taken and the grades they received. You will output, for each student, the overall GPA, and the GPA for each major such that the student completed all requirements for this major. The GPA is weighted: for each class the student took, you multiply the student's grade (a number between 0.0 and 4.0) by the number of units of the class. In the end, you divide the sum of all these weighted grades by the total number of units the student took. That's the student's GPA. Similarly for the major GPA, except you only sum the classes relevant for the major.

Input

The first line contains a number $1 \leq K \leq 100$, 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 the data set contains three integers $1 \leq C \leq 10000, 1 \leq M \leq 500, 1 \leq s \leq 1000$ (the number of classes, the number of majors, and the number of students who are graduating).

This is followed by C lines, each describing a class i, by giving its name n_i and number of units u_i. The name consists of 2–4 uppercase letters, followed immediately by exactly 3 digits. The number of units is an integer between 1 and 6, inclusive.

Next come the M majors. The first line of a major j contains first a string of 2–4 uppercase letters (the name of the major), and then the number $r_j \geq 1$ of classes the major requires (an integer). This is followed by r_j lines, each containing exactly one class name. The class name will be a legal name from the first C lines. The total number of units of any major will add up to at most 128.

Finally, we have the s students. For each student k, you first get the integer $t_k \geq 1$ of classes the student has taken. This is followed by t_k lines, each containing first the name of a class (a legal string from the first C lines), followed by the student's grade in the class, a floating point number between 0.0 and 4.0. No student ever takes more than 200 units total. And no student will ever take the same class twice.

Output

For each data set, first output "`Data Set x:`" on a line by itself, where `x` is its number. Then, output the overall GPA and all applicable major GPAs, for each student, in the order the majors were listed in the input file. Specifically, first print a line "Student k", where k is the student's number. Then print a line with the student's GPA. Then, for each major for which the student took all required classes, print the name of the major, followed by the student's GPA in that major. (See below for the format.) All GPAs should be rounded to two decimals. Each data set should be followed by a blank line.

Sample Input/Output

Example I/O on the next page.

[3]Well, ok, there may also be the student debt.

Sample input `gpa.in`

```
1
5 3 1
CSCI104 4
CSCI109 2
EE364 3
ISE330 4
ORIG101 1
CSCI 3
CSCI104
CSCI109
EE364
CENG 2
CSCI104
EE364
ISE 3
ISE330
EE364
CSCI104
4
CSCI104 3.7
CSCI109 4.0
EE364 3.3
ORIG101 4.0
```

Corresponding output

```
Data Set 1:
Student 1
GPA: 3.67
CSCI: 3.63
CENG: 3.53
```

Problem E: Department Receptions

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

Input File: `food.in`

Description

After you've walked and received your degree (and moved your tassel), it's finally time to go and raid — pardon, visit — the department receptions. Unfortunately, you only have limited time to do so, because you have an appointment to meet your parents and friends for a photo shoot. So your goal is to eat as much good food as possible in the limited time. The emphasis is on "good;" it is well known that food quality varies between different departments,[4] so you want to choose carefully which receptions you attend.

There are a few snags to watch out for: (1) some receptions do access control, and won't let you in unless you have some kind of access (like being a student in their department, or being friends with one of the staff members); (2) there are places with larger crowds, which slow you down; and (3) if you don't eat enough, you might just drop from being famished before you reach the truly delicious food. Write a program to compute the maximum total food points you can consume.

Campus will be represented as a rectangular map. In any one step, you can move one square up, down, left, or right (but not diagonally). Each step you take burns one unit of energy, and each unit of food you consume restores one unit of energy. Consuming a unit of food requires you to stay in a food spot for one unit of time. In addition to your start and finish square (which will be marked by 'S' and 'T' on the map), there are the following types of squares:

- Move squares, denoted by '.', ':', ';', and '#'. There's no food or access control. Stepping onto such a square takes time: one unit for '.', two units for ':', three units for ';', and four units for '#'. Think of '.' as an empty walkway and '#' as the huge crowds around Tommy Trojan or the chocolate fountain.
- Access controlled squares, denoted by capital letters A–H. These are locations where some department has set up a checkpoint to keep greedy students away from their food. You can only enter such a square if you have that privilege — you will be given the list of all your privileges as part of the input. Entering an access controlled square takes one unit of time. By the way, so does entering the start or finish square.
- Food squares, denoted by digits 1–5. Entering a food square takes one unit of time. Then, with each time step you wait (i.e., don't move), you gain one unit of energy and the given number of food points.

Your goal is to start at 'S', arrive at 'T' by a given time, never have your energy drop to 0 or below (including in the final step when you arrive at 'T'), and maximize the total number of food points you got along the way.

