acm International Collegiate Programming Contest 2007 IBM event sponsor

ACM International Collegiate Programming Contest 2007

South American Regional Contests

November 9-10, 2007

Contest Session

This problem set contains 10 problems; pages are numbered from 1 to 20.

This problem set is used in simultaneous contests hosted in the following countries:

- Argentina
- Bolivia
- Brazil
- Chile
- Colombia
- Peru
- Venezuela

Organization in Brazil:

Sociedade Brasileira de Computação

Sponsor in Brazil:

Fundação Carlos Chagas

Problem A
Ambiguous Codes

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

An extensive area of research in computer science is the field of communications. With computer networks being part of everyday life of many people, the development of ways for making networks faster, more reliable and secure is constantly needed. This practical need motivates an extensive research activity in the theory behind communications.

The very first thing needed to establish any kind of communication is a common code. A code is a way of changing the *form* of a piece of information into some other form, in general to make it possible to convey that piece of information from one place to another. Flag codes used by boats and the Morse code used in telegraphy are examples of codes for translating letters into different forms to enable communication over different media.

More formally, a code is a set of strings composed of symbols from one alphabet. Each string defined in the code is called a code word. A message is then composed concatenating a set of code words to convey the information needed. For example, in Morse code the alphabet is composed of the symbols hyphen and dot; letter "`S`" is represented by the code word "`...`", letter "`O`" is represented by the code word "`---`", and therefore the distress message "`SOS`" in Morse code is "`...---...`".

Codes for communication can have many desirable and undesirable properties such as ambiguity, entropy, redundancy, and many more. In this problem we will focus on ambiguity as a key property.

A code is *ambiguous* when there exists a message using that code that can be partitioned into different sequences of code words. In other words, in an ambiguous code a message may have more than one meaning. For example, consider the binary alphabet, composed of symbols {`0`,`1`}. For the code composed of the words $\{\texttt{10}, \texttt{01}, \texttt{101}\}$ the message `10101` can be understood as `10-101` or `101-01` and therefore the code is ambiguous. On the other hand, for the code composed of the words $\{\texttt{01}, \texttt{10}, \texttt{011}\}$ no ambiguous message exists and therefore the code is unambiguous.

As a part of the computer science community, you are required to develop a tester that checks if codes are ambiguous. In case a code is indeed ambiguous, you are also required to report the length (i.e. the number of symbols) of the shortest ambiguous message for that code.

Input

Each test case will consist on several lines. In all test cases the alphabet will be the set of hexadecimal digits (decimal digits plus the uppercase letters "`A`" to "`F`"). The first line of a test case will contain an integer N $(1 \leq N \leq 100)$, the number of code words in the code. Each of the next N lines describes a code word and contains a different and non-empty string of at most 50 hexadecimal digits.

Input is terminated by $N = 0$.

The input must be read from standard input.

Output

For each test case, output a single line with the length of the shortest ambiguous message for the provided code or -1 if the code is unambiguous.

The output must be written to standard output.

Sample input	**Output for the sample input**

```
3
10
01
101
3
AB
BA
ABB
0
```
	```
	5
	-1
	```

Problem B
Ballroom Lights

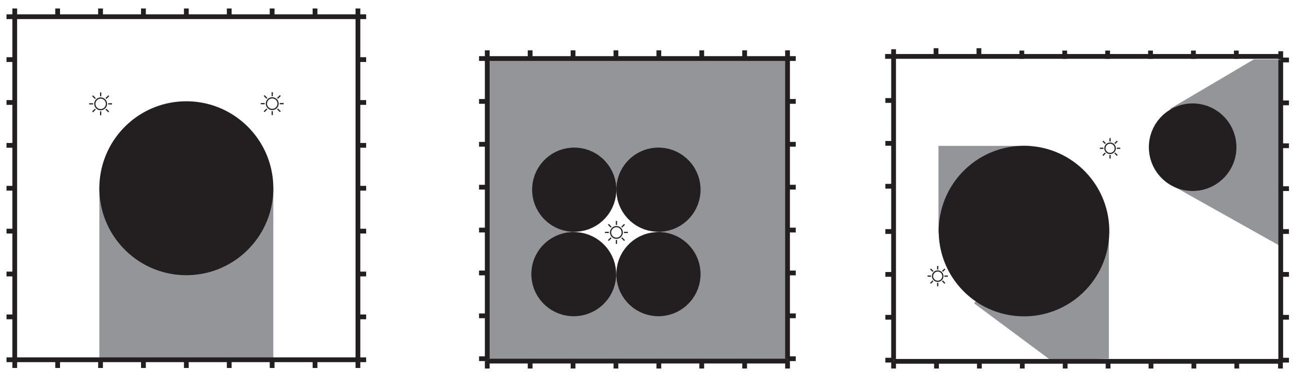
Source file name: `ballroom.c`, `ballroom.cpp` *or* `ballroom.java`

