MIUP 2011

Maratona Inter-Universitária de Programação

October 22, 2011 (10:00 to 15:00)

http://10.10.23.14/~miup/

Departamento de Engenharia Eletrónica e Informática
Faculdade de Ciências e Tecnologia
Universidade do Algarve

Contents

Team

Scientific Committee

- Francisco Coelho, Universidade de Évora
- Paul Crocker, Universidade da Beira Interior
- Carlos M. Fonseca, Universidade de Coimbra
- Pedro Guerreiro, Universidade do Algarve
- José Paulo Leal, Universidade do Porto
- Margarida Mamede, Universidade Nova de Lisboa
- Vasco Manquinho, Instituto Superior Técnico
- Fábio Marques, Universidade de Aveiro
- Rui Mendes, Universidade do Minho
- José Valente de Oliveira, Universidade do Algarve
- Delfim Torres, Universidade de Aveiro
- João Neto, Universidade de Lisboa
- André Restivo, Universidade do Porto
- Pedro Ribeiro, Universidade do Porto
- Fernando Silva, Universidade do Porto
- Simão Melo de Sousa, Universidade da Beira Interior
- Cristina C. Vieira, Universidade do Algarve

Organizing Committee

- Núcleo de Eletrónica e Informática da Universidade do Algarve
- José Bastos
- José Luís do Carmo
- Helder Daniel
- Pedro Guerreiro
- Fernando G. Lobo
- Artur Neves
- José Valente de Oliveira
- Cristina C. Vieira

Staff

- Núcleo de Eletrónica e Informática da Universidade do Algarve

Program

Location: Departamento de Engenharia Eletrónica e Informática, Faculdade de Ciências e Tecnologia, Universidade do Algarve

When	What	Where
09H00	Reception and registration	Departments' lobby (entrance)
09H30	MIUP opening session	Auditorium Teresa Gamito
09H45	Go to contest rooms	Rooms 1.59, 1.63
10H00	Contest beginning	Rooms 1.59, 1.63
15H00	Contest end	Rooms 1.59, 1.63
15H30	Problem discussion	Auditorium Teresa Gamito
16H30	Closing session	Auditorium Teresa Gamito

Preliminary Remarks

Event

MIUP is a programming contest for teams, of at most 3 elements, composed by higher education students of Portuguese Universities or Polytechnic Institutions. MIUP follows the International Collegiate Programming Contest (ICPC) rules.

The set of problems used in MIUP is of the authorship of the Scientific Committee members. During the 5 hours of the contest (10:00 till 15:00), the teams must submit their solutions to Mooshak, an automatic judge developed at University of Porto. Mooshak allows an immediate evaluation of the submissions, delivering different feedback results according to the solutions. These feedbacks are: *Accepted*, *Presentation Error*, *Runtime Error*, *Compile Time Error*, *Time Limit Exceeded*, *Memory Limit Exceeded*, *Wrong Answer* and *Invalid Function*. The contest accepteds the following programming languages: C, C++ and Java.

During the competition, the judges (members of the Scientific Committee) provide online support through Mooshak. The team that solves more problems is the winner. In case of a draw, Mooshak sums all the times needed for solving the problems, with 20 minutes penalty for each non-Accepted submission, and places first the teams that used less time.

Documentation

During the contest, documentation will be available at `http://10.10.23.14/miup2011/docs/`. Aside from the Mooshak platform, this is the only web link accessible for contestants.

Compilation

Constraints

- Maximum compilation time: 60 seconds
- Maximum source code size: 100kb
- Every source code must be submitted in a single file
- In case of Java submissions, the ".java" file has to have the same name as the class that contains the main method. There is no limit for the number of classes to be contained in that file.

Command line

- gcc -Wall -lm
- g++ -Wall -lm
- javac -nowarn (execution: java -Xss8M -Xmx64M)

Compiler versions

- gcc 4.4.5
- g++ 4.4.5
- java 1.6.0_26

IDEs and Environment

All machines run Debian 6.0 Squeeze Linux with graphical environments KDE and Gnome, have the user *miup* (password *miup2011*) and the following IDE/editors:

- netbeans
- eclipse
- kdevelop
- xemacs21
- kate
- gedit
- kwrite
- joe
- jed
- vim
- ddd

Runtime Constraints

- Max CPU time: 1 second
- Available memory for dynamic and global variables: 64 MB
- Available memory for local variables and execution stack: 8 MB

Input Specification

For each problem, there is a section named Input that describes exactly the details of the input test files that are used by Mooshak. Utmost attention must be paid to this section. Although there may be exceptions, the input files use the following guidelines:

- Generally, the first line of the input data file contains an integer indicating the number of test sets, or the size of the test at stake. However, this does not always happen.
- The last line of the file always terminates with a "\n" (new line character)
- Except when stated, white space is used as a separator. No line starts or ends with a whitespace, and there are no blank lines in the input, unless when explicitly stated.

