TASK	FAKTOR	RIMSKI	KUTEVI	VUK	POSLOZI	PASIJANS

source code	`faktor.pas`
	`faktor.c`
	`faktor.cpp`
		`rimski.pas`
		`rimski.c`
		`rimski.cpp`
			`kutevi.pas`
			`kutevi.c`
			`kutevi.cpp`
				`vuk.pas`
				`vuk.c`
				`vuk.cpp`
					`poslozi.pas`
					`poslozi.c`
					`poslozi.cpp`
						`pasijans.pas`
						`pasijans.c`
						`pasijans.cpp`

input	standard input (*stdin*)					
output	standard output (*stdout*)					
time limit	1 s	1 s	1 s	1 s	1.5 s	6 s
memory limit	32 MB	32 MB	32 MB	32 MB	32 MB	128 MB
point value	**30**	**50**	**70**	**100**	**120**	**130**
	500					

2. round, 21. November 2009.

Author: Marko Ivanković, Goran Žužić

Impact factor of a scientific journal is a measure reflecting the average number of citations to articles published in science journals. For this task we are using a simplified formula for calculating the impact factor:

$$\frac{\text{Total sum of all citations articles published in the journal recived}}{\text{Total number of articles published}}$$

Rounding is always preformed up. For example the impact factor of the “Journal for ore research and time wasting” that published 38 articles quoted 894 times is 894 / 38 = 23.53 rounding up to 24.

You are the editor of one scientific journal. You know how much article you are going to publish and the owners are pushing you to reach a specific impact factor. You are wondering how many scientists you will have to bribe to cite your article to meet the owners demands. Since money is tight you want to bribe the **minimal** amount of scientists.

INPUT

First and only line of input will contain 2 integers, **A** (1 ≤ **A** ≤ 100), number of articles you plan to publish and **I** (1 ≤ **I** ≤ 100) impact factor the owners require.

OUTPUT

First and only line of output should contain one integer, the **minimal** number of scientists you need to bribe.

SAMPLE TESTS

input	**input**	**input**
38 24	1 100	10 10
output	**output**	**output**
875	100	91

Author: Filip Barl

Using roman numerals the numbers 1, 2, 3, 4, 5, 6, 7, 8, 9 are written as 'I', 'II', 'III', 'IV', 'V', 'VI', 'VII', 'VIII', 'IX'. Numbers 10, 20, 30, 40, 50, 60, 70, 80, 90 are written as 'X', 'XX', 'XXX', 'XL', 'L', 'LX', 'LXX', 'LXXX', 'XC'.

Any number smaller than 100 can be written by converting tens and ones separately and concatenating the results. So, for example, the number 48 would be written as XLVIII, XL for 40 and VIII for 8.

Given a number written in roman numerals, rearrange it's characters so that you create the smallest possible number, written in roman numerals.

INPUT

The first and only line of input contains one integer **B** (1 ≤ **B** < 100), written using roman numerals.

OUTPUT

The first and only line of output should contain a rearrangement of input characters so that it represents the smallest possible number, written in roman numerals.

SAMPLE TESTS

input
`VII`
output
`VII`
	input
	`VI`
	output
	`IV`
		input
		`III`
		output
		`III`

Author: Bruno Rahle

One day Mirko was cleaning up his room and found a straightedge and a compass. He went to the school the next day and challenged his friend Slavko to a geometric construction battle. Mirko knows how to construct some angles using the straightedge and compass and knows how to subtract and add any two angels he constructs. Slavko now shouts random angles and Mirko must draw them as fast as possible.
You are observing this battle and would like to know if Mirko can construct the angles Slavko shouts at all.

INPUT

The first line of input contains two integers, **N** (1 ≤ **N** ≤ 10), number of angles Mirko knows how to construct initially and **K** (1 ≤ **K** ≤ 10), number of angles Slavko selected.

The second line of input contains **N** integers, all smaller than 360, the angles Mirko knows how to construct initially.

The third line contains **K** integers, all smaller than 360, the angles Slavko selected.

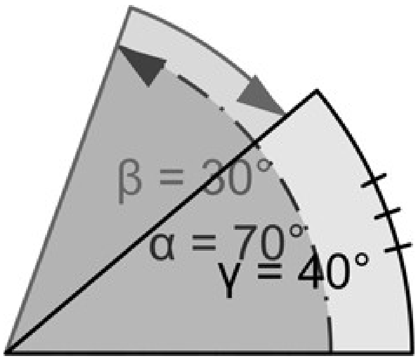
OUTPUT

Output consist of K lines, one for each angle Slavko selected. The i-th line should contain "YES" if Mirko can construct the i-th angle, and "NO" otherwise.

