XXVI Maratón Nacional de Programación ACIS REDIS 2012

Problemas

(Este conjunto contiene 10 problemas; páginas numeradas de 1 a 19)

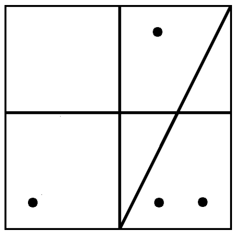
Problem A
Cookie

Source file name: `cookie.c, cookie.cpp` *or* `cookie.java`

The *Association of Cookie Makers* (ACM) has developed its newest and most decadent cookie in history, the "Square Chocolate Chip Cookie". This new cookie is a biscuit delight with lots and lots of chocolate chips for all the cookie lovers out there. However, in the development of this revolutionary type of cookie, the chocolate chips sprinklers have had some problems; though all the chips fall on the cookie, they are not evenly spread.

With a tradition of more than 26 years of cookie making, ACM sprinklers cannot be replaced. That is why the sales department has decided to cut the cookie in several pieces to maximize revenue. Each cut is made by a laser that moves across the cookie following a straight line path. Any chip chocolate on the laser path is destroyed during the cut process. Each piece of a square cookie is then valued according to its *chocolate chip density*, defined as the number of chips that it has, divided by its area.

The following figure shows an original cookie with four chips and three cuts. It is apparent that the piece at the lower right corner has a maximal chocolate chip density.



As part of the product testing team at ACM, you have to write a program that detects the *maximal chocolate chip density* among the resulting pieces, given an original square chocolate cookie, the positions on it that received a chocolate chip, and the cuts that should be made.

Input

The input consists of several test cases. The first line in a test case contains three integers L, C, and K, separated by blanks, where L is the length of the cookie side, C is the number of chips, and K is the number of cuts ($2 \leq L \leq 500$, $0 \leq C \leq 5000$, $0 \leq K \leq 50$). Each one of the following C lines contains two blank-separated integers $\hat{x}$ and $\hat{y}$, representing the (x, y) coordinates of a chocolate chip ($0 < \hat{x} < L$, $0 < \hat{y} < L$) in a Cartesian coordinate system where the cookie is represented by the square with vertices $(0, 0)$, $(L, 0)$, (L, L), and $(0, L)$. Each one of the following K lines contains four blank-separated integers x_1, y_1, x_2, and y_2, representing two distinct points on a straight line that defines a cut on the same coordinate system ($0 \leq x_1, y_1, x_2, y_2 \leq L$). You may suppose that there are not two chocolate chips placed at the same location. The last test case is followed by a line containing three zeros.

The input must be read from the file cookie.in.

Output

For each case, output a single line with a number indicating the maximal chocolate chip density among all the resulting pieces after the cuts. You may suppose that no answer will be greater than 10^2. The answer should be formatted and approximated to three decimal places. The floating point delimiter must be '`.`' (i.e., the dot). The rounding applies towards the *nearest neighbor* unless both neighbors are equidistant, in which case the result is rounded up (e.g., 78.3712 is rounded to 78.371; 78.5766 is rounded to 78.577; 78.3745 is rounded to 78.375, etc.).

The output must be written to standard output.

Sample input

```
6 4 3
1 1
4 1
5 1
4 5
3 0 3 6
0 3 6 3
3 0 6 6
0 0 0
```

Output for the sample input

```
0.296
```

Problem B
Movie Police

Source file name: `movie.c`, `movie.cpp` *or* `movie.java`

Movie Police (MP) is an international top secret law enforcement agency, controlling illegal movie downloads on internet. With their elite team of programmers, MP has developed a very smart algorithm to produce movie signatures. A *movie signature* is a binary string, one bit for every frame in the movie, so that the i-th bit in the signature corresponds to the i-th frame in the movie. The algorithm is so amazing that it outputs consistently the same signature for versions of the same film with different quality resolutions. One application of this revolutionary technology uses it to detect if a small clip is part of a movie, looking for a high similarity between the clip signature and the movie signature.

Now MP has started to apply this technology and, as a first step, a massive online database of movie signatures was already built. As a new member of the MP crew, you must write a program that, given the signature of a clip, finds the index in the MP database of a movie whose signature *matches* the clip signature at most. That is, a movie whose signature has a substring, of the same length of the clip signature, that is most *similar* to the clip signature.