The ICPC world finals will be held in a luxurious hotel with a big ballroom. A buffet meal will be served in this ballroom, and organizers decided to decorate its walls with pictures of past champion teams.

In order to avoid criticism about favouring some of those teams over others, the organizing commitee wants to make sure that all pictures are appropiately illuminated. The only direct way they've found for doing this is ensuring each picture has at least one lightbulb that directly illuminates it.

In this way, the perimeter of the ballroom wall can be divided into illuminated parts (in which pictures may be placed) and dark parts (which are not suitable for placing the pictures).

The ballroom has the shape of a box and contains several lightbulbs. Each lightbulb emits light in all directions, but this light can be blocked by columns. All columns in the ballroom have cylindrical shape and go from the floor to the ceiling, so light cannot pass over or above them. Columns are of course placed so that its circular section is parallel to the ballroom floor. Any given point p on the perimeter wall is said to be illuminated if there exists a line segment (a light ray) which starts on a lightbulb, ends in p and does not touch or pass through any column.


Top view of 3 ballrooms with their lightbulbs, columns and illuminated and dark areas

Your task as a helper of the ICPC organization is to examine the blueprints of the ballroom and determine the total length of illuminated sections of the perimeter wall. The blueprint consist of a rectangle indicating a top view of the ballroom, with the lightbulbs and columns marked in it.

Input

Each test case will consist on several lines. The first line will contain four integers: L, the number of lightbulbs, C, the number of columns, X, the size of the ballroom on the x coordinate and Y, the size of the ballroom on the y coordinate. The lower-left corner of the ballroom is at $(0, 0)$ while the upper-right corner is at (X, Y).

The next L lines will contain two integers each representing the x and y coordinate of each lightbulb. The last C lines of the test case will contain three integers each, representing the x and y coordinates of the center of a column and its radius, in that order. You can assume that $1 \leq L, C \leq 10^3$ and $4 \leq X, Y \leq 10^6$. Also, for all pairs of coordinates (x,y), $0 < x < X$ and $0 < y < Y$, both for lightbulbs and column center locations. All radii of the columns will be positive. Finally, no two columns will overlap, although they may touch, and no column will touch or intersect with the border of the ballroom. No lightbulb will be inside a column or in its boundary and no two lightbulbs will be in the same place.

Input is terminated with $L = C = X = Y = 0$.

The input must be read from standard input.

Output

For each test case, output a single line with the total length of the illuminated parts of the perimeter wall. The result must be printed as a real number with exactly four decimal figures, with the lowest-order decimal figure rounded up.

The output must be written to standard output.

Sample input

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

Output for the sample input

```
28.0000
0.0000
25.8214
```

Problem C
Car Plates Competition

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

Martin and Isa are very competitive. The newest competition they have created is about looking at the plates of the cars. Each time one of them sees a car plate in the streets, he or she sends to the other an SMS message with the content of that plate; the one who has seen the newest plate is in the lead of the game. As the Automobile Car Management (ACM) office assigns the plates sequentially in increasing order, they can compare them and find out who is the winner.

Martin has a very smart eye and he has stayed on the lead for several weeks. Maybe he keeps looking at the streets instead of working, or maybe he stays all day in front of car selling companies waiting for new cars to go out with new plates. Isa, tired of being always behind, has written a program that generates a random plate, so the next time Martin sends a message to her, she will respond with this generated plate. In this way, she hopes to give Martin a hard time trying to beat her.