To test your program you may build your own "input" files following the exact description of the Input section of the problem. You should compile your source code using the appropriate command specified in the "Command Line" section. To test it just as Mooshak, suppose a.out is the compiled source code and input1 is your test file corresponding to the sample input of the problem, then type:

$ a.out<input1

The result displayed on the console should be the expected sample output.

Output Specification

The output also has a specific section where you may find rigorous description of the format of the output. Any small detail may be a reason to obtain a Presentation Error or a Wrong Answer. Pay attention to the number of whitespaces or "\n" or capitalization.

To simulate the Mooshak evaluation, you may use the sample output in conjunction with the sample input, as follows:

- (command to produce the result)
  $ a.out<input1>output.mine
- (usage of diff to compare results)
  $ diff output.mine output.correct

Documentation

Online Help

Contestants have three sources of online help:

- Mooshak automatic help. Press the "help" button when you're logged in. This is a page that has information on available (memory and time) limits, compilers and a description of a number of Mooshak items (e.g. explanation of what a "Presentation Error" means).
- Mooshak online judge help (Questions). Press the "Ask" button and you can ask a question directly to the judge. Don't forget to select the correct problem (otherwise the judge won't know what your question refers to). And please check before in the "Question" section if there is information that already responds to your doubts.
- STL and Javadocs. Just access `http://10.10.23.14/miup2011/docs/`.

Contest Material

MIUP follows the ICPC SWERC's Rules. As such it is not allowed to bring reference materials (book, program listings or notes into the competition rooms). It is not allowed the usage of any electronic device. You should bring your own "dumb" pens and pencils. Paper is provided by the organization.

Each team may prepare a notebook of no more than 25 private A4 pages, font size greater or equal to 8. Each team member may use an exact copy of the notebook. The three copies of the notebook must be submitted at the registration desk on arrival. The organization will return them to the team at the beginning of the contest.

Problems

Problem A: Conveyor Belts

There are two parallel conveyor belts pushing, each, a sequence of products of varying value and type. Each conveyor belt can advance as far as you like but cannot move backwards. If, at the edge of the conveyors, there are two compatible products, i.e., two products with the same type, you got a pair and you can take both products together from the conveyor belts. The value of a pair is the sum of the two product values.

For reasons unknown, every product falling from a conveyor without being taken as part of a pair cannot be reused.

Task

Given two product sequences, return the maximum possible value you can take and the minimum number of pairs necessary to achieve it.

Input

The first line is an integer with the number of test samples. For each sample, the next line has the number of products of the first conveyor belt. Then, on each line, from first to last product, comes each product information: product name (a string), product type (an uppercase char) and product value (an integer). The same format follows to describe the products of the second conveyor belt.

Output

For each sample, a line consisting of: the maximum possible value, one space, the minimum number of pairs, a newline.

Constraints

- $0 <$ number of test samples ≤ 10
- $0 <$ length of product name ≤ 32
- $0 \leq$ number of products ≤ 1000
- $0 \leq$ product value $\leq 10^6$

Input example

```
3
4
nail B 5000
spoon A 1200
orange C 5
nail B 50
```

```
3
fork A 50000
hammer B 10
apple C 600
3
zorg X 500
xylf Y 50
krypt Z 450
3
xylf Y 50
tonite Z 450
lum X 500
1
a b 1
0
```

Output example

```
51805 2
1000 1
0 0
```

Problem B: Ant Travels in Bugniverses

Bugniverses are, excluding time, one-dimensional universes populated by dragonflys, grasshoppers and ants. Ants use dragonflys and grasshopers to travel (moving creates wormholes, so collisions are avoided). In each *bugniverse* there are two universal constants that determine the length of moves: grasshoppers always jump a meters and dragonflys always fly b meters (both moves either left or right).