Similarity between strings of the same length is defined by means of their *Hamming distance* (number of bits that do not match), so that "more similar" means "less Hamming distance".

Input

The first line of the input contains two positive integer numbers M and Q, separated by a blank, where M indicates the number of movie signatures in the database and Q indicates the number of clip signatures to process ($1 \leq M \leq 1000$, $1 \leq Q \leq 500$). Each one of the following M lines contains a binary string s_i describing the i-th movie signature in the database. You may suppose that s_i has length l_i, where $1 \leq l_i \leq 100$. Finally, there are Q lines, each one with a binary string that corresponds to a clip signature to search for maximal similarity in the database. You may assume that, for every clip signature to be searched, there is at least one movie signature in the database whose length is greater or equal than the clip's length.

The input must be read from the file movie.in.

Output

For each clip signature given in the input, output a single line with the lowest index i of a movie s_i ($1 \leq i \leq M$) that matches the clip at most, as above explained. If there are two movie signatures that match the clip signature maximally, answer the one with lower index in the database.

The output must be written to standard output.

Sample input	**Output for the sample input**

`3 1`
`000011`
`1101111111`
`1111100000`
`1000111`
	`2`

Problem C
Cinema-cola

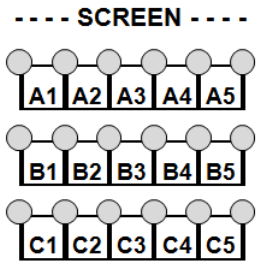
Source file name: `cinema.c, cinema.cpp` *or* `cinema.java`

A group of Z friends is going to the movies. They have already reserved some specific locations in the theater. As a kind of ritual, every friend likes to drink something while seeing the film.

Locations at the theater are disposed in a rectangular array of R rows and C columns. Rows are named sequentially with English capital letters (`'A'`, `'B'`, `'C'`, ...). Columns are named with integers (1, 2, 3, ..., C). Then, locations are identified by a string formed by the corresponding row letter and column number of the location.

Locations at the theater are chairs with plastic supports at both arms, to put a drink on each one. Supports are shared by neighbor locations but, of course, it is expected that at most one person uses a support to put his/her drink, but there is no etiquette rule that forces someone to use the right or the left support of his/her chair.

As a matter of example, the next figure illustrates a theater with 3 rows and 5 columns. Locations are depicted with rectangular forms, and supports with circles at both sides of them.



The group of friends goes to occupy the reserved locations. Before them, some persons came and occupied some of the locations (different from those of the group) and everyone used one support for a drink. Is it possible that the group of friends sit at their locations warranting that everyone in the group can put his/her drink on a support that corresponds to his/her location? Clearly, sometimes the spectators that came before the group may have used the supports in such a way that not everyone in the group may use a support for his/her drink.

Input

There are several cases to consider. Each case begins with a line with two integer numbers R and C, indicating the number of rows and columns in the theater ($1 \leq R \leq 26$, $2 \leq C \leq 99$). The next line contains an integer number P indicating the number of persons that came to the theater before than the group ($0 \leq P \leq R \cdot C - 1$). Each one of the next P lines contains two text strings; the first one corresponds to the location id (one letter, one integer numeral) and, the second one is an one-character string with a sign `'-'` or a sign `'+'` denoting the fact that the person is using the left or the right support of his/her location, respectively. The next line contains an integer number Z indicating the number of persons in the group of friends ($1 \leq Z \leq R \cdot C - P$). Finally, each one of the following Z lines contains the locations' ids of the reserved seats for the group of friends. In each case, you may suppose that the given identifiers are distinct and that every drink was put in an empty support before the friends came.

Each line in a case description has no leading spaces and items on a line are separated by exactly one space. There are no spaces at the end of any line. Input ends with a line with two 0 values.

The input must be read from the file cinema.in.

Output

For each case output a line with one of the texts '`YES`' or '`NO`' depending on the fact that the group of friends can occupy their seats and use some of the supports to place their drinks with the constraints imposed by the already occupied seats and supports.

