The Programming Contest Programming Contest

Cornell ACSU Programming Contest, Fall 2000

October 14, 2000

While last year's contest went a lot more smoothly thanks to PC^2 and other aspects, there's still a lot more room for improvement. Given that there now are dozens of wonderful programmers at hand, we thought we'd let you write some programs to make the contest even better in the future. To avoid chaos in subsequent years, we will still have to test your programs, though, and make sure they are correct.

While all problems will revolve around the programming contest itself, there will **not** be a common input format this time. And while the questions will appear roughly in the order in which they are relevant to the contest, there is no need to solve them in this order — choose whatever order you like best.

As to the output format, read the instructions **carefully**, and look at the sample output provided. We expect output to be exactly in the format described, other formats will be considered incorrect (read the paragraph on "error messages" on the rules handout to find out more details).

Along the same lines, adhere to the fixed file names prescribed by this problem set. This applies both to the names of the input and output files (that you are supposed to read from and write to), and the name of the file you submit. That is, if your file solves problem A, it has to be named `proba.cpp` or `proba.java`, depending on the programming language you use. If you want, you can use different programming languages for different problems. Do check programming language constraints on the rules handout.

[Including this page, this problem set should contain 6 single-sided pages. If yours doesn't, please contact one of the helpers immediately.]

Problem A: Got problems?

Input File: PROBA.IN

Output File: PROBA.OUT

Description

In preparation for the contest, David and Hubie came up with a number of algorithms and programming topics that they want to check, such as graph algorithms, geometry, searching, sorting, string processing, etc.

Also, they made up a number of problems (one of which you're looking at), and for each problem determined which categories it falls into. A problem will always belong to at least one category, but it can belong to several categories, potentially even all of them.

Now they want to find a set of problems that will cover all categories. And as good problems are hard to make up and therefore precious, they want to use as few problems as possible. You are to write a program that will save them from doing this by hand.

Input

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

The first line of a data set contains two numbers, M and N $(1 \leq M, N \leq 20)$. M is the number of problem categories, and N is the number of available problems. Problem categories are denoted by $1, \ldots, M$, and problems by $A, B, \ldots$. The following N lines each contain the numbers of the categories covered by a problem — the first line for problem A, the second one for B, and so on.

Output

The output file contains one line for each input, separated by an empty line. Each line contains first the number of the data set, and then the solution. The solution is the list of problems to be used, separated by whitespace, and sorted alphabetically.

You may assume that there is always a solution, but it need not be unique. If there are several best solutions (i.e. of equal length), output the one which comes alphabetically first in the output format just described.

Sample Input/Output

Sample input PROBA.IN

```
2
2 3
1
2
1 2
4 5
1 2 3
1 2
2 4
1 3 4
2 3 4
```

Corresponding output PROBA.OUT

```
Data Set 1: C

Data Set 2: A C
```

Problem B: Hilbert's Contest?

Input File: PROBB.IN

Output File: PROBB.OUT

Description

The participants start arriving, and they must be assigned seats at the computers. To keep things in order, the seats are supposed to be assigned in alphabetical order, i.e. the alphabetically first person gets machine 1, the next one machine 2, etc. To be more precise, by "alphabetical", we mean the telephone directory order, e.g. "Cornel, Uncle Ezra" would come before "Cornell, Ezra" (note the single "l").

Unluckily, participants don't tend to arrive in alphabetical order, and we don't want to wait until everybody is there before assigning machines (who knows when the last person will show up?). So, everybody gets a seat right away, at the risk of having to move again later on.

More precisely, at any time, when p people have already arrived, they are sitting at machines $1, \ldots, p$, in alphabetical order. Now, when the $(p+1)$st person arrives, she takes her machine, say m, and the people previously sitting at machines $m, \ldots, p$ have to move to machines $m+1, \ldots, p+1$.

This is quite a bit of commotion, and inconvenient for the participants, so we want to find out who has to move most, and how often that person has to move. Write a program that does it for us!

Input

The first line will contain the number K of input data sets. This is followed by the K data sets, each of the following format:

The first line contains the number P of participants ($1 \leq P \leq 500$). This is followed by P lines, each containing the name of one participant. The names appear in the order of arrival. Names will be at most 80 characters long, and consist only of uppercase and lowercase letters, comma (","), and blank (" ").

Output

For each data set, first output its number, then in a new line the maximum number of moves that anyone has to make (sitting down initially does not count as a move), and then the names of all people who have to move this often, in alphabetical order. Output for different data sets is to be separated by an empty line.

Sample Input/Output

Don't worry about the poor grammar in the output — but **do** copy the exact formulation:

Sample input PROBB.IN

```
3
3
Gates, Bill
Kempe, David
Chen, Hubie
4
Skywalker, Anakin
Prefect, Ford
Kirk, James D.
Dent, Arthur
2
Gromit
Wallace
```

Corresponding output PROBB.OUT

```
Data Set 1:
The following have to move 1 times:
Gates, Bill
Kempe, David

Data Set 2:
The following have to move 3 times:
Skywalker, Anakin

Data Set 3:
The following have to move 0 times:
Gromit
Wallace
```

Problem C: Format Error!

Input File: PROBC.IN

Output File: PROBC.OUT

Description

Those of you who participated in the last contest probably still remember them with pain, and the new ones will soon find out what we're talking about: Format Errors. You write a wonderful correct solution and submit it, and Bang!, you get the result "Format Error!" from the judges. Just because you had one blank too much, or accidentally typed the wrong kind of parentheses.