The problem is, in some *bugniverses* not all traveling distances are possible. Suppose that an ant wants to travel from point A to point B, c meters apart (c can be positive or negative). If, for example, $a = 9$, $b = 6$ and the ant travels $x = 2$ grasshoper moves ($a \times x = 18$ meters) and $y = -1$ dragonfly moves ($b \times y = -6$ meters), ends at $c = 18 - 6 = 12$ meters away from the start point. However, in that universe, it is impossible to travel $c = 14$ meters!

Task

Given the *bugniverse*'s constants a, b and a travel distance c, the goal is to find out if either the travel is impossible and, if not, the *travel solution*: the *minimum non-negative* number of grasshopper moves, x, and the corresponding dragonfly moves, y, such that the travel ends c meters away from the start.

Input

The first line of the input has a single integer, n, giving the number of travels to solve. Each one of next n lines contains three integers, a, b and c separated by single spaces, describing a travel problem. In each line, the first integer, a, is the length of grasshopper moves, b of dragonfly moves and c the travel distance.

Output

The output has n lines. Each single line can be either the expression `impossible`, if the travel is impossible, or the two integers x and y, separated by a single space, with the *travel solution*. The first integer, x is the number of grasshoper moves and the second, y, the number of dragonfly moves.

Constraints

$$\begin{array}{rcccl} 0 & \leq & a, b & \leq & 2\,147\,483\,647 \\ -2\,147\,483\,647 & \leq & c & \leq & 2\,147\,483\,647 \\ 0 & \leq & x & & \end{array}$$

Input example 1

```
3
5 3 -5
9 6 14
1043207 871185 2114708923
```

Output example 1

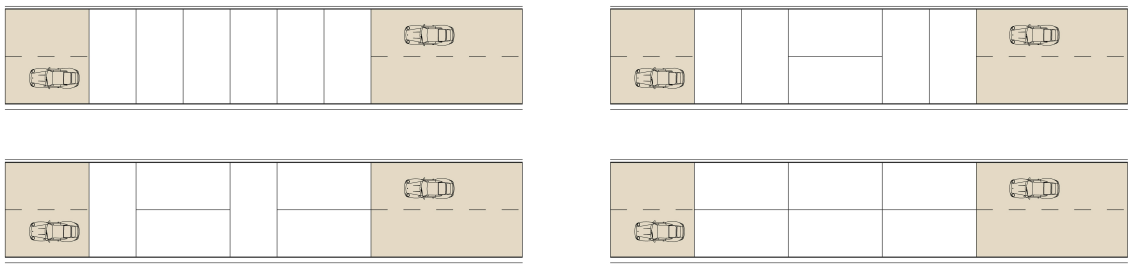
```
2 -5
impossible
255419 -303426
```

Problem C: Prince's Highway

The main highway in the Algarve is Via do Infante. It used to be free but not anymore, because the Portuguese government decided to charge tolls on all motorways. Most users were unhappy, other fiercely objected to this measure, but a creative group of drivers came up with a clever idea, that, if implemented, might raise enough money to cover road maintenance and other operating costs, thus allowing Via do Infante to remain free. The idea is to rent the road pavement for advertising. Each side of the road has two lanes. If we take a straight part of the road, then we could rent either lengthwise rectangles, along a single lane, or widthwise rectangles, across the road, covering both lanes. Each of these rectangles could contain a floor poster advertising some product or some company, thus providing a nice source of income that could avoid charging those atrocious tolls. For reasons of economy of scale, all rectangles must have the same area.

The question is: in a given stretch of the highway, how many different ways are there to cover the pavement with such rectangles?

For example, the figure below illustrates 4 of the 13 ways of covering a pavement of length 6 with *2-by-1* rectangles.



Task

Your task is to find how many ways there are to fill a *N-by-2* rectangle with *2-by-1* rectangles.

Input

The input file contains only a line with a single integer, N, representing the length of the stretch of the highway we are interested to rent for advertising.

Output

The output file contains a single line displaying the number of different ways of covering the given stretch of highway with *2-by-1* rectangles.

Constraints

- $N > 0$
- In all test cases, the result will not be larger than $2^{31} - 1$.

Input example

```
42
```

Output example

```
433494437
```

Problem D: 255 bfd0b7d49ae891e7cb98e2c7

Social networks are becoming increasingly used as a medium of communication. Adam and some of his geek friends, intensive users of these tools, decided to exchange messages amongst themselves using their preferred social network, but they do not want that anyone outside their geek circle understands the messages. For that purpose, they invented a new cryptographic algorithm, with which they will encipher their messages, so that only friends holding the key used for the cipher can decipher the messages.