The output must be written to standard output.

Sample input	**Output for the sample input**

```
3 5
3
A3 -
B1 -
B4 -
4
A1
A2
B2
B3
3 5
3
A3 -
B1 +
B4 -
4
A1
A2
B2
B3
0 0
```
	```
	YES
	NO
	```

Problem D
Digit Sum

Source file name: `digsum.c, digsum.cpp` *or* `digsum.java`

The security system for *Associated Computer Informatics Systems* (ACIS) has an interesting test to check the identity for authorized personal. These persons have got a piece of software that allowed them to calculate, given two integer positive numbers M and N, what is the sum of the decimal digits in the sequence M, $M+1$, …, N. Then, when somebody is trying to access ACIS system, he/she is asked to answer the question of the sum for some M and N that are provided at that moment, by means of the given software.

ACIS programmers have developed a rather naïve algorithm only to verify that the method calculates the right answer. Now they are interested in developing a faster algorithm, in order to stop unauthorized users (who may be detected because they do not answer the sum question fast enough). And then you have been hired to help ACIS programmers to find such a method.

Input

The problem input consists of several cases, each one defined by a line with two integer numbers, M and N, without leading blanks and separated by a blank. You may assume that $1 \leq M \leq N \leq 10^9$. The end of the input is signaled by a line with two zero values.

The input must be read from the file digsum.in.

Output

For each case, output a line with the sum of the decimal digits for the sequence M,$M+1$,…,N.

The output must be written to standard output.

Sample input	**Output for the sample input**

```
3 8
5 18
1 50
0 0
```
	```
	33
	80
	330
	```

Problem E
LCD Extravaganza

Source file name: `lcd.c`, `lcd.cpp` *or* `lcd.java`

Liquid Crystal Developers (LCD) is a top-notch manufacturer of alarm clocks. Alarm clocks are one of the few things that have not changed dramatically since their invention. The market department at LCD has devised a plan to manufacture customizable alarm clocks in order to attract more customers. The new alarm clocks could be programmed by the owner to display digits with heterogeneous sizes, that is, display amplified versions of the traditional digits according to the user's preference.

A *traditional digit*, or simply a *digit*, is a nonnegative integer d ($0 \leq d \leq 9$). A digit is displayed in a 3×3 grid consisting of *dots* (decimal ASCII code 46), *underscores* (decimal ASCII code 95), and *bars* (decimal ASCII code 124), as depicted in the following figure:

```
._.  ...  ._.  ._.  ...  ._.  ._.  ._.  ._.  ._.
|.|  ..|  ._|  ._|  |_|  |_.  |_.  ..|  |_|  |_|
|_|  ..|  |_.  ._|  ..|  ._|  |_|  ..|  |_|  ._|

 0    1    2    3    4    5    6    7    8    9
```

An *amplified digit* is a pair (d, f) consisting of a digit d and a positive integer *factor* f ($0 \leq d \leq 9$, $f \geq 1$). An amplified digit (d, f) is displayed in a grid with $(2 \cdot f)+1$ rows and $f+2$ columns. Each such a grid consists of underscores, dots, and bars, in which underscores and bars are expanded horizontally and vertically, respectively, by a factor of f with respect to the display of digit d. Dots are used to fill any space in the grid not corresponding to an underscore or a bar. As an example, consider the following figure displaying five amplified digits:

```
                                    .______.
                                    |.......
                  ._____.           |.......
                  |......           |.......
......            |......           |.......
|....|            |......           |.......
|....|   .....    |......           |______.
|....|   ....|    |_____.           |......|
|____|   ....|    ......|   .__.    |......|
.....|   ....|    ......|   ...|    |......|
.....|   ....|    ......|   .__|    |......|
.....|   ....|    ......|   |...    |......|
.....|   ....|    ._____|   |__.    |______|

(4,4)    (1,3)    (5,5)     (2,2)   (6,6)
```

