In the popular show "Dinner for Five", five contestants compete in preparing culinary delights. Every evening one of them makes dinner and each of other four then grades it on a scale from 1 to 5.

The number of points a contestant gets is equal to the sum of grades they got. The winner of the show is of course the contestant that gets the most points.

Write a program that determines the winner and how many points they got.

INPUT

Five lines, each containing 4 integers, the grades a contestant got.

The contestants are numbered 1 to 5 in the order in which their grades were given.

The input data will guarantee that the solution is unique.

OUTPUT

Output on a single line the winner's number and their points, separated by a space.

EXAMPLES

input	input
5 4 4 5 5 4 4 4 5 5 4 4 5 5 5 4 4 4 4 5	4 4 3 3 5 4 3 5 5 5 2 4 5 5 5 1 4 4 4 4
output	**output**
4 19	2 17

Luka is fooling around in chemistry class again! Instead of balancing equations he is writing coded sentences on a piece of paper. Luka modifies every word in a sentence by adding, after each vowel (letters 'a', 'e', 'i', 'o' and 'u'), the letter 'p' and then that same vowel again.

For example, the word "kemija" becomes "kepemipijapa" and the word "paprika" becomes "papapripikapa". The teacher took Luka's paper with the coded sentences and wants to decode them.

Write a program that decodes Luka's sentence.

INPUT

The coded sentence will be given on a single line. The sentence consists only of lowercase letters of the English alphabet and spaces. The words will be separated by exactly one space and there will be no leading or trailing spaces. The total number of character will be at most 100.

OUTPUT

Output the decoded sentence on a single line.

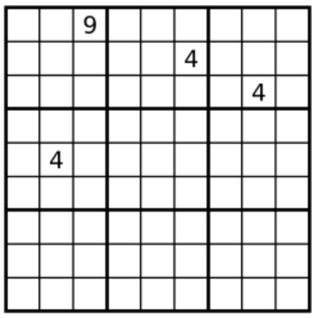
EXAMPLES

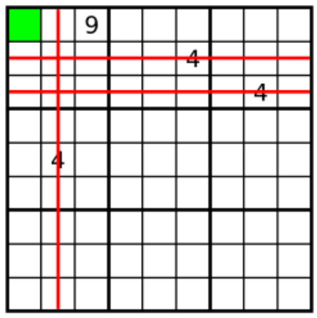
input
 	
`zepelepenapa papapripikapa`
 	
output
 	
`zelena paprika`
	input
	
	`bapas jepe doposapadnapa opovapa kepemipijapa`
	
	output
	
	`bas je dosadna ova kemija`

In the game of Sudoku, the objective is to place integers between 1 and 9 (inclusive) into a 9x9 grid so that each row, each column, and each of the nine 3x3 boxes contains all nine numbers. The starting board is partially filled in so that it is possible to logically deduce the values of the other cells. Sudoku puzzles range in difficulty, and complex analysis methods are required to solve the hardest puzzles. In this problem, however, you will implement one of the simplest methods, cross-hatching.

In cross-hatching, we select one of the nine numbers and, for each of its occurrences in the grid, cross out the corresponding row, column and 3x3 box. Now look for any 3x3 boxes where there is only one possible placement for the number and place it there.

The first image below shows a very sparsely filled in Sudoku grid. However, even in this grid it is possible to deduce using cross-hatching that the number in the top left cell is 4, as illustrated in the second image.





You will be given a partially filled-in grid. Your task is to repeatedly apply the cross-hatching method for different numbers until no more deductions can be made about any number.

The initial placement of the numbers in the grid may be invalid. It is also possible that there will be no available cell for a number in a 3x3 box. In both cases, you are to report an error.

INPUT

Input will consist of 9 lines, each containing exactly 9 characters. Each character will either be a digit between 1 and 9, or a period ('.') denoting an empty cell.

OUTPUT

If the input is valid and there is no contradiction while solving, you should output the grid in the same format it was given in, with cells filled in if their value can be deduced using cross-hatching. Otherwise, output "ERROR" (quotes for clarity).

EXAMPLES

input	**input**	**input**	**input**
..9...... .....4... ....4. .... .4.......	...1...6. 18...9... ..7.642.. 2.9..6.5. .43...72. .6.3..9.1 ..265.1.. ...2...97 .5...3...	1........ ..1...... ....1.	........2 ....1.... 1........ ......1.. ....1. ....
output	**output**	**output**	**output**
4.9...... .....4... ....4. .... .4.......	524137869 186529473 397864215 219476358 843915726 765382941 972658134 638241597 451793682	ERROR	ERROR

A matrix is a rectangular table of letters. A square matrix is a matrix with an equal number of rows and columns. A square matrix M is called **symmetric** if its letters are symmetric with respect to the main diagonal ($M_{ij} = M_{ji}$ for all pairs of i and j).

The following figure shows two symmetric matrices and one which is not symmetric:

`AAB`
`ACC`
`BCC`
	`AAA`
	`ABA`
	`AAA`
		`ABCD`
		`ABCD`
		`ABCD`
		`ABCD`
			`AAB`
			`ACA`
			`DAA`

Two symmetric matrices.		Two matrices that are not symmetric.	

Given a collection of available letters, you are to output a **subset of columns** in the **lexicographically smallest symmetric** matrix which can be composed using **all** the letters.

If no such matrix exists, output "IMPOSSIBLE".

To determine if matrix A is lexicographically smaller than matrix B, consider their elements in row-major order (as if you concatenated all rows to form a long string). If the first element in which the matrices differ is smaller in A, then A is lexicographically smaller than B.

INPUT

The first line of input contains two integers N ($1 \le N \le 30000$) and K ($1 \le K \le 26$). N is the dimension of the matrix, while K is the number of distinct letters that will appear.

Each of the following K lines contains an uppercase letter and a positive integer, separated by a space. The integer denotes how many corresponding letters are to be used. For example, if a line says "A 3", then the letter A must appear three times in the output matrix.

The total number of letters will be exactly N^2. No letter will appear more than once in the input.

The next line contains an integer P ($1 \le P \le 50$), the number of columns that must be output.

The last line contains P integers, the indices of columns that must be output. The indices will be between 1 and N inclusive, given in increasing order and without duplicates.

OUTPUT

If it is possible to compose a symmetric matrix from the given collection of letters, output the required columns on N lines, each containing P character, without spaces. Otherwise, output "IMPOSSIBLE" (quotes for clarity).

SCORING

In test cases worth 60% of points, N will be at most 300.

In test cases worth 80% of points, N will be at most 3000.

EXAMPLES

input 3 3 A 3 B 2 C 4 3 1 2 3 **output** AAB ACC BCC	**input** 4 4 A 4 B 4 C 4 D 4 4 1 2 3 4 **output** AABB AACC BCDD BCDD	**input** 4 5 E 4 A 3 B 3 C 3 D 3 2 2 4 **output** AC BE DE ED	**input** 4 6 F 1 E 3 A 3 B 3 C 3 D 3 4 1 2 3 4 **output** IMPOSSIBLE

A binary search tree is a tree in which every node has **at most** two children nodes (a left and a right child). Each node has an integer written inside it. If the number X is written inside a node, then the numbers in its left subtree are less than X and the numbers in its right subtree are greater than X.

You will be given a sequence of integers between 1 and N (inclusive) such that each number appears in the sequence exactly once. You are to create a binary search tree from the sequence, putting the first number in the root node and inserting every other number in order. In other words, run insert(X, root) for every other number:

```
insert( number X, node N )
    increase the counter C by 1
    if X is less than the number in node N
        if N has no left child
            create a new node with the number X and set it to be the left child of node N
        else
            insert(X, left child of node N)
    else (X is greater than the number in node N)
        if N has no right child
            create a new node with the number X and set it to be the right child of node N
        else
            insert(X, right child of node N)
```

Write a program that calculates the value of the counter C after every number is inserted. The counter is initially 0.

INPUT

The first line contains the integer N ($1 \le N \le 300000$), the length of the sequence.

The remaining N lines contain the numbers in the sequence, integers in the interval [1, N]. The numbers will be distinct.

OUTPUT

Output N integers each on its own line, the values of the counter C after each number is inserted into the tree.

SCORING

In test cases worth 50% of points, N will be at most 1000.

EXAMPLES

input 4 1 2 3 4 **output** 0 1 3 6	**input** 5 3 2 4 1 5 **output** 0 1 2 4 6	**input** 8 3 5 1 6 8 7 2 4 **output** 0 1 2 4 7 11 13 15

A road network in a country consists of N cities and M one-way roads. The cities are numbered 1 through N. For each road we know the origin and destination cities, as well as its length.

We say that the road F is a **continuation** of road E if the destination city of road E is the same as the origin city of road F. A **path** from city A to city B is a sequence of road such that origin of the first road is city A, each other road is a continuation of the one before it, and the destination of the last road is city B. The length of the path is the sum of lengths of all roads in it.

A path from A to B is a **shortest** path if there is no other path from A to B that is shorter in length.

Your task is to, for each road, output **how many different** shortest paths containing that road, modulo 1 000 000 007.

INPUT

The first line contains two integers N and M ($1 \leq N \leq 1500$, $1 \leq M \leq 5000$), the number of cities and roads.

Each of the following M lines contains three positive integers O, D and L. These represent a one-way road from city O to city D of length L. The numbers O and D will be different and L will be at most 10000.

OUTPUT

Output M integers each on its own line – for each road, the number of different shortest paths containing it, modulo 1 000 000 007. The order of these numbers should match the order of roads in the input.

SCORING

In test cases worth 30% of points, N will be at most 15 and M will be at most 30.

In test cases worth 60% of points, N will be at most 300 and M will be at most 1000.

EXAMPLES

input 4 3 1 2 5 2 3 5 3 4 5 **output** 3 4 3	**input** 4 4 1 2 5 2 3 5 3 4 5 1 4 8 **output** 2 3 2 1	**input** 5 8 1 2 20 1 3 2 2 3 2 4 2 3 4 2 3 3 4 5 4 3 5 5 4 20 **output** 0 4 6 6 6 7 2 6

Mirko purchased a new microprocessor. Unfortunately, he soon learned that many of his programs that he wrote for his old processor didn't work on the new processor.

Deep inside the technical documentation for both processors, he found an explanation. In order to work faster, the new processor imposes certain constraints on the **machine code** of programs, constraints that never existed on the previous model.

The machine code of a processor consists of instructions that are executed sequentially. Each instruction uses a byte of memory. Also, instructions can have zero or more parameters, each of which uses an additional byte of memory. In machine code, parameters immediately follow an instruction.

When formatted as text, machine code instructions are uppercase letters, while parameters are lowercase letters. For example:

A	b	c	b	B	c	c	C	D	e	f	g	h

This program consists of four instructions; the first takes three parameters, the second two, the third none and the fourth takes four parameters. The program uses 13 bytes of memory.

The new processor model fetches memory in four-byte chunks so each instruction must start at a memory address that is **divisible by four** (the first byte in memory is address 0). To achieve that, we can insert NOP (no operation) instructions into the old program, instructions that do nothing and are not limited to memory locations divisible by four. The above program, adapted to run on the new processor, can look like this:

A	b	c	b	B	c	c	NOP	C	NOP	NOP	NOP	D	e	f	g	h

The instructions A, B, C and D are now at memory locations 0, 4, 8 and 12, which satisfies the processor's constraints.

Write a program that determines the **smallest number of NOP instructions** that need to be inserted for the given program to work on the new processor model.

INPUT

The input contains the machine code of the program written for the old processor model. The program will consist of at most 200 English letters.

The program will always start in an instruction i.e. the first letter in the machine code will be uppercase. If an instruction appears more than once in the machine code, it will always take the same number of parameters.

OUTPUT

Output the smallest number of NOP instructions needed to adapt the program for the new processor.

EXAMPLES

input	input	input
Abcd	EaEbFabG	AbcbBccCDefgh
output	**output**	**output**
0	5	4

Rock-paper-scissors is a popular two-player game. In the game, each of the players uses their hand to show one of three symbols: rock, paper or scissors. If both players show the same symbol, the game is a tie. Otherwise, scissors beat paper, paper beats rock and rock beats scissors.

Sven has been studying the psychological intricacies of the game for years and has become a real master at the game, his friends not standing a chance against him in one-on-one games.

With the world championships around the corner, Sven is practicing his skills playing simultaneous games with N of his friends. One such game consists of R rounds. In each round, Sven and each of his friends show one of the three symbols.

When calculating the score, in each round, Sven's symbol is independently compared to each of his friends' symbols. Sven scores two points for every win and one point for every tie. Sven does not get points for losing.

Write a program that calculates Sven's total score, and also his largest possible score had he known in advance all the symbols his friends would show.

INPUT

The first line contains the integer R ($1 \leq R \leq 50$), the number of rounds played.

The second line contains a string of R letters 'S', 'P' or 'R'. The string represents symbols that Sven showed in each round. 'S' is for scissors, 'P' for paper, 'R' for rock.

The third line contains the integer N ($1 \leq N \leq 50$), the number of friends.

Each of the following N lines contains a string of R letters 'S', 'P' or 'R'. These are the symbols shown by each of the N friends in each of the R rounds.

OUTPUT

Output Sven's actual score on the first line.

Output his largest possible score on the second line, assuming his friends didn't change their symbols.

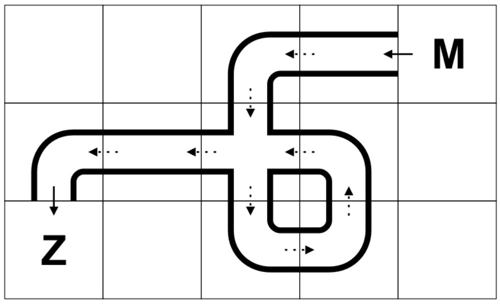
EXAMPLES

input	input	input
5 SSPPR 1 SSPPR	5 SSPPR 2 PPRRS RRSSP	4 SPRS 4 RPRP SRRR SSPR PSPS
output	**output**	**output**
5 10	10 15	12 21

To help design the new gas pipeline which will be used to deliver Russian gas to Croatia, Zagreb and Moscow are using the computer game Pipe Mania. In the game, Europe is divided into R rows and C columns. Each cell can be empty or contain one of the seven basic pipeline building blocks:

Block '|'   Block '-'   Block '+'   Block '1'   Block '2'   Block '3'   Block '4'

Gas flows from Moscow to Zagreb. Gas **can** flow in either direction through the building blocks. Block '+' is special in that gas **must** flow in two directions (one vertical, one horizontal), as in the following example:



Work on the new pipeline had already started when it was found that malicious hackers got hold of the plan and **erased exactly one building block** from the plan i.e. replaced it with an empty cell.

Write a program that determines where the block was erased from and what type it was.

INPUT

The first line contains two integers R and C, the dimensions of Europe (1 ≤ R, C ≤ 25).

The following R lines contain the plan, each consisting of exactly C characters. The characters are:

- Period ('.'), representing an empty cell;
- The characters '|' (ASCII 124), '-', '+', '1', '2', '3', '4', representing the building block types;
- The letters 'M' and 'Z', representing Moscow and Zagreb. Each of these will appear exactly once in the plan.

The flow of gas will be uniquely determined in the input; exactly one building block will be adjacent to each of Moscow and Zagreb. Additionally, the plan will **not** have redundant blocks i.e. **all** blocks in the plan must be used after the missing block is added.

The input will be such that a solution will exist and it will be unique.

OUTPUT

Output the row and column of the erased block, and the type of the block (one of the seven characters as in the input).

EXAMPLES

<table>
<tr>
<td>

input

```
3 7
.......
.M-.-Z.
.......
```

output

```
2 4 -
```

</td>
<td>

input

```
3 5
..1-M
1-+..
Z.23.
```

output

```
2 4 4
```

</td>
<td>

input

```
6 10
Z.1----4..
|.|....|..
|..14..M..
2-+++4....
..2323....
..........
```

output

```
3 3 |
```

</td>
</tr>
</table>

Ivo has an N×N table. The table has the integers 1 through N^2 inscribed in row-major order. The following operations can be done on the table:

1. Rotate a row – all cells in a single row are rotated right, so that the number in the last column moves to the first.
2. Rotate a column – all cells in a single column are rotated down, so that the number in the last row moves to the first.

Ivo occasionally feels the urge to move a number X to cell (R, C) and proceeds as follows:

- While X is not in column C, rotate the row it is in.
- While X is not in row R, rotate the column it is in.

Here is an example of how to move number 6 to cell (3, 4), start from the initial configuration:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4
8	**5**	**6**	**7**
9	10	11	12
13	14	15	16

1	2	3	4
7	**8**	**5**	**6**
9	10	11	12
13	14	15	16

1	2	3	**16**
7	8	5	**4**
9	10	11	**6**
13	14	15	**12**

Ivo wants to move K numbers one after another. Write a program that calculates the number of rotations needed.

INPUT

The first line contains two integers N (2 ≤ N ≤ 10000) and K (1 ≤ K ≤ 1000), the table dimension and the number of moves.

Each of the following K lines contains three integers X (1 ≤ X ≤ N^2), R and C (1 ≤ R, C ≤ N), the description of one move Ivo wants to make. Ivo does the moves in the order in which they are given.

OUTPUT

Output K lines; for each move, output the number of rotations needed.

EXAMPLES

input

```
4 1
6 3 4
```

output

```
3
```

input

```
4 2
6 3 4
6 2 2
```

output

```
3
5
```

input

```
5 3
1 2 2
2 2 2
12 5 5
```

output

```
2
5
3
```

Two groups of cavemen got into a land dispute and decided to settle it the old fashion-way, by throwing sticks at each other. The fight was organized in a cave, high enough that the ceiling is of no concern, but mineral deposits on the ground get in the way of flying sticks.

The cave can be divided into R rows and C columns, so that the entire cave consists of R×C cells. Each cell in the cave is either empty or contains a **chunk** of mineral. Two chunks of minerals are part of the same **cluster** if they are adjacent in one of the four main directions (up, down, left, right).

One group of cavemen is on the left side of the cave, the other on the right side. The groups alternate throwing sticks at the other group; first a group **chooses the height** at which the stick will fly and then (climbing on each others' shoulders as necessary) they throw it and the stick flies **horizontally** through the cave at the chosen height.

If the stick hits a chunk of mineral on the way, it destroys the chunk, the cell becomes empty and the stick stops its journey.

When a chunk is destroyed, it is possible that a cluster falls apart. If a newly created cluster would float in the air, then it **falls down** because of gravity. While falling, the cluster **does not change shape** i.e. all chunks in it fall together. As soon as some chunk in the falling cluster lands on a chunk from a different cluster or the ground, the entire cluster stops falling. Of course, if a cluster lands on another, they merge and become one.

Your program will be given the layout of minerals in the cave and the heights at which sticks were thrown. Determine the layout of minerals after the sticks are exchanged.

INPUT

The first line contains two integers R and C ($1 \le R, C \le 100$), the dimensions of the cave.

Each of the following R lines will contain C characters. The character '.' represents an empty cell, while the letter 'x' represents a chunk of mineral.

The next line contains an integer N ($1 \le N \le 100$), the number of sticks thrown.

The last line contains N integers separated by spaces, the heights at which sticks were thrown. All heights will be between 1 and R (inclusive), with height 1 being the **bottom** of the matrix and height R the top. The first stick is thrown left to right, the second right to left and so on.

No cluster will initially float in the air. Also, the input data will be such that at no point will two or more clusters fall simultaneously, so that there will be no ambiguous situations.

OUTPUT

The output should consist of R lines, each containing C characters, the final layout of the cave, in the same format as in the input.

EXAMPLES

input 5 6 ....xx.. ..x... ..xx.. .xxxx. 1 3 **output** ....xx.. ..xx.. .xxxx.	**input** 8 8 ....x.xx. ...xxx.. ..xxx... ..x.xxx. ..x...x. .xxx..x. 5 6 6 4 3 1 **output** ....x.. ..xxxx.. ..xxx.x. ..xxxxx.	**input** 7 6 .... xx.... .xx... ..xx.. ...xx. ....x. 2 6 4 **output** ....xx.. xx.xx. .x..x.

In the second example,

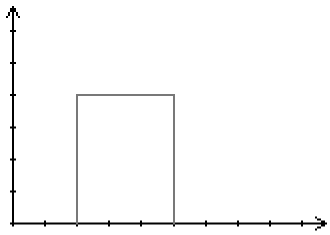
The first stick destroys the chunk in the fourth column at height 6, the second destroying the chunk in the seventh column of the same row.

The third stick destroys the chunk in the third column at height 4, after which the starting cluster splits in two, but both new clusters still lay on the ground.

The fourth stick destroys the chunk in the seventh column at height 3, splitting the right cluster into two. The largest cluster would float in the air and falls two cells down.

Finally, the fifth stick destroys a chunk in the second column at height 1.

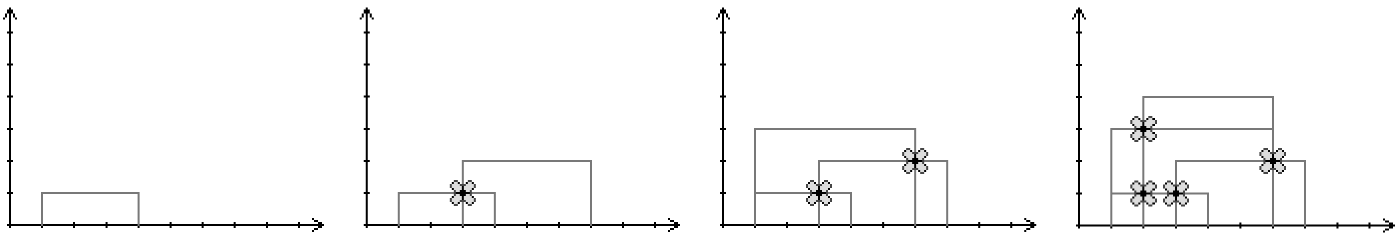
On a faraway planet, strange plants with two stems can be found. Every plant on the planet can be described by three numbers: the x-coordinates of the stems L and R, and the height H at which the stems are connect. The image depicts a plant with L=2, R=5 and H=4.



Every day a new plant grows on the planet. The plant that grows on day 1 is of height 1, and every subsequent plant is one higher than the previous one.

When a stem of a new plant **intersects** the **horizontal** segment of another plant, a small flower grows (if one wasn't there already). If segments merely touch in a point, a flower will not grow there.

The following images are a visualization of the first example on the next page.



Write a program that, given the coordinates of all plants, calculates the number of new flower every day.

INPUT

The first line contains an integer N ($1 \leq N \leq 100000$), the number of days.

Each of the following N lines contains two integers L and R ($1 \leq L < R \leq 100000$), the coordinates of the stems of a plant.

OUTPUT

Output N lines, the number of new flowers after each plant grows.

EXAMPLES

<table>
<tr>
<td>

input

```
4
1 4
3 7
1 6
2 6
```

output

```
0
1
1
2
```

</td>
<td>

input

```
5
1 3
3 5
3 9
2 4
3 8
```

output

```
0
0
0
3
2
```

</td>
</tr>
</table>