Problem	JEZIK	MRAVI	TOKI

Source file	`jezik.pas`
	`jezik.c`
	`jezik.cpp`
		`mravi.pas`
		`mravi.c`
		`mravi.cpp`
			`toki.pas`
			`toki.c`
			`toki.cpp`

Input file	`jezik.in`	`mravi.in`	`toki.in`
Output file	`jezik.out`	`mravi.out`	`toki.out`
Time limit per test	5 sekundi	5 sekundi	5 sekundi
Number of tests	10	10	10
Points per test	5	6	9
Total points	**50**	**60**	**90**
	200		

When analyzing source code in some programming language, it's sometimes useful to know if there are some functions that will never execute. If there are, that may lead to an error in the code.
In this occasion we observe simple programming language that consists of following functions:

RADI	program continues to next function
IDI a	program goes to a-th function
IDI a ILI b	program goes to a-th or b-th function

Program always starts at **first** function.
Write the program that will calculate **the number of functions that will never execute**.

Input data

In every line of input file there is one function, in i-th line there is i-th function.
In last line (after last function, i+1-th line) there is '**.**' (a spot). That line is not the part of a program and is not considered function.
Number of functions will be less or equal then 10,000.

Output data

In first and only line of output should be printed the number of unexecuted functions.

Examples

jezik.in

```
RADI
RADI
RADI
.
```

jezik.out

```
0
```

jezik.in

```
IDI 1
RADI
.
```

jezik.out

```
1
```

jezik.in

```
RADI
IDI 4 ILI 6
RADI
IDI 3
RADI
IDI 8
RADI
RADI
.
```

jezik.out

```
2
```

MRAVI

One of most valued jobs in ant population is traffic officer. The reason is **narrow tunnels** of ant hole in which ants who go in opposite directions can't **pass near each other**. Luckily, in the tunnel there are some **spots where ant can stand and let other ants pass**. Ants can pass each other at both **exits of the tunnel** without a problem.

Traffic officer knows the length of the tunnel as well as locations of every spots. Every morning he receives the list of arrival times of every ant to each tunnel exit. His mission is to organize traffic and enable all ants to pass the tunnel. That, of course, can be achieved in different (faster or slower) ways, but in that time, when last ant passes the tunnel, our friend ends his shift, so it's crucial for him to make them pass as quickly as possible.

Speed of every ant is **1 cm/s**. In one spot there can be **any number** of ants and we can **ignore** length of ants and width of spots as well.

Write the program that calculates **minimum time when all ants will pass the tunnel**.

Input data

In the first line of input file there are integers D and U, between which is single space,

$1 \leq D \leq 1{,}000{,}000$, $1 \leq U \leq 100{,}000$, $D > U$, D is the length of the tunnel in cm, and U is the number of spots. In next U lines there are locations of spots from left to right in increasing sequence.

In next line there is integer L, $1 \leq L \leq 100{,}000$, number of ants that come to left side of the tunnel. In next L lines there are arrival times of every ant to left side of the tunnel (the numbers are less or equal to 2,000,000).

In next line there is integer R, $1 \leq R \leq 100{,}000$, number of ants that come to right side of the tunnel. In next L lines there are arrival times of every ant to right side of the tunnel (the numbers are less or equal to 2,000,000).

Output data

In only line of output file you should print the number from the text of the problem.

Examples

mravi.in

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

mravi.out

```
8
```

mravi.in

```
10 1
3
1
0
1
2
```

mravi.out

```
16
```

mravi.in

```
10 2
4
6
2
0
4
1
0
```

mravi.out

```
14
```

Substring of some string A is defined as one of more (**not necessary succeeding**) elements of the string with maintaining the sequence.

There are given two strings, string VOKI and string TOKI.

Write the program that will **calculate any shortest substring** of string VOKI such as it **is not** substring of string TOKI.

Input data

In first line of input file there is string VOKI and in second one is string TOKI.

The only characters that will occur are lowercase characters of English alphabet (‘a’- ‘z’). String lengths will be less or equal to 2000.

Note: input will be such so there will always be a solution.

Output data

In the first line of file you should print the length of wanted substring, and in the second line you should print any valid substring from the text of the problem.

Examples

toki.in	**toki.in**	**toki.in**
ababaa abbaa	babab babba	banana anbnaanbaan
toki.out	**toki.out**	**toki.out**
3 bab	3 aab	5 banna