Of course, we are aware that this is painful, but we don't want to judge all these solutions with lots of output by hand — you'd never get the results. However, if we had a program to do the job, things might look a lot better. Guess what! You get to write us such a program.

The program will compare two strings at a time, and decide whether they are "equal" (in a relaxed sense) to one another. Strings will consist of uppercase and lowercase letters, digits, whitespace (no tabs), and the following punctuation marks (no others will appear in our test cases):

```
( ) [ ] { } . , ; :
```

The following are the rules according to which strings will be considered equal:

- Upper and lowercase letters can be interchanged.
- The size of a block of whitespace is unimportant (this of course applies only to blocks consisting of at least one whitespace). Whitespace immediately following or preceding punctuation marks is unimportant. Leading or trailing whitespace (i.e. at the beginning or end of strings) is unimportant.
- The various kinds of opening brackets can be interchanged. The various kinds of closing brackets can be interchanged (however, an opening bracket can not be interchanged with a closing bracket). Brackets **need not** match.
- Comma (",") and semicolon (";") can be interchanged.

Input

The first line is the number K of data sets in the file. This is followed by the K data sets, each consisting of two strings s_1 and s_2 (of length at most 1000 characters each) on separate lines. The newline character is not considered part of the string.

Output

For each data set, output its number, and either "equal" or "not equal", depending on whether s_1 and s_2 represent the same string for this data set. Output for different data sets should be separated by an empty line.

Sample Input/Output

Sample input PROBC.IN

```
3
(1, 4) (2, 3) (2, 4)
{1;4}  {2;3}  {2;4}
Data Set 1: equal
 data set 1 :   EQUAL
Data Set 1: equal
DataSet 1: equal
```

Corresponding output PROBC.OUT

```
Data Set 1: equal

Data Set 2: equal

Data Set 3: not equal
```

Problem D: Who gets the prizes?

Input File: PROBD.IN

Output File: PROBD.OUT

Description

The contest is over, everybody is disappointed at having screwed something up, and now, the judges have to look at hundreds of solutions, correct and incorrect, and determine who has won. How they wish they had a program to do that!

Judging will go exactly by the rules of the contest — if you are not sure about the exact rules, you should look them up on the "rules" handout that you received. You can assume that in all our test cases, no two participants will simultaneously have the exact same number of problems solved and exact same total time.

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 three number, M, N, and P. M is the number of problems ($1 \leq M \leq 10$), N is the total number of submissions ($1 \leq N \leq 5000$), and P is the number of participants ($1 \leq P \leq 500$). The problems are numbered $A, B, \ldots$, participants are numbered $1, \ldots, P$.

This line is followed by N lines, each describing one submission. Each line is of the form "$p\ m\ t\ j$". Here, p is the number of the submitting participant, m the identification letter of the problem (i.e. A, B, and so on), t the time (in minutes into the contest) at which the submission took place, and j the judgment, a 1 for a correct submission, and a 0 for an incorrect one. The lines will be sorted by non-decreasing submission times.

Output

The output for a data set is the number of the data set, and the list of all participants sorted according to their ranking. For each participant, it should list the number of problems solved and the total time (including penalties). The output for different data sets should be separated by an empty line.

Sample Input/Output

Sample input PROBD.IN

```
2
4 6 3
2 C 11 0
1 B 24 1
2 C 25 1
1 C 40 0
3 D 55 1
2 B 55 1
2 8 2
1 A 8 0
2 B 11 1
1 A 11 0
1 A 14 0
1 A 20 0
2 B 22 1
2 A 23 0
1 A 24 1
```

Corresponding output PROBD.OUT

```
Data set 1:
2 2 100
1 1 24
3 1 55

Data set 2:
2 1 11
1 1 104
```

Problem E: Pizza!

Input File: PROBE.IN

Output File: PROBE.OUT

Description

Finally, we get to the most important part of the contest: Eating the pizza. But even here, problems lurk for you! The pizza has delicious soy nuggets on it, but it isn't cut yet. So we have to cut it into slices. There are two important constraints we have to meet: every slice must have the same size, and every slice must have the same number of soy nuggets on it. To make things worse, the pizza is circular, and we can only cut along a straight line from the center of the circle to its circumference. And we can never cut through a soy nugget, it must always be on one side of the cut or the other.

You are to write a program that, given the positions of the soy nuggets, determines which is the largest number of slices into which the pizza can be cut such that every piece has the same size and the same number of soy nuggets (notice that a pizza can always be cut into 1 slice — so there always is a solution.)

Input

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

The first line contains the number N of soy nuggets on the pizza ($1 \leq N \leq 200$). This is followed by N lines, describing the positions of the soy nuggets. Each position is given in **polar coordinates**, measured from the center of the pizza. That is, a position is a pair "α r", where $0 \leq \alpha < 2\pi$ is the counterclockwise angle between the positive x-axis and the line from the pizza center to the soy nugget. $0 < r \leq 1$ is the distance of the nugget from the center (each pizza will have radius 1). We assume that nuggets are points, and no two nuggets will be lying on the same spot of the pizza.

Output

For each data set, output its number, followed by the maximum number of slices into which it can be cut according to the above rules. Remember that no cut can go through a soy nugget. Output for different data sets should be separated by an empty line.

Sample Input/Output

Don't worry about the grammar in the output.

Sample input PROBE.IN

```
2
2
1.57 0.5
1.57 0.7
4
0.7 0.9
1.5 0.1
1.8 0.5
3.05 1.0
```

Corresponding output PROBE.OUT

```
Data Set 1: 1 slices

Data Set 2: 2 slices
```