However, Martin has grown suspicious, and he wants to determine if Isa actually saw a car with the plate she sent or not. This way, he will know if Isa is in the lead of the game.

He knows some facts about the plates assigned by the ACM:

- Each plate is a combination of 7 characters, which may be uppercase letters (A-Z), or digits (0-9).
- There exists two kinds of plate schemes: the old one, used for several years, and the new one which has been in use for some months, when the combinations of the old one were exhausted.
- In the old scheme, the first three characters were letters, and the last four were digits, so the plates run from AAA0000 to ZZZ9999.
- In the new scheme, the first five characters are letters, and the last two are digits. Unfortunately the chief of ACM messed up with the printing system while he was trying to create a poster for his next campaign for mayor, and the printer is not able to write the letters A, C, M, I, and P. Therefore, in this new scheme, the first plate is BBBBB00, instead of AAAAA00.
- The plates are assigned following a sequential order. As a particular case, the last plate from the old scheme is followed by the first plate from the new scheme.

As Isa is not aware of all of this, she has just made sure that her random generator creates a combination consisting of seven characters, where the first three characters are always uppercase letters, the last two characters are always digits, and each one of the fourth and fifth characters may be an uppercase letter or a digit (possibly generating an illegal combination, but she has not much time to worry about that).

Of course, Martin will not consider Isa the winner if he receives an illegal combination, or if he receives a legal plate, but equal to or older than his. But that's not all of it. Since Martin knows that new plates are not generated too fast, he will not believe that Isa saw a car with a plate newer than the one he sent, but sequentially too far. For example, if Martin sends DDDDD45, and receives ZZZZZ45, he will not believe that Isa saw a car with that plate, because he knows that the ACM couldn't have printed enough plates to get to ZZZZZ45 in the time he received that answer.

So, Martin has decided to consider Isa the winner only if he receives a legal plate, newer than his, and older than or equal to the C-th consecutive plate after the one he sent. He calls C his *confidence number*. For example, if Martin sends ABC1234, and his confidence number is 6, he will think that Isa is the winner only if he receives any plate newer than ABC1234, but older than or equal to ABC1240.

Input

The input contains several test cases. Each test case is described in a single line that contains two strings S_M and S_I, and an integer C, separated by single spaces. S_M is the 7-character string sent by Martin, which is always a legal plate. S_I is the 7-character string answered by Isa, which was generated using her random generator. C is Martin's confidence number ($1 \leq C \leq 10^9$).

The end of input is indicated by $S_M = S_I =$"*" and $C = 0$.

The input must be read from standard input.

Output

For each test case, output a single line with the uppercase character "Y" if, according to Martin, Isa is the winner, and with the uppercase character "N" otherwise.

The output must be written to standard output.

Sample input	**Output for the sample input**
`ABC1234 ABC1240 6`	`Y`
`ABC1234 ABC1234 6`	`N`
`ACM5932 ADM5933 260000`	`N`
`BBBBB23 BBBBC23 100`	`N`
`BBBBB23 BBBBD00 77`	`Y`
`ZZZ9997 ZZZ9999 1`	`N`
`ZZZ9998 BBBBB01 3`	`Y`
`ZZZZZ95 ZZZZZ99 10`	`Y`
`BBBBB23 BBBBB22 5`	`N`
`* * 0`	

Problem D
Drop the Triples

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

The inhabitants of a small Caribbean island in the region known as Bermuda's Triangle love to spend their warm summer nights playing cards. As a tribute to the region where they live, all of their card games have some connection to triangles. One of the most popular games in the island is known as Triples, and has very simple rules.

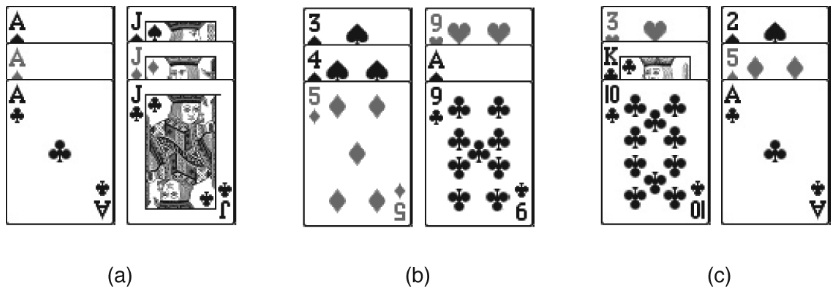
The game is played between two players, with a set of standard playing cards. Cards are distinguished only by their values, from 1 (Ace) to 13 (King). The cards are shuffled and placed as a pile in the center of the table, face down. This pile is called the *stock*. The two players take turns at playing. At each turn, a player

- draws the top card from the stock, adding it to her/his hand; and
- decides whether she/he wants to "drop some triples".