The new alarm clocks display sequences of amplified digits. The amplified digits in a display are aligned vertically with respect to their bottom rows. Moreover, in the display, each two consecutive amplified digits are separated by a column of dots and any space that does not contain an underscore or a bar is represented by a dot. The following figure displays the sequence $(4,4),(1,3),(5,5),(2,2),(6,6)$ of amplified digits:

```
...........................______.
..........................|.......
..............._____.......|.......
.............|............|.......
.............|............|.......
|....|.......|............|.......
|....|.......|............|______.
|....|.....|.|_____.......|......|
|____|.....|.......|..__..|......|
.....|.....|.......|....|.|......|
.....|.....|.......|..__|.|......|
.....|.....|.......|.|....|......|
.....|.....|.._____|.|__..|______|
```

As a member of the programming department at LCD, you have been assigned the implementation of an algorithm for sampling the display of the new alarm clocks.

Input

The input consists of several test cases. The first line in a test case contains a positive integer N ($1 \leq N \leq 10^4$), representing the number of amplified digits in the display. Each one of the following N lines contains two blank-separated integers d_i and f_i ($0 \leq d_i \leq 9$, $1 \leq f_i \leq 10^5$ for each $1 \leq i \leq N$), representing an amplified digit (d_i, f_i). The next line contains a nonnegative integer M ($0 \leq M \leq 10^4$), representing the number of samples for the display. Then M lines follow, each containing two blank-separated integers x_j and y_j ($0 \leq x_j < c$, $0 \leq y_j < r$ for each $1 \leq j \leq M$), representing a sample. The positive integers c and r denote the total number of columns and rows in the displayed sequence of amplified digits, respectively. The last test case is followed by a line containing a single zero value.

The input must be read from the file lcd.in.

Output

For each test case with N displayed amplified digits $(d_1, f_1), \ldots, (d_N, f_N)$ and M samples $(x_1, y_1), \ldots, (x_M, y_M)$, M lines must be printed, each containing either an underscore, a dot, or a bar. The j-th line in the output ($1 \leq j \leq M$) must identify the character at column x_j and row y_j in the display of $(d_1, f_1), \ldots, (d_N, f_N)$. Coordinate $(0,0)$ identifies the character in the leftmost column and bottom row (i.e., a coordinate represents a point in the first quadrant of the integer Cartesian coordinate system with origin in the leftmost column and bottom row).

The output must be written to standard output.

Sample input

```
5
4 4
1 3
5 5
2 2
6 6
7
5 0
6 0
13 5
33 12
32 12
5 7
5 8
3
1 2
1 1
7 4
4
13 8
13 7
0 8
7 1
0
```

Output for the sample input

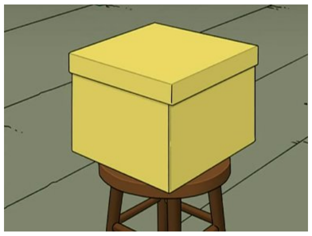
```
|
.
|
.
_
|
.
_
.
.
|
```

Problem F
The Farnsworth Parabox

Source file name: `parabox.c, parabox.cpp` *or* `parabox.java`

Professor Farnsworth, a renowned scientist that lives in year 3000 working at *Planet Express Inc.*, performed a failed experiment that nearly killed him. As a sub-product, some strange *boxes* were created. Farnsworth gave one of the boxes to Leela, who accidentally discovered that it leads to a *parallel universe*. After that, the *Planet Express* crew traveled to the new discovered parallel universe using the box, meeting their corresponding parallel copies, including a parallel Professor Farnsworth who also created some boxes.

Simultaneously, some parallel copies of the Professor created similar boxes in some existing parallel universes. As a result, some universes, including the original one, were endowed with a (possibly empty) collection of boxes leading to other parallel universes. However, the boxes have a bug: besides allowing travels among different parallel universes, they allow for time travels. So, a particular box leads to a distinct parallel universe possibly allowing a voyager to gain or lose a certain number of time units.


One of the boxes invented by Farnsworth Professor, from *Futurama*.
© *The Curiosity Company* and 20th *Century Fox.*

More precisely, given two distinct universes A and B, and a non-negative integer number t, a (A, B)-box with *time displacement* t is an object designed to travel between the two universes that can be used *directly* (traveling from A to B) or *reversely* (traveling from B to A). A such box exists in both universes, allowing travels among both universes. A voyager that uses the (A, B)-box directly can travel from universe A to universe B landing t time units in the future. On the other hand, a voyager that uses the (A, B)-box reversely can travel from universe B to universe A landing t time units in the past. Box building requires so much energy that there may be built at most one box to travel between a given pair of different universes.