Since the numeric codes of the characters are in the range [0..255], they decided to use the following function to scramble the numeric values involved in the computations:

$$\mathcal{F} : [0..255] \rightarrow [0..255]$$

$$\mathcal{F}(x) = \begin{cases} x - 10, & \text{if } x \text{ is even} \\ x + 4, & \text{if } x \text{ is odd} \end{cases}$$

In these expressions the plus sign represents addition modulus 256, and likewise for the minus sign. For example $\mathcal{F}(6) = 252$, $\mathcal{F}(253) = 1$.

The result of encrypting a string of characters $x = (x_1, x_2, \ldots, x_m)$ is a string of integers $y = (y_1, y_2, \ldots, y_m)$, computed as follows:

$$y_1 = \mathcal{F}(\mathcal{G}(x_1) \oplus key)$$
$$y_{i+1} = \mathcal{F}(\mathcal{G}(x_{i+1}) \oplus y_i)$$

In these expressions $\mathcal{G}(c)$ stands for the numeric value of the character c, a value in the range [0..255], $\oplus$ represents the bitwise exclusive or (the XOR operation in computer speak), and key is the secret key, also a value in the range [0..255].

The final step of the encrypting consists in creating a string of characters $(z_1, z_2, \ldots, z_{2n})$ where the pair of characters, z_{2i-1}, z_{2i} is the hexadecimal representation of the y_i. This hexadecimal string is then posted on the social network.

To exemplify the process let us suppose that Adam would like to cipher the following word "Lisboa" and uses the key 103. Then, applying the ciphering function he obtains the string of integers y = (47,60,83,53,80,53). Finally, the byte string is transformed in its hexadecimal representation "2f3c53355035" which could be posted.

Task

Your task is to create an application that converts the ciphered text back into the original text. To achieve this, you should reverse the encryption process.

Input

The input has 2 lines. The first line represents the key. The second line contains the ciphered text in hexadecimal notation.

Output

The output consists in a single line with the original text.

Constraints

- The original text has at most 2500 characters
- $0 \leq key \leq 255$

Input example 1

```
103
2f3c53355035
```

Output example 1

```
Lisboa
```

Input example 2

```
0
385b30064c345f2d4267
```

Output example 2

```
Boa Prova!
```

Input example 3

```
142
cfb2bce1c596acdda6cbef8d03582f4056187b0f3336572a4e2f5f
```

Output example 3

```
Esta e uma frase top secret
```

Problem E: To win, or not to win: that is the question

A large enterprise created a difficult multiple-choice e-quiz to promote a new product, promising an outstanding prize to the first contestants whose quizzes were totally correct. No-one could fill in the questionnaire twice.

When the competition ended, all contestants whose quizzes were correctly answered (henceforth called *aces*, to simplify) started to be collected. However, at that moment, the staff realised that they could not determine the exact UTC time of all answers. To tackle this problem, they processed the available information, creating *rules* of the form (a_1, a_2) to represent that they were absolutely certain that ace a_1 had answered before ace a_2. Now that winners must be announced, they still have not decided how to proceed. Nevertheless, aces have already been informed that prizes will be assigned strictly according to those rules. That is,

$$\text{for every rule } (a_1, a_2),\ \ a_2 \text{ can win a prize only if } a_1 \text{ wins a prize.}$$

So, whatever the final prizes assignment will be, it will assign all prizes to distinct aces and it will not break any rule. An ace *always wins* if, for every possible final assignment, he wins a prize. Similarly, an ace *never wins* if, for every possible final assignment, he does not win any prize.

As an example, let us consider that there are two prizes, four aces, denoted by x, y, z, w, and two rules, (x, y) and (y, z). Then, the two prizes can be assigned either to aces x and y, or to aces x and w. Therefore, ace x always wins a prize, ace z never wins any prize, whereas aces y and w may or may not win a prize.

Task

Given the number of prizes, the set of aces, and the set of rules, the goal is to find out whether an ace always wins and, if not, whether there is a chance that he wins a prize.

Input

The first line of the input contains three integers, P, A and R, separated by a single space. P is the number of prizes, A is the number of aces, and R is the number of rules. Aces are identified by integers, ranging from 0 to $A - 1$.

Each of the following R lines contains two integers, a_1 and a_2, which indicate that (a_1, a_2) is a rule. The two integers are separated by a single space.

The next line contains an integer T, which is the number of test cases, and each of the remaining T lines has also an integer, which represents the ace that wishes to know his chances of winning a prize.

