problem	kviz	pisoar	koze	nule
source file	kviz.pas kviz.c kviz.cpp	pisoar.pas pisoar.c pisoar.cpp	koze.pas koze.c koze.cpp	nule.pas nule.c nule.cpp
input data	stdin			
output data	stdout			
time limit (pentium III 850 mhz)	2 sec			
memory limit	32 MB			
points	30	40	60	70
	200			

In one very popular internet quiz, player has to give the answer to one very hard question. If player does not give the answer after some period of time, the quiz software will give him the first hint, after that the second hint, and in the end the third hint.

The only characters that appear in the answer are letters and the following characters: '**.**' (dot), '**,**' (comma), '**:**' (colon), '**;**' (semi-colon), '**!**' (exclamation mark), '**?**' (question mark), '**-**' (dash) and space (there will be no leading or trailing spaces however). By LETTERS, we mean all letters of English alphabet 'a'-'z' and 'A'-'Z', and by VOWELS we mean letters 'aeiouAEIOU'.

How do we generate hints?

- First hint is generated from the correct answer - we simply replace the letters with the character '**.**' (dot).
- Second hint is generated from the first hint - we reveal the first third of all the letters (if the number of letters is not divisible by 3, quotient should be rounded to the nearest integer).
- Third hint is generated from the second hint - we reveal all the remaining vowels. If there are no unrevealed vowels, then the third hint is generated from the first hint - we reveal first two-thirds of all the letters (if number of letters is not divisible by 3, quotient should be rounded to the nearest integer).

Write a program that will generate all three hints from the given answer.

Scoring: For each correct hint you will get one third of the number of points for that test.

input data

First and only line of input contains the correct answer from which we have to generate hints. The total number of characters will be at most 50.

output data

First line of output should contain the first hint, second line should contain the second hint, and the third line should contain the third hint.

examples

input	**input**	**input**
`Ante, Ante!`	`Vi ste? -Hur. Ben-Hur.`	`Upomoc! Lpv s nm pkrl sv smglsnk.`
output	**output**	**output**

`....., ....!`
`Ant., ....!`
`Ante, A..e!`
	`.. ...? -.... ...-....`
	`Vi ste? -.... ...-....`
	`Vi ste? -.u.. .e.-.u..`
		`.......! ... . .. .... .. ........`
		`Upomoc! Lp. . .. .... .. ........`
		`Upomoc! Lpv s nm pkrl s. ........`

We have a new model of a modern urinal equipped with an optical sensor and a "flush" function. We don't have software for it, so we have to write a program that will calculate **all** the moments when it is time for flushing. The rules are:

- sensors mark that urinal is being used if someone **is standing** in front of the urinal **for K or more consecutive seconds**,
- sensors mark that urinal use has completed when **nobody is standing** in front of the urinal **for L consecutive seconds**, starting from a point where the sensors marked that the urinal is being used (according to the first rule); **in that moment**, flushing is activated.

Before and after given time interval, we consider that **nobody is standing** in front of the urinal.

input data

First line of input contains three integers, K, L and N, $1 \leq K, L \leq 1000$, $1 \leq N \leq 10{,}000$.

Second line contains a sequence of N digits - zeros and ones. They are representing data received by sensors for a given time interval. Each digit tells us state of sensor in one second. Zero means that nobody was in front of the urinal during that second, and one means that somebody was in front of the urinal.

output data

For each flushing, write the corresponding activation time in seconds counting from the beginning of the interval. These numbers must be sorted in **ascending order**, and written in consecutive lines.

If the toilet is never flushed, then just output the word '**NIKAD**'.

examples

input	**input**	**input**
1 1 3 101	3 1000 3 111	3 2 18 011101001101110001
output	**output**	**output**
2 4	1003	8 16

There is certain number of goats in Mickey's backyard. While he was firmly sleeping, hungry wolves came in the yard and attacked the goats.

Yard is of **rectangular** shape and consists of fields arranged in rows and columns. Character '**.**' (dot) denotes an empty field, character '**#**' denotes a fence, character '**k**' denotes a goat and character '**v**' denotes a wolf.

We consider that **two fields are in the same region** if we can move from one field to the other by a path consisting of **horizontal and vertical moves only** and not containing any fences. Fields from which we can "escape" from the yard are not considered to be a part of any region.

Fortunately, our goats know karate – they can fight the wolves within a region and win (i.e. kill wolves) if their number in that region **is greater** than number of wolves in the same region. Otherwise, wolves eat all the goats inside that region.

In the beginning, all the goats and wolves are situated **inside** regions in the yard.

Write a program that will calculate the number of goats and the number of wolves still alive in the morning.

input data

First line of input contains two integers, R i C, $3 \leq R, C \leq 250$, the number of rows and the number of columns of Mickey's yard.

Each of the following R lines contains C characters. All of them together represent the layout of the yard i.e. positions of the fences, goats and wolves in the yard.

Note: 50% of given test data will be "simple" in the sense that the inside area of every region will be of rectangular shape, and within that rectangle there will be no other fences.

output data

First and only line of output should contain two numbers, number of goats and number of wolves still alive in the morning.

examples

input

```
6 6
...#..
.##v#.
#v.#.#
#.k#.#
.###.#
...###
```

output

```
0 2
```

input

```
8 8
.######.
#..k...#
#.####.#
#.#v.#.#
#.#.k#k#
#k.##..#
#.v..v.#
.######.
```

output

```
3 1
```

input

```
9 12
.###.#####..
#.kk#...#v#.
#..k#.#.#.#.
#..##k#...#.
#.#v#k###.#.
#..#v#....#.
#...v#v####.
.####.#vv.k#
.......####.
```

output

```
3 5
```

We are given a game board consisting of squares arranged in N rows and N columns, with each square containing a single **non-negative integer**

In the beginning of the game, a piece is situated on the **top-left** square (1,1) and it has to get to the **bottom-right** square (N,N) only moving **one-square-down** or **one-square-to-the-right** in each step. Furthermore, a piece **can not be placed** on a square containing number 0.

We define the **cost** of a path to be the product of all the numbers on that path.

We say that some path is **optimal** if **number of zeros** at the end of decimal representation of its cost is **minimal**.

Write a program that will calculate the number of zeros at the end of the decimal representation of cost of optimal path.

input data

First line of input contains the integer N, $1 \leq N \leq 1000$.

Each of the following N lines contains N numbers. These numbers represent the board from the problem statement. All the numbers will be non-negative integers less than or equal to 1,000,000.

Note: there will always be solution for each test data.

output data

First and only line of output should contain the number of zeros from the problem statement.

examples

input

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

output

```
0
```

input

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

output

```
1
```

input

```
4
1 3 0 0
0 8 2 25
6 5 0 3
0 15 7 4
```

output

```
2
```