A *circuit* is defined as a non-empty sequence of parallel universes $\langle s_1, s_2, \ldots, s_m\rangle$ such that:

- The first and the last universe in the sequence are the same (i.e., $s_1{=}s_m$).
- For every k ($1{\le}k{<}m$) there is a (s_k, s_{k+1})-box or a (s_{k+1}, s_k)-box to travel (directly or reversely) from universe s_k to universe s_{k+1}.

The possible existence of circuits is very interesting. Using the corresponding boxes of a circuit, a voyager may experiment real time travels. Professor Farnsworth wants to know if there is a circuit that starts in the original universe and allows travels to the past, constituting a phenomenon known as the *Farnsworth Parabox*. For example, imagine that there are three universes, A, B and C, and that there exist the following boxes: a (A, B)-box with time displacement 3, a (A, C)-box with time displacement 2, and a (B, C)-box with time displacement 4. Clearly, the sequence $\langle A, B, C, A\rangle$ is a circuit, that allows to travel five time units in the future, starting and ending at universe A.

The original Farnsworth Professor, who lives in the original universe, wants to know if the *Farnsworth Parabox* is true or not. Can you help him?

Input

There are several cases to solve. Each case begins with a line with two integer numbers N and B, indicating the number of parallel universes (including the original) and the number of existing boxes, respectively ($2 \le N \le 10^2$, $1 \le B \le N \cdot (N-1)/2$). The distinct universes are identified uniquely with the numbers $1, 2, \ldots, N$, where the original universe is the number 1. Each one of the next B lines contains three integer numbers i, j and t, describing a (i, j)-box to travel between the universe i and the universe j with time displacement t ($1 \le i \le N$, $1 \le j \le N$, $i \ne j$, $0 \le t \le 10^2$). The input ends with a line with two 0 values.

The input must be read from the file parabox.in.

Output

For each test case output one line with the letter 'Y' if the *Farnsworth Parabox* is true; or with the letter 'N', otherwise.

The output must be written to standard output.

Sample input	Output for the sample input

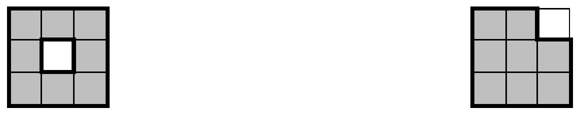
```
2 1
2 1 1
3 3
1 2 3
1 3 2
2 3 4
4 4
1 2 2
3 2 2
3 4 2
1 4 2
0 0
```
	```
	N
	Y
	N
	```

Problem G
Square Garden

Source file name: `garden.c, garden.cpp` *or* `garden.java`

The farmer Rick has a square garden of side L meters long, divided in a grid with L^2 square modules, each one of side 1 meter long. Rick wants to cultivate N modules of the garden and he knows that the production will be better if the cultivated area receives more water. He uses drip irrigation technology, which is done by means of hoses installed along the perimeter of the cultivated area.

It should be clear that there are different ways to choose the N modules that should be cultivated. The following figure shows two ways to cultivate N=8 modules in a square garden with side L=3. The left diagram shows a cultivated surface with a perimeter of length 16; the right one depicts another possibility, with perimeter 12.



Rick wants to maximize the perimeter of the selected area in order to optimize the production. Then you have been hired to help him determining the largest perimeter that an area of N modules of the garden may attain.

Input

There are several cases to consider. Each case is described with a line with two integer numbers L and N separated by one blank, indicating the length of the garden's side and the number of modules that must be cultivated, respectively ($1 \leq L \leq 10^6$, $0 \leq N \leq L^2$). Input ends with a line with two 0 values.

The input must be read from the file garden.in.

Output

For each case, output a line with the maximum perimeter that can be achieved.

The output must be written to standard output.