SAMPLE TESTS

input 2 1 30 70 40 **output** YES	**input** 1 1 100 60 **output** YES	**input** 3 2 10 20 30 5 70 **output** NO YES

First example description:



Subtracting 30° from 70° yields 70° - 30° = 40°.

Second example description:

Adding 100° 15 times yields 1500°, also known as 60°.

Vjekoslav the Wolf is running away from a bunch of blood hungry hunters. The hunters are smart and hide behind trees. Vjekoslav knows this, but doesn't know which trees. He would like to run to his comfortable, civilized cottage (as opposed to the hunters quite uncivilized den, yes I am rooting for the Wolf here) staying as far away as possible from any trees.

The forest can be represented as a N by M gird. Let us mark empty meadow patches with '.', patches with a tree in the middle with '+', Vjekoslav's current position with 'V' and the position of his cottage with 'J'. Vjekoslav can run from his current patch to any other patch north, east, south or west from him, **even if it contains a tree**.

If Vjekoslav is standing in R-th row and C-th column on the grid and there is a tree in the A-th row and B-th column then the distance between Vjekoslav and that tree is:

$$|R-A| + |C-B|$$

Help Vjekoslav find the best route to his cottage. The best route is any route that maximizes the minimal distance between Vjekoslav and all trees at any given moment.

Note that Vjekoslav's cottage doesn't occupy the entire patch so that patch must also be included in the route.

INPUT

The first line of input contains integers **N** and **M** (1 ≤ **N, M** ≤ 500), grid dimensions.

The next **N** lines contain **M** characters each: **'.'**, **'+'**, 'V', 'J'.

Input will contain exactly one character 'V' and 'J' and at least one character '+'.

OUTPUT

Output a single integer, the minimal distance from a tree in the optimal route.

SAMPLE TESTS

input
 	
`4 4`
`+...`
`....`
`....`
`V..J`
 	
output
 	
`3`
	input
	
	`4 5`
	`.....`
	`.+++.`
	`.+.+.`
	`V+.J+`
	
	output
	
	`0`

“Arrange” is a planetary popular Flash game. In “Arrange” the player is given a permutation of numbers 1 to N and a list of allowed swaps. He then has to perform a sequence of swaps that transforms the initial permutation back to the ordered sequence 1,2,3,4,5...N.
In order to break the high score list, you need to perform the minimal amount of swaps possible. You can't do that, but you can write a program that does it for you!

INPUT

The first line of input contains two integers, **N** (1 ≤ **N** ≤ 12), the length of the initial sequence and **M** (1 ≤ **M** ≤ **N***(**N** – 1) / 2) number of allowed swaps.

The second line of input contains a permutation of number 1 to **N**.

The next **M** lines contain descriptions of allowed swaps. If the line contains numbers A and B you are allowed to swap the A-th number with the B-th number. The input will never contain two identical swaps.

Note: the test data shall be such that the solution, not necessarily unique, will always exist.

OUTPUT

In the first line of input print the minimal number of swaps, X.
In the next X lines print the required swaps, in order. In each line print the index of the swap performed. Swaps are numbered increasingly as they appear in the input, starting from 1.

SAMPLE TESTS

input

```
2 1
2 1
1 2
```

output

```
1
1
```

input

```
3 2
2 1 3
1 3
2 3
```

output

```
3
2
1
2
```

input

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

output

```
0
```

Pasijans, patience, or solitaire is the name for a group of single player card games. One new such game, so new it has no name, is played with cards sporting random integers as values. The game starts by shuffling all cards and distributing them in **N** sequences, not necessarily of equal length.

During each turn, the player can remove the first card in any sequence and place it at the end of the “Solution sequence”. The card that was second in the selected sequence now becomes the first and the turn ends. Of course once the card is in the “Solution sequence” it cannot be removed, replaced or altered in any way. So don't even try.

The game ends when all cards are in the “Solution sequence”. The object of the game is to construct the best possible “Solution sequence”. One sequence is better than the other if for the first cards they differ, lets call them X and Y, the value on the card X is smaller than the value on the card Y.

Write a program that finds the best possible “Solution sequence”.

INPUT

The first line contains one integer **N** (1 ≤ **N** ≤ 1000), number of starting sequences.

Next **N** lines contain description of input sequences. Each line starts with an integer **L** (1 ≤ **L** ≤ 1000), length of the sequence. It's followed by L integers, smaller than 100.000.000.

OUTPUT

One line containing $\sum L$ numbers, the best possible “Solution sequence” obtainable.

SAMPLE TESTS

input 3 1 2 1 100 1 1	**input** 2 5 10 20 30 40 50 2 28 27	**input** 2 3 5 1 2 3 5 1 1
output 1 2 100	**output** 10 20 28 27 30 40 50	**output** 5 1 1 5 1 2