Dropping a triple consists of choosing three cards (a *triple*) from the hand and placing them on the table, face up. The dropped triples stay on the table until the end of the game. Only some sets of three cards form a valid triple. There are two types of valid triples:

- *Perfect triples* are made of three cards whose values represent the length of sides of an equilateral triangle;
- *Common triples* are made by three cards whose values represent the length of sides of any other (not equilateral) triangle.

The figure below shows examples of perfect triples (a), common triples (b), and invalid triples (c).



Only valid triples can be dropped, but a player may drop any number of triples at a given turn. In particular, since players know the number of cards in the stock at every turn, a player may decide to drop all triples in her/his last turn. Some players, however, normally drop some triples during the game, to maintain as few cards in their hands as possible.

The game finishes when the stock is empty. The winner is the player that dropped the largest number of perfect triples. If both players dropped the same number of perfect triples, the winner is the player that dropped the largest number of common triples. If both players dropped the same number of perfect triples and the same number of common triples, the result is a tie.

Given the description of the cards in the stock, write a program that determines the winner of a game of Triples, considering both players play as best as possible.

Input

The input contains several test cases. The first line of a test case contains one integer N representing the number of cards in the stock ($6 \leq N \leq 10^4$). The next line contains N integers X_i, separated by single spaces, representing the cards in the stock ($1 \leq X_i \leq 13$, for $1 \leq i \leq N$). The cards are given in the order they are drawn by the players: the first card in the input (X_1) is the first card drawn, the second card in the input (X_2) is the second card drawn, and so on. Several cards with the same value may be present in the stock, and not necessarily all card values are present in the stock. The end of input is indicated by $N = 0$.

The input must be read from standard input.

Output

For each test case your program must output a single line, containing '`1`' if the first player to play wins the game, '`2`' if the second player wins, or '`0`' if there is a tie.

The output must be written to standard output.

Sample input	**Output for the sample input**
`7`	`0`
`5 6 5 6 5 6 8`	`2`
`12`	`1`
`13 13 13 13 13 13 1 3 2 9 3 9`	
`12`	
`1 2 1 2 1 2 3 1 4 2 5 3`	
`0`	

Problem E
Emoticons :-)

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

Emoticons are used in chat and e-mail conversations to try to express the emotions that printed words cannot. This may seem like a nice feature for many, but a lot of people find it really annoying and wants to get rid of emoticons.

George is one of those people. He hates emoticons so bad, that he is preparing a plan to remove all emoticons from all e-mails in the world. Since you share his visionary plans, you are preparing a special program to help him.

Your program will receive the list of emoticons to proscribe. Each emoticon will be a string of characters not including any whitespace. You will also receive several lines of text. What you need to do is change some characters of the text into spaces to ensure no emoticon is left on the text. For an emoticon to be considered to appear in the text it has to appear in a single line and be made of consecutive characters.

To help George's plan remain secret as long as possible, you need to do your job with the minimum possible amount of character changes.

Input

The input file contains several test cases. Each test case consists of several lines. The first line of each test case will contain two integers separated by a single space: N, the number of emoticons to proscribe, and M, the number of lines the text has. The next N lines contain one emoticon each, a non-empty string of at most 15 characters. Each of the last M lines of the test case contains a line of text of at most 80 characters. You can assume $1 \leq N, M \leq 100$.

Valid input characters for emoticons are uppercase and lowercase letters, digits and the symbols "`!?.,:;-_'#$%&/=*+(){}[]`" (quotes for clarity). Each line of the text may contain the same characters with the addition of the space character.

The input is terminated by $N = M = 0$.

The input must be read from standard input.