Sample input	**Output for the sample input**
`1 0`	`0`
`1 1`	`4`
`2 3`	`8`
`3 8`	`16`
`0 0`	

Problem H
Teamface

Source file name: `teamface.c`, `teamface.cpp` *or* `teamface.java`

Teamface is the new social network from NewPie High School, whose purpose is to increase collaboration among team members from all their sport teams. This new application is one of the new actions taken to conquer all national championships and ratify NewPie's place as the top school in the country.

In Teamface, members of teams can befriend each other under the following rules:

1. There are N students in NewPie teams ($N>0$), identified with numbers 1, 2, ..., N.
2. A student can belong to only one of the T NewPie teams ($T>0$).
3. All teams have the same number of members, M ($2 \leq M \leq N$).
4. Every team has exactly one *captain* (one of its members).
5. All members of a given team are Teamface friends among them. They can share strategies, post-match comments and watch replays online.
6. A student must have less than $\lfloor M/2 \rfloor$ friends not in his/her team (this is meant to prevent distractions).

Following the NewPie honor code, all players are registered in Teamface and have added their friends according to these rules.

This year's sponsors have requested the list of players for each team in order to send the training uniforms for the next training session, to be held on Monday. They need a list that contains, for each team, the number of training uniforms of each size. Information about team names and team captains was received correctly, but due to an administrative error at NewPie, the players' information was apparently incomplete. Indeed, NewPie sent, for each player, his/her identifier, size, and a list with the Teamface friends' identifiers.

Today is Saturday afternoon and nobody at NewPie can answer any question. But your boss, one of the NewPie sponsors, realized that it is possible to build the required list without any further information. He has chosen you to do this as soon as possible.

Input

There are several cases to solve. Each case begins with a line with two integer numbers T and N, indicating the number of teams and the number of students, respectively ($1 \leq T \leq 100$, $1 \leq N \leq 5000$). Then, T lines follow, each one containing a non-empty string w and an integer number i ($1 \leq i \leq N$), where w represents the name of a NewPie Team and i represents the identifier of its captain. You may assume that w does not contain any blank and that its length does not exceed 30 characters. This is followed by N lines, one per NewPie student. Each one of these lines contains the student's identifier j ($1 \leq j \leq N$), his/her uniform size as an integer s_j ($1 \leq s_j \leq 50$), an integer number f_j indicating how many friends j has ($0 \leq f_j < N$), and the list containing the identifiers of his/her Teamface friends (that list does not have repeated elements and does not include the identifier j). Every line in the input has no leading blanks and its corresponding data is separated by one blank. The input ends with a line with two 0 values.

The input must be read from the file `teamface.in`.

Output

The output for each case starts with "`Case` i`:`" on a single line, where i is the case number starting at 1. For each team in the test case (in the same order that they are given in the input), your program should output the name of the team on a single line. Then it should output a line with two integers t and n, where n is the number of uniforms needed for a given size t ($1 \leq t \leq 50$, $1 \leq n$), for all distinct sizes needed in the team. Sizes should be printed in ascending order.

The output must be written to standard output.

Sample input

```
3 12
LightningSalsa 1
PinkChiguiro 6
MightyPiranha 12
1 16 4 2 3 4 5
2 14 4 1 3 4 6
3 14 4 1 2 4 9
4 14 3 1 2 3
5 16 4 1 6 7 8
6 14 4 2 5 7 8
7 14 3 5 6 8
8 18 4 5 6 7 10
9 16 4 3 10 11 12
10 16 4 8 9 11 12
11 14 3 9 10 12
12 14 3 9 10 11
2 6
TamalSuperDragon 1
ArequipeNinjas 4
1 12 2 2 3
2 14 2 1 3
3 18 2 1 2
4 20 2 5 6
5 22 2 4 6
6 24 2 4 5
0 0
```

Output for the sample input

```
Case 1:
LightningSalsa
14 3
16 1
PinkChiguiro
14 2
16 1
18 1
MightyPiranha
14 2
16 2
Case 2:
TamalSuperDragon
12 1
14 1
18 1
ArequipeNinjas
20 1
22 1
24 1
```

Problem I
The Imperial Problem

Source file name: `imperial.c, imperial.cpp` *or* `imperial.java`

Since the times of the Mighty Roman Empire, it has been customary to have signals by the roads informing the distance to the greatest city on Earth (that is Rome), the location of a landmark along the road, and in more recent times, the maximum allowed speed. Making all these road signs can be a problem even for someone as mighty as the famous Emperor Tiberius: when he committed the construction of the *Via Solis*, he personally supervised the design of every road sign. Little did he know that he was making an infantile mistake because of a basic misunderstanding of how Roman numerals work.