Output

The output consists of T lines, one for each test case.

An output line contains: `Congratulations!`, if the ace always wins; `Sorry`, if the ace never wins; or `You have chances`, otherwise.

Constraints

- $1 \leq P < A$
- $2 \leq A \leq 5\,000$
- $1 \leq R \leq 30\,000$
- $1 \leq T \leq \min(A, 250)$

Input example 1

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

Output example 1

```
Congratulations!
You have chances
Sorry
You have chances
You have chances
```

Input example 2

```
2 5 2
4 3
3 2
5
0
1
2
3
4
```

Output example 2

```
You have chances
You have chances
Sorry
You have chances
You have chances
```

Problem F: Book Hand Delivery

John is part of a large reading group where members share their books with each other. This group never meets physically so they resort to sending the books by mail. The problem is that this solution is expensive and often books get lost.

So, when John discovered most members shared some activities together, he came up with an ingenious plan. John's idea was to deliver the books by hand, from one member to the other, until they reached their destination.

Another special thing about this group is that their schedule is always the same, week after week. John also noticed that each member only has one activity per day where they may meet another member. At this point you might be thinking that their lives are really monotonous and tedious. They are part of a reading group, what did you expect?

The following table shows an example of a possible timetable for the group.

Person	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Jonh	School	-	-	-	-	-	-
Mary	School	School	-	-	-	-	-
Oliver	-	School	-	School	-	-	-
Carl	Bar	-	School	School	-	-	-
Joan	Bar	-	-	-	School	-	-
Peter	Bar	-	-	Bar	School	-	-
Ashley	-	Bar	-	Bar	-	-	-
Megan	-	Bar	-	-	Bar	-	-
Emely	-	-	-	-	Bar	School	-
Robert	-	-	-	-	-	School	-

In this case, if John wants to deliver a book to Robert he can start by giving it to Mary when they both go pick up their kids at school. Mary can then deliver it to Oliver the next day. In two days Oliver can deliver the book to Carl and he can give it to Peter four days later when they meet at the bar. Three days after that Peter can give the book to Ashley and it will take another five days for the book to get into Megan's hands the next Tuesday. Megan can then deliver it to Emely on Friday, three days later, and the next day the book will finally reach Robert. This means it would take 19 days for the book to get from John to Robert ($0 + 1 + 2 + 4 + 3 + 5 + 3 + 1$). The next table depicts this plan.

John	Mary	Oliver	Carl	Peter	Ashley	Megan	Emely	Robert
-	Monday	Tuesday	Thursday	Monday	Thursday	Tuesday	Friday	Saturday
-	School	School	School	Bar	Bar	Bar	Bar	School
-	0	1	3	7	10	15	18	19

Task

Your task is to write a program that, given the group's timetable, computes the number of days necessary to take a book from one given person to another given person. The time starts counting on the first day the book changes hands.

Input

The first line of the input will contain a single integer (P) representing the number of persons in the group.

The next P lines will contain each person's schedule represented by 7 characters, separated with spaces, from 'A' to 'Z' or the character '-' if that person has no recorded activities on that day. The individual characters on the line are separated by single spaces.

This line will be followed by another integer (Q) representing the number of questions.

The next Q lines will each contain two different integers (S and D) representing a question: what is the minimum number of days necessary to get a book from the source (S) to the destination (D)? The first person will be refered to as person 0 (zero) and the last one will be person $P-1$.

Output

The output will contain Q lines, one for each question, containing either one integer representing the number of days it would take for the book to get from the source to the destination or the word 'impossible' if there is no way to get it there.

Constraints

$2 <= P <= 1000$
$1 <= Q <= 20$
$0 <= S, D <= P-1$
$S \neq D$

Input example 1

```
10
S - - - - - -
S S - - - - -
- S - S - - -
B - S S - - -
B - - - S - -
B - - B S - -
- B - B - - -
- B - - B - -
- - - - B S -
- - - - - S -
2
0 9
2 8
```

Output example 1

```
19
15
```

Input example 2

```
4
X - X R - A -
X K - R - Z -
X K - - - Z -
- - Y X - - -
2
0 2
1 3
```

Output example 2

```
0
impossible
```

Problem G: The Burrow Wheeler Transform

Burrow and Wheeler discovered in 1994 a clever but simple way to achieve very nice compression ratios when compressing blocks of data.

Basically, they proposes a very simple procedure to pre-process the blocks to be compressed returning a convenient word that enables simple compression methods to be used that give good compression rates.

Let us look at an example.

Imagine that you want to compress the word *abracadabra*. Burrow and Wheeler propose to compute the set of all the possible rotations (right to left rotations) of the word to be compressed. This gives:

0	abracadabra
1	bracadabraa
2	racadabraab
3	acadabraabr
4	cadabraabra
5	adabraabrac
6	dabraabraca
7	abraabracad
8	braabracada
9	raabracadab
10	aabracadabr

Then they tell us to sort this list of strings and to look at the word composed by the letters of the last column. We get:

0	10	aabracadab**r**
1	7	abraabrac**d**
2	0	abracadabr**a**
3	3	acadabraab**r**
4	5	adabraabra**c**
5	8	braabracad**a**
6	1	bracadabra**a**
7	4	cadabraabr**a**
8	6	dabraabrac**a**
9	9	raabracada**b**
10	2	racadabraa**b**

and the resulting word is *rdarcaaaabb*.

Surprisingly enough this process tends to gather the same letters in the same part of the resulting word (and this then offers a very competitive compression ratio using very simple algorithms).

But more interesting is the following fact: there is an **efficient way** to recover the original word *abacadabra* by knowing only the word *rdarcaaaabb* and the position of the original word *abacadabra* in the rotation list that gave *rdarcaaaabb*, i.e. **position** 2, in this example.

Task

Your task is compute the inverse transformation: given a preprocessed word and the position of the original word in the sorted rotation list write a program that computes the original word.

Input

The input contains two lines. The first line is the preprocessed word. The second line contains the position (an integer) of the original word in the ordered rotation list.

Constraints

The maximum size of a word to be compressed is 1000.

The ascii code of the input characters is exclusively included in the interval [33..126]. This constraint ensures that the given characters are human readable characters.

Output

A single line with the original word.

Sample Input

```
rdarcaaaabb
2
```

Sample Output

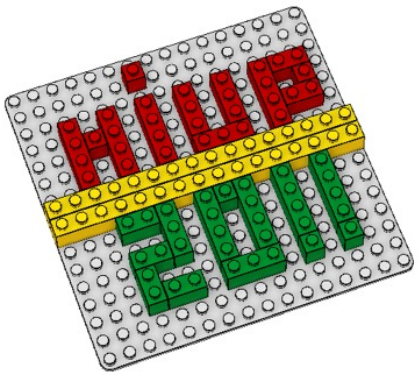
```
abracadabra
```

Problem H: Lego Mosaics

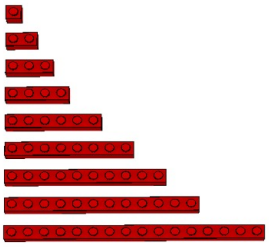
Building mosaics is an art dedicated to the construction of images by assembling small pieces of some material. Lego pieces are all about building things by assembling small pieces, and therefore they present the flexibility and versatility to create wonderful mosaics.

You have been commissioned to build a large lego mosaic. The basic idea is to use a lego plate and on the top of it attach some lego bricks. Each lego stud is like a pixel in the image. If we used text to represent a mosaic, with `'.'` representing a stud without a brick, `'R'` for a red brick, `'Y'` for a yellow brick and `'G'` for a green brick, then the mosaic of the image below would be represented by:

```
................
......R.........
................
..R.R.R.R.R.RRR.
..RRR.R.R.R.R.R.
..R.R.R.R.R.RRR.
..R.R.R.RRR.R...
YYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYY
...GGG.GGG.G.G..
.....G.G.G.G.G..
...GGG.G.G.G.G..
...G...G.G.G.G..
...GGG.GGG.G.G..
................
................
```



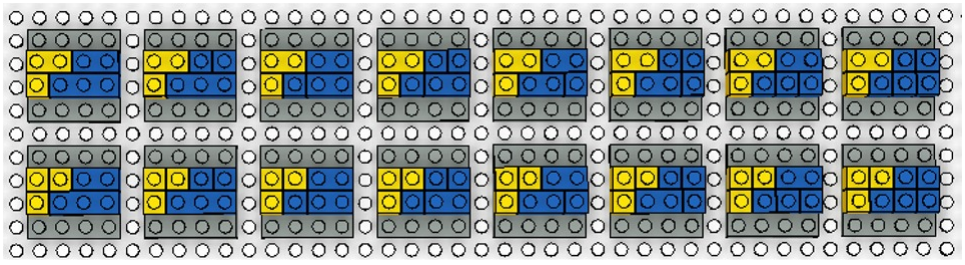
For the construction of the mosaic you have available a very large set of `1×N` bricks. In particular, for the purposes of the mosaic, you can assume you have an infinite number of 9 different types of bricks (in any needed color). These are `1×1`, `1×2`, `1×3`, `1×4`, `1×6`, `1×8`, `1×10`, `1×12` and `1×16` bricks, and are depicted in the figure on the right.



For aesthetically reasons, you only want to use the bricks in the horizontal positions, that is, parallel to the bottom of the plate. This means that when seen from the top, as in the textual representation above, the width of the bricks is variable, but the height is always `1`.

When you were starting the construction, you noticed that even with those constraints, there were several different ways of building the mosaic. For example, the mosaic below has 16 different ways of using pieces to obtain the exact same image (with `'B'` meaning a blue brick):

```
....
YYBB
YBBB
....
```



For a general case, can you tell in how many different ways you could build the desired mosaic?

Task

Given the description of a lego mosaic, your task is to compute in how many different ways you can build that mosaic assuming you have an infinite pool of 1×1, 1×2, 1×3, 1×4, 1×6, 1×8, 1×10, 1×12 and 1×16 lego bricks, in any color. The bricks must be positioned in horizontal positions.

Input

The first line of input contains two integers **R** and **C**, indicating respectively the number of rows and columns of the mosaic to be built.

The next **R** lines of input contain each exactly **C** characters detailing the mosaic to be built. Each character must either be `'.'` (representing a stud without any brick on top of it) or a capital letter (from `'A'` to `'Z'`) representing a stud of a given color. Two bricks with the same letter representing it have the same color. You can assume that there is at least one brick in the mosaic.

Output

The output should consist of a single integer **W** indicating the number of ways in which the mosaic can be built, given that you can only use bricks of type 1×1, 1×2, 1×3, 1×4, 1×6, 1×8, 1×10, 1×12 and 1×16 on horizontal positions. You can assume you always have enough bricks to build the mosaic using any combination of the bricks you need.

Constraints

$1 \leq \mathbf{R}, \mathbf{C} \leq 1,000$	Number of rows and columns of the mosaic
$1 \leq \mathbf{W} \leq 2,000,000,000$	Number of different ways to build the mosaic

Input example 1

```
4 4
....
YYBB
YBBB
....
```

Output example 1

```
16
```

Input example 2

```
3 6
GGRRRR
GYYRRR
GGRRRR
```

Output example 2

```
2048
```

Problem I: Watermelons

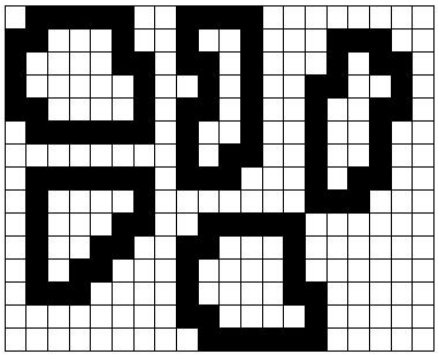
We want our watermelons juicy, sweet and smooth. Juiciness and sweetness are obvious qualities, but smoothness is also highly commendable. Smoothness is about the skin of the watermelon: when we hold them and caress them, we want to feel a firm, slightly humid skin, without any irregularities or soft spots. Of course!

In our watermelon farm, workers pick the watermelons when they are ripe and carry them to the production plant, where the watermelons are sorted and controlled for quality and smoothness. These operations are entirely automatic. The watermelons travel on a conveyor belt and are scanned by a robotic camera. Those who are perfectly smooth pass and those who are not are destroyed by a powerful laser beam that makes them explode (not a nice view, I can tell you ...)

You are called to write the program that analyses the video image, identifies the unsmooth watermelons, directs the laser beam at them and fires.

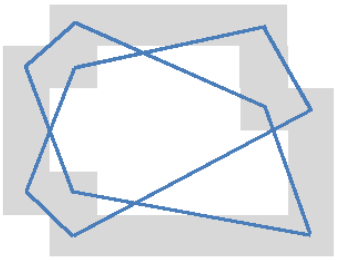
Task

Your task is to write a program that reads from the standard input the description of an image of the watermelons on the conveyor belt and identifies those that are not smooth. On the image, only the contours of the watermelons are displayed. Each contour is a sequence of connected squares in a grid, which approximates the actual contour. Each square is connected to exactly two other squares, which lay (each of them) either on the same row or in the same column as itself. The following figure represents an image with five watermelons.



Mathematically, the criteria for smoothness of a continuous closed curve could be explained by taking the tangent to each point of the curve, starting at a some point, and moving clockwise (or counterclockwise) along the contour. If the watermelon is smooth, the slope of the tangent would decrease until minus infinity, and then decrease from plus infinity to minus infinity again and then and finally decrease from plus infinity until the initial value at the initial point. This is if we go around the contour in the clockwise direction. If we go counterclockwise, the slope increases to plus infinity, then from minus infinity to plus infinity, etc. Note that we are being informal: if the tangent does not exist, then the curve is not smooth; and "decrease until minus infinity" does not mean that the slope strictly decreases: it may remain constant for a part of the curve that is perfectly straight. In other words, again informally, what we mean is that from one point to the next (so to speak ...), the slope of the tangent either decreases or stays the same. The same applies, *mutatis mutandis*, when we go counterclockwise (and the slope increases).

On a ragged contour, like the one in the image, we replace the slope of the tangent by the slope of segments taken along the contour. Let p_1, p_2, p_3, ..., p_N, be the N points in the contour where the contour changes direction (either from horizontal to vertical or from vertical to horizontal). Then instead of an infinite number of tangents we only have to consider the n segments defined by the points $< p_1, p_3 >$, $< p_2, p_4 >$, ..., $< p_{N-1}, p_1 >$ and $< p_N, p_2 >$. If the slopes of these segments behave as the tangents in a curve then the watermelon is smooth. Otherwise it is not and should be destroyed. The figure below shows the contour of one of the watermelons from first figure and displays the segments that represent the tangents.



Input

Each input file represents the image of set of melons on the conveyor belt. The first part of the input file contains two integers, representing the number of rows, R, and the number of columns, C, in the image. Exactly R lines follow, each line having exactly C characters. Each character is either 'M', representing a square in the contour of one watermelon, or '-' (the minus sign) for all other cases (squares either inside or outside contours). If there is an 'M' in the position $< x, y >$ (column x, row y), then exactly two of the four positions $< x+1, y >$, $< x, y+1 >$, $< x-1, y >$ and $< x, y-1 >$ also contain an 'M'. Thus, each 'M' belongs to a closed contour. All positions inside such a contour contain a minus sign. Note that watermelons can touch each other at corners, as exemplified in the first figure.

Output

The output contains one line for each unsmooth watermelon, displaying the horizontal and vertical coordinates of the center of the watermelon, or the message "ALL SMOOTH" (in uppercase, without the quotes) if all the watermelons are smooth. The lower left corner has coordinates $< 0, 0 >$. Horizontal coordinates grow rightwards and vertical coordinates grow upwards. The horizontal coordinate of the center of the watermelon is the average of the horizontal coordinates of all the dots in the contour, and likewise for the vertical coordinate. This average is computed using integer division. Theoretically, the center of a watermelon may lie outside the contour, in which case the laser beam will fail to blow out the watermelon, but such a bizarre watermelon has never been seen in nature. The pairs of coordinates are to be sorted by the vertical coordinate and then (in case of a tie) by the horizontal coordinate.

Constraints

- Number of rows, $R : 2 < R \leq 256$
- Number of columns, $C : 2 < C \leq 256$
- Number of watermelons per test case (not used in the problem description), $W : 1 \leq W \leq 100$
- Length of watermelon contour (not used in the problem description), $L : 4 \leq L \leq 1000$

Input example 1

```
15 20
-MMMMM--MMMM--------
MM---M--M--M---MMM--
M----MM-MM-M---M-MM-
M-----M--M-M--MM--M-
MM----M-MM-M--M--MM-
-MMMMMM-M--M--M--M--
--------M-MM--M--M--
-MMMMMM-MMM---M-MM--
-M----M-------MMM---
-M---MM--MMMMM------
-M--MM--MM---M------
-M-MM---M----M------
-MMM----M----MM-----
--------MM----M-----
---------MMMMMM-----
```

Output example 1

```
10 2
15 9
9 10
```

Input example 2

```
9 17
---MMM-----MMM---
--MM-MM----M-M---
-MM---MM---MMM---
MM-----MM--------
M-------M---MMMMM
MM-----MM--MM---M
-MM---MM---M---MM
--MM-MM----MMMMM-
---MMM-----------
```

Output example 2

```
ALL SMOOTH
```