Input

The first line is the number $1 \leq K \leq 100$ of input data sets, followed by the K data sets, each of the following form:

The first line of the data set contain four integers h, w, e, t with $1 \leq h, w \leq 30$, $1 \leq e, t \leq 100$ and one string of up to 8 distinct upper-case letters from A–H. h and w are the height and width of the map, e is your initial energy level, and t is the number of time steps until you meet your parents and friends. The string (which might be empty) describes your access privileges.

This is followed by h lines of exactly w characters each. Each character is exactly one of those described above (letters 'A'–'H', 'S', 'T', digits 1–5, '.', ':', ';', '#'). There will be exactly one 'S' and exactly one 'T' in the map.

Output

For each data set, output "`Data Set x:`" on a line by itself, where `x` is its number. Then output the maximum total number of food points you can gain and arrive at the meeting point at time t or earlier. If it is impossible for you to arrive at the meeting point at time t or earlier, output "Impossible" instead.

Each data set should be followed by a blank line.

Sample Input/Output

Example I/O on the next page.

[4] How come EE always has much better food than CS?

Sample input `food.in`

```
1
8 15 4 37 EAD
1D.ST.1.....##5
;A..........##.
;A...........#.
;A.....BBBBB;;;
;E.....B4...;.;
3E....2B.......
.......B.......
3.....;;.......
```

Corresponding output

```
Data Set 1:
40
```

Explanation: You start out really hungry (energy: 4), so all you can manage with your initial energy level is to make it to the bad food in the top left corner. You now need to eat enough to make it somewhere else. Unfortunately, because of the access control letters 'B' and the really slow squares in the top right, you cannot make it to the locations of quality 4 and 5 in time to make your meeting at time 37. So you eat the minimum amount of food to make it to the level 3 food on the left, almost at the bottom. Instead of going straight down (which would be slow), you go one square to the right, using your access privileges for 'D', 'A', and 'E' along the way. You eat level-3 food for as long as you can (and enough to safely make it to the meeting spot), then leave to make it to the meeting location in time. Overall, you will spend 7 units of time eating bad food (because that's how much energy you need to make it to level-3 food), and 11 units of time eating level-3 food, for a total of $7 + 3 \cdot 11 = 40$ food points.

Instead of initially heading to the top-left corner, you could have gone to the level-1 food in the top middle. But even if you eat there, because of the 'B' access control and the slow squares in the top right, you won't be able to eat enough level-4 or level-5 food to do better, and heading for level-2 food (which you could do) will give you fewer food points as well.

Problem F: Jump!

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

Input File: `jump.in`

Description

Now that you are well fed, it is time to take a graduation picture with your good friends. For some reason, about half of all graduation pictures seem to involve many students simultaneously jumping in the air, while holding up their caps. Being engineers, you get obsessive about this: you want the picture to be such that the held-up caps are as close as possible to being at exactly the same height. In order to achieve this, you will program the camera to time its shot perfectly. So when exactly does the camera need to take the shot? Did we say "program" the camera? We meant it!

For each of the people in the picture, you will be given two numbers: (1) the distance d_i (in meters) from the feet to the raised cap (which stays the same throughout the jump, i.e., your friends and you all extend your hands by a constant amount above your feet/heads), and (2) the speed v_i (in meters/second) with which the person initially jumps. Recall from middle school physics that due to the Earth's gravity, someone who jumps up with speed v will be at height $vt - \frac{1}{2}gt^2$ at time t, where $g = 9.81 m/s^2$ is Earth's gravity constant.[5] To be more precise, it is this height until you hit the ground again, which we assume happens exactly when the height is 0 — afterwards, you stay at height 0. Your goal is to find a time t at which the height difference between the highest hand/cap and the lowest hand/cap at that time is minimized.

Input

The first line contains a number $1 \leq K \leq 100$, 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 a single integer $1 \leq n \leq 100$, the number of friends you are taking a picture with. This is followed by n lines, each containing two doubles $0.0 \leq d_i, v_i \leq 10.0$.

Output

For each data set, first output "`Data Set x:`" on a line by itself, where `x` is its number. Then, output the time at which the distance between the highest and lowest caps is minimized, rounded to two decimals. If there are multiple such times, output the earliest one.

Each data set should be followed by a blank line.

Sample Input/Output

Sample input `jump.in`

```
1
3
6.0 0.0
4.5 7.0
5.5 7.0
```

Corresponding output

```
Data Set 1:
0.08

```

[5] It is not exactly that, but you should use this approximation.