The Roman number system is based on seven *basic* capital letters, where '`I`'=1, '`V`'=5, '`X`'=10, '`L`'=50, '`C`'=100, '`D`'=500, and '`M`'=1000. Like in the case of Arabic numerals, the decimal digits of a number to be represented with a Roman numeral are translated into Roman notation from left to right, starting with the most significant and finishing with the least significant digit. For example, the number 1111 is written as '`MCXI`', and 1055 is written as '`MLV`'. As the second example shows, the digit 0 does not appear.

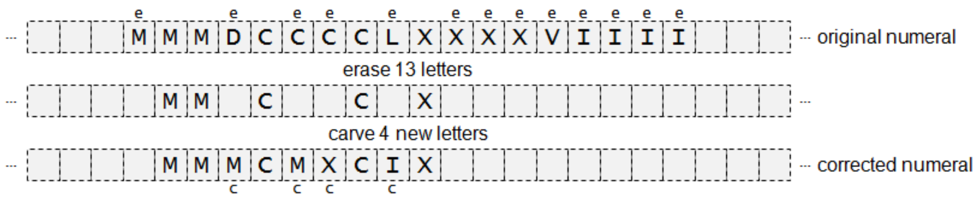
To translate a digit not defined by one of the basic capital letters, a system based on *repetition* and *addition* is used. For example, '`III`'=3, '`XXXII`'=32, and '`MMVIII`'=2008. Only the letters '`I`', '`X`', '`C`', and '`M`' can be repeated, and it does not make sense to repeat a letter more than four times because there is a shorter version for that (i.e., the expressions '`L`' and '`LX`' *must* be used instead of '`XXXXX`' and '`XXXXXX`', respectively).

The *final rule* states that no symbol can appear more than three times in a row, which happens whenever one of the decimal digits is a 4 or a 9. In those cases, a subtraction system *must* be used by combining the problematic symbol with a greater one. For example, 44 should not be written as '`XXXXIIII`' but as '`XLIV`' ((50−10)+(5−1) = 44). Note that '`I`' can only be placed at most once before '`V`' or '`X`', '`X`' can only be placed at most once before '`L`' or '`C`', and '`C`' can only be placed at most once before '`D`' or '`M`'. Because of this, the largest number that can be written using standard Roman numerals is '`MMMCMXCIX`' (3000+(1000−100)+(100−10)+(10−1) = 3999) and one should refrain from writing it as '`IMMMM`'.

The problems for Emperor Tiberius started when he designed all the road signs forgetting the final rule (e.g., instead of writing '`XLIV`', he wrote '`XXXXIIII`'). Many unnecessary letters were carved into the stone signs and the empire run into large overhead costs. For the sake of keeping his name among the greatest emperors of Rome, Tiberius wishes to amend his error. He has hired you to help him fixing his mistake. You must write a program that, given a Roman numeral, written forgetting the final rule (as Tiberius would write it), reports two integers e and c where e is the number of letters that must be *erased* and c is the number of letters that must be *carved* to transform that numeral into that one written using all the stated rules. If there are multiple solutions, your program must print the answer that minimizes the value of

e+c. If still there are two or more solutions minimizing e+c, the one that minimizes the value of e must be printed.

For example, considering a sign with the Roman numeral '`MMMDCCCCLXXXXVIIII`' (3999), it is easy to see that the minimum number of operations needed to fix it is e+c=17: e=13 erased letters plus c=4 new carved letters. The following artifact shows both the original Roman numeral and the corrected one. Erased letters have been marked with an '`e`' above the original numeral and the new carved letters with a '`c`' under the corrected one.



Note that it is not allowed to leave blank spaces inside a Roman numeral, and that every letter uses the same space when carved (i.e., all letters are written using a fixed-width font).

Input

The input consists of several lines, each one containing one Roman numeral written forgetting the final rule. No blanks are found on any of these lines. Input ends with a line containing a single asterisk ('`*`').

The input must be read from the file imperial.in.

Output

For each Roman numeral given in the input, output a single line with two non-negative integer numbers e and c, separated by one blank, where e is the number of erased letters and c is the number of new carved letters that solve the problem.

The output must be written to standard output.

Sample input	Output for the sample input
`IIII`	`3 1`
`CCCII`	`0 0`
`CCCCIII`	`3 1`
`CCCCXXXX`	`6 2`
`XXXXVIIII`	`7 2`
`MMMDCCCCLXXXXVIIII`	`13 4`
`*`	

Problem J
Lazy Professor

Source file name: `lazyprof.c`, `lazyprof.cpp` *or* `lazyprof.java`

Professor Yzal does not like to grade the exams of his students, so that he assigns this task to his assistants. Final students' grades are defined as weighted sums of the grades obtained in the exams. But Professor Yzal does not define *a priori* the exam's relative weights. Instead of that, only when the assistants end their job (and all the students' grades in each exam are known) Professor Yzal decides the exam's weights. He does it trying to be pleasant with the whole class, so that the assigned weights should maximize the average final grade that the class obtains, and expecting that eventual complaints about his grading criteria will be as few as possible. And finally, maybe rewarding harder exams, Yzal defines that the weight of every particular exam should lie within a specific range of values.

This term is about to finish and you were hired to help Yzal with his lazy grading job. Your task is to write a program that, given the students' grades and the *reasonable range* of weights for each exam, determines the maximum possible average according to the following rules:

- Each exam must have a weight, defined as an integer number in the closed interval $[0, 100]$, describing the assigned percentage.
- The weight of an exam i must be in an integer closed interval $[min_i, max_i]$ that describes the corresponding reasonable range previously defined by Yzal.
- The sum of all exam weights must be 100.
- The final grade of each student is a weighted sum calculated as the sum of his/her own grades previously multiplied by the corresponding exam weight divided by 100.
- The exam weights are so chosen that the average of the students' final grades is maximized.

Input

There are several cases to consider. Each test case is described as follows:

- A line with two integer numbers S and N, separated by a blank, ($1 \leq S \leq 100$, $1 \leq N \leq 20$), where S corresponds to the number of students and N corresponds to the number of exams in the course.
- Each one of the following S lines describes the grades of each student, containing N integer numbers $c_{j1}, c_{j2}, \ldots, c_{jN}$, separated by a blank, ($0 \leq c_{ji} \leq 100$ for each $1 \leq i \leq N$), where c_{ji} indicates the grade obtained by the student j in the exam i ($1 \leq j \leq S$, $1 \leq i \leq N$).
- Each one of the following N lines describes the reasonable range of weights of each exam, containing two integer numbers min_i, max_i, separated by a blank ($0 \leq min_i \leq max_i \leq 100$), where min_i and max_i represent the minimum and maximum weight that can be assigned to the exam i, respectively ($1 \leq i \leq N$).

You can suppose that

$$\sum_{i=1}^{N} min_i \leq 100, \quad \text{and} \quad \sum_{i=1}^{N} max_i \geq 100.$$

The last test case is followed by a line containing two zeros.

The input must be read from the file lazyprof.in.

Output

For each given case, output a single line with a number indicating the maximum possible average of the students' final grades if the weights are defined according to the rules. The answer should be formatted and approximated to two decimal places. The floating point delimiter must be '`.`' (i.e., the dot). The rounding applies towards the *nearest neighbor* unless both neighbors are equidistant, in which case the result is rounded up (e.g., 78.312 is rounded to 78.31; 78.566 is rounded to 78.57; 78.345 is rounded to 78.35, etc.).

The output must be written to standard output.

Sample input

```
1 1
0
0 100
2 2
50 90
70 50
0 100
0 100
2 2
50 90
70 50
30 70
30 70
2 2
50 90
70 50
50 50
50 50
2 2
73 52
92 81
20 50
60 80
0 0
```

Output for the sample input

```
0.00
70.00
67.00
65.00
72.90
```