Problem	NUMBERS	BLAST	TWO

Source file	`numbers.pas`
	`numbers.c`
	`numbers.cpp`
		`blast.pas`
		`blast.c`
		`blast.cpp`
			`two.pas`
			`two.c`
			`two.cpp`

Input file	`numbers.in`	`blast.in`	`two.in`
Output file	`numbers.out`	`blast.out`	`two.out`
Time limit per test	10 seconds	10 seconds	10 seconds
Number of tests	10	10	10
Points per test	5	7	8
Total points	**50**	**70**	**80**
	200		

NUMBERS

Let A be a sequence of numbers from 1 to N in **random** order (every number from 1 to N appears **exactly once**).

We define the sequence B:

B[k] = 1 ... if and only if the first **k** numbers of the sequence A constist only of numbers **1** to **k** in **random** order (every number from **1**...**k** appears **exactly once**),

0 ... otherwise.

We know the sequence B and **some** of the elements of the sequence A.

You are to write a program the will reconstruct the **whole** string A.

Input data

First line of the input file consists of integers N and M, $1 \leq N \leq 100000$, $0 \leq M \leq N$.
Second line of the input file consists of the elements of the sequence B.
Next M lