TASK	NOP	CIJEVI	ROBOTI	RELJEF
input	standard input			
output	standard output			
time limit (per test case)	1 second			
memory limit (per test case)	32 MB			
points	**35**	**45**	**55**	**65**
	200			

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.

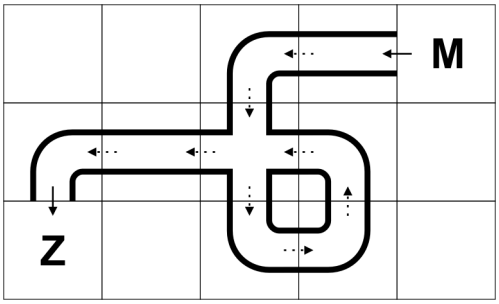
EXAMPLES

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

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 ($1 \leq R, C \leq 25$).

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

input	**input**	**input**
3 7 .... .M-.-Z. .... **output** 2 4 -	3 5 ..1-M 1-+.. Z.23. **output** 2 4 4	6 10 Z.1----4.. |.|....|.. |..14..M.. 2-+++4.... ..2323.... .... **output** 3 3 |

In the computer game Robots, the player tries to escape mad robots. There are many robots against the lone player, but their movement is predictable and the player can take advantage of that.

The game is played on a board of size R×C. The following 5 steps are repeated:

1. The player moves in one of eight directions (horizontally, vertically or diagonally) or stays in the same cell.
2. If the player moves to a cell occupied by a robot, the game ends and the player loses.
3. Every robot moves in one of eight directions, to the neighbouring cell closest to the player. More precisely, the robot attempts to minimize the value of $|r1-r2| + |s1-s2|$, where (r1, s1) and (r2, s2) are the positions of the player and robot.
4. If any robot has entered the player's cell, the game ends and the player loses.
5. If two or more robots have entered the same cell, a large explosion destroys all the robots in that cell.

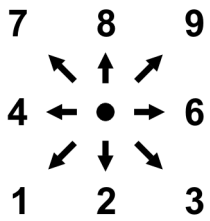
You will be given the starting location of the player, the locations of all robots and the moves the player makes. If the player makes all his moves and survives, you are to output the state of the board after all his moves. Otherwise, output how many moves he was able to make.

INPUT

The first line of input contains two integers R and C ($1 \leq R \leq 100$, $1 \leq S \leq 100$), the number of rows and columns on the board.

The following R lines contain C characters each, depicting the initial state on the board: the character '.' denotes an empty cell, 'R' a cell occupied by a robot, and 'I' the cell with the player.

The final line contains a string of at most 100 digits, the player's moves. Each move is a digit from 1 to 9, 5 meaning the player does not move, the rest according to this diagram:



The input sequence of moves will be such that the player will never attempt to move outside the board.

OUTPUT

If the player makes all the moves and survives, output the final state of the board in the same format as in the output. Otherwise, output `"kraj X"` (quotes for clarity), where X is the number of moves the player made.

EXAMPLES

input

```
4 5
I....
.....
.R.R.
.....
6
```

output

```
.I...
.RR..
.....
.....
```

input

```
9 10
.........
.........R
.........
R........
R...I.....
R........
.........
.........R
....R.....
5558888
```

output

```
....I.....
....R.....
.........
.........
.........
.........
.........
.........
.........
```

input

```
12 8
...I....
........
........
........
........
RR......
......RR
R......R
........
........
........
...R....
66445394444162
```

output

```
kraj 11
```

In the second example, the three robots on the left side of the map crash after the player opts to stand still in his first move. After the third move, the two robots from the right side of the map crash, while the robot from the bottom of the map pursues the player once he starts to move upwards.

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 ($1 \leq R, C \leq 100$), 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 ($1 \leq N \leq 100$), 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.