Output

For each test case, output exactly one line containing a single integer that indicates the minimum number of changes you need to make to the entire text to ensure no emoticon on the list appears in it.

The output must be written to standard output.

Sample input	**Output for the sample input**

```
4 6
:-)
:-(
(-:
)-:
Hello uncle John! :-) :-D
I am sad or happy? (-:-(?
I feel so happy, my head spins
(-:-)(-:-)(-:-)(-:-) :-) (-: :-)
but then sadness comes :-(
Loves you, Joanna :-)))))
3 1
:)
):
))
:):)):)):)):(:((:(((:):)
0 0
```
	```
	11
	8
	```

Problem F
Finding Seats

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

A group of K friends is going to see a movie. However, they are too late to get good tickets, so they are looking for a good way to sit all nearby. Since they are all science students, they decided to come up with an optimization problem instead of going on with informal arguments to decide which tickets to buy.

The movie theater has R rows of C seats each, and they can see a map with the currently available seats marked. They decided that seating close to each other is all that matters, even if that means seating in the front row where the screen is so big it's impossible to see it all at once. In order to have a formal criteria, they thought they would buy seats in order to minimize the *extension* of their group.

The *extension* is defined as the area of the smallest rectangle with sides parallel to the seats that contains all bought seats. The area of a rectangle is the number of seats contained in it. They've taken out a laptop and pointed at you to help them find those desired seats.

Input

Each test case will consist on several lines. The first line will contain three positive integers R, C and K as explained above ($1 \leq R, C \leq 300$, $1 \leq K \leq R \times C$). The next R lines will contain exactly C characters each. The j-th character of the i-th line will be `'X'` if the j-th seat on the i-th row is taken or `'.'` if it is available. There will always be at least K available seats in total.

Input is terminated with $R = C = K = 0$.

The input must be read from standard input.

Output

For each test case, output a single line containing the minimum extension the group can have.

The output must be written to standard output.

Sample input

```
3 5 5
...XX
.X.XX
XX...
5 6 6
..X.X.
.XXX..
.XX.X.
.XXX.X
.XX.XX
0 0 0
```

Output for the sample input

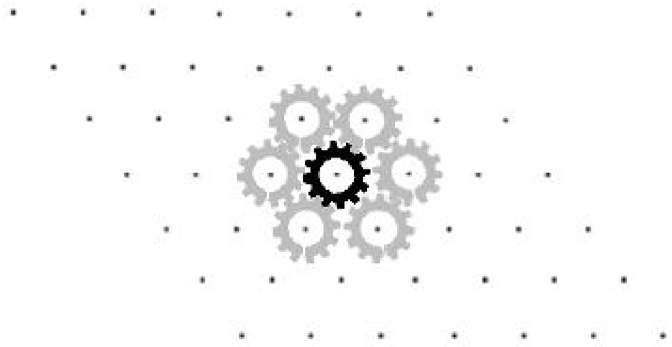
```
6
9
```

Problem G
Galou is back!

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

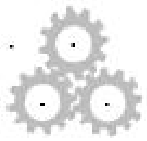
The famous witch is back. After killing an incredible amount of monsters in order to find a hidden treasure, Zak Galou decided to buy vineyards in Burgundy and retired. Everything was calm in his new life, until the day that his farm tractor stopped working.

His tractor's engine works based on a mechanism of gears. The engine can be represented by a bidimensional grid. At most one gear can be attached to each position of the grid. All the gears are identical and can mesh with adjacent gears. In this grid, a gear can have up to six other adjacent gears, see figure below:



Under normal utilization, when the tractor is started, some of the gears are initially activated and try to turn clockwise. When a gear tries to turn in one direction, all the adjacent gears try to turn in the opposite direction.

When Zak Galou opened his engine he noticed that it had been sabotaged (probably by a jealous treasure hunter who was not able to find the treasure). Some of the gears were removed from the engine and others have been added to it. As a consequence, some of the gears were immobile. A gear can be immobile either if it is *free* or if it is *blocked*. A gear is *free* when it is not an initially activated gear and no adjacent gear is trying to turn. A gear is *blocked* when it is trying to turn in both directions at the same time. For example, consider that there are three gears in the engine as shown in the figure below. If any of the gears is initially activated when the tractor is started, all of them will be blocked. If none of the gears are initially activated, all of them will be free.



As a part of the work of fixing his tractor, Zak Galou asks for your help to solve the following problem. Given the description of the engine and the gears that are initially activated in the clockwise direction, he wants to know for each gear, what is its state when the tractor is started: turn clockwise, turn counter-clockwise, free or blocked.

Input

The input contains several test cases. The first line of a test case contains two integers R and C, separated by a single space, representing respectively the number of rows and columns of the engine grid ($1 \leq R, C \leq 100$). The next R lines describe the engine. The i-th line represents the i-th row of the engine and contains C characters. The character "`.`" indicates that there is no gear in the corresponding position, the character "`*`" indicates that there is a gear that *is not* initially activated when the engine is started and an "`I`" indicates that there is a gear that *is* initially activated when the engine is started. Notice that, for simplicity reasons, the parallelogram representing the engine grid is described in the input as if it was a rectangle with each row left aligned. The end of input is indicated by $R = C = 0$.

The input must be read from standard input.

Output

For each test case, your program must output $R + 1$ lines. The first line must be empty; each of the following R lines must have C characters. The characters printed must represent the state of each position of the grid when the engine is started. Print a "`.`" if there is no gear in the position; a "`(`" if there is a gear turning in the clockwise direction; a "`)`" if there is a gear turning in the counter-clockwise direction, an uppercase "`F`" if there is a gear that is free and an uppercase "`B`" if there is a blocked gear.

The output must be written to standard output.

Sample input	**Output for the sample input**

```
4 3
...
.*.
.I.
...
4 4
....
.**.
.I..
..*.
0 0
```
	<pre>
	...
	.).
	.(.
	...
	
	....
	.BB.
	.B..
	..F.</pre>

Problem H
He is offside!

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

Hemisphere Network is the largest television network in Tumbolia, a small country located east of South America (or south of East America). The most popular sport in Tumbolia, unsurprisingly, is soccer; many games are broadcast every week in Tumbolia.

Hemisphere Network receives many requests to replay dubious plays; usually, these happen when a player is deemed to be offside by the referee. An attacking player is *offside* if he is nearer to his opponents' goal line than the second last opponent. A player is not offside if

- he is level with the second last opponent or
- he is level with the last two opponents.

Through the use of computer graphics technology, Hemisphere Network can take an image of the field and determine the distances of the players to the defending team's goal line, but they still need a program that, given these distances, decides whether a player is offside.

Input

The input file contains several test cases. The first line of each test case contains two integers A and D separated by a single space indicating, respectively, the number of attacking and defending players involved in the play ($2 \leq A, D \leq 11$). The next line contains A integers B_i separated by single spaces, indicating the distances of the attacking players to the goal line ($1 \leq B_i \leq 10^4$). The next line contains D integers C_j separated by single spaces, indicating the distances of the defending players to the goal line ($1 \leq C_j \leq 10^4$). The end of input is indicated by $A = D = 0$.

The input must be read from standard input.

Output

For each test case in the input print a line containing a single character: "`Y`" (uppercase) if there is an attacking player offside, and "`N`" (uppercase) otherwise.

The output must be written to standard output.

Sample input	**Output for the sample input**

`2 3`
`500 700`
`700 500 500`
`2 2`
`200 400`
`200 1000`
`3 4`
`530 510 490`
`480 470 50 310`
`0 0`
	`N`
	`Y`
	`N`

Problem I
ICPC Scoreboard

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

Charles is the contest director for the ICPC Tumbolian regional contest. His responsibility is ensuring the contest flows smoothly, that the contest rules are applied fairly, and, of course, announcing the final contest ranking.

According to ICPC rules, a team with more solved problems ranks above a team with less solved problems. If two teams have the same number of solved problems, the team with the smaller total penalty ranks above the team with the larger total penalty (in case both teams have the same number of solved problems and the same penalty, Charles considers them as tied).

The *total penalty* for a team is the sum of all the problem penalties of the problems that team has solved. The problem penalty for a problem is $TP + EP \times FA$, where TP is the time penalty for that problem, EP is the contest's error penalty and FA is the number of failed attempts at solving the problem before submitting a correct solution.

The *time penalty* for a problem is the time since the start of the contest, in minutes, that the team needed to solve the problem. The *error penalty* is a positive integer chosen by the contest director, designed to reward teams that submit correct solutions on the first attempt.

Charles wants to change the error penalty from the "standard" value of 20 minutes to stir things up. To study the effects of that change on the final rankings, he wants to know the range of error penalties that *don't* change the final standings.

In other words, if team A is ahead of team B in the original standings, then A should be ahead of B in the modified standings; if A and B are tied in the original standings, they should also be tied in the modified standings (the original standings are the ones obtained with an error penalty of 20 minutes).

Charles has been very busy organizing the Tumbolian regional, so he asked you to make a program that will compute the range for him.

Input

The input contains several test cases. The first line of each test case contains two integers T and P separated by a single space, indicating the number of teams and the number of problems, respectively ($2 \leq T \leq 100$, $1 \leq P \leq 10$). Each one of the next T lines describes the performance of a team. A team's performance description is a line containing P problem descriptions separated by single spaces. Teams are not necessarily given in order of their final standings.

Each problem description is a string "`A/S`", where A is an integer representing the number of attempts that the corresponding team made at solving that problem ($0 \leq A \leq 100$), and S is either "`-`", if the team did not solve that problem, or an integer indicating the number of minutes it took for the team to submit a correct solution ($1 \leq S \leq 300$). Attempts made after the first correct submission are not counted.

The end of input is indicated by $T = P = 0$.

The input must be read from standard input.

Output

For each test case in the input print two positive integers separated by a single space, indicating the smallest and largest error penalties that would not change the final ranking. If there is no upper bound for the error penalty, print a "*" instead of the upper bound.

The output must be written to standard output.

Sample input

```
5 3
0/- 0/- 0/-
2/- 2/- 1/-
1/60 1/165 1/-
1/80 0/- 2/120
0/- 1/17 0/-
4 2
17/- 5/-
2/7 3/-
3/- 2/-
1/15 0/-
3 2
1/- 2/15
2/53 1/17
1/70 1/20
0 0
```

Output for the sample input

```
1 24
9 *
20 20
```

Problem J
Justice League

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

Thirty five years ago, a group of super heroes was chosen to form the Justice League, whose purpose was to protect the planet Earth from the villains. After all those years helping mankind, its members are retiring and now it is time to choose the new members of the Justice League.

In order to keep their secret identity, let's say, secret, super heroes usually use an integer number to identify themselves. There are H super heroes on Earth, identified with the integers from 1 to H. With a brief look at the newspapers anyone can find out if two super heroes have already worked together in a mission. If this happened, we say that the two heroes have a *relationship*.

There must be only *one* Justice League in the world, which could be formed by any number of super heroes (even only one). Moreover, for any two heroes in the new league, they *must* have a relationship.

Besides, consider the set of the heroes not chosen to take part in the Justice League. For any two heroes on that set, they *must not* have a relationship. This prevents the formation of unofficial justice leagues.

You work for an agency in charge of creating the new Justice League. The agency doesn't know if it is possible to create the League with the restrictions given, and asked for your programming skills. Given a set of super heroes and their relationships, determine if it is possible to select any subset to form the Justice League, according to the given restrictions.

Input

The input is composed of several test cases. The first line of each test case contains two integers separated by a single space, H ($2 \leq H \leq 5 \times 10^4$) and R ($1 \leq R \leq 10^5$), indicating, respectively, the number of heroes and the number of relationships. Each of the following R lines contains two integers separated by a single space, A and B ($1 \leq A < B \leq H$), indicating that super hero A has a relationship with super hero B. Note that if A has a relationship with B, B also has a relationship with A. A relationship is never informed twice on a test case.

The end of input is indicated by $H = R = 0$.

The input must be read from standard input.

Output

For each test case in the input print a single line, containing the uppercase letter "`Y`" if it is possible to select a subset of heroes to form the Justice League according to the given restrictions, or the uppercase letter "`N`" otherwise.

The output must be written to standard output.

Sample input

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

Output for the sample input

```
Y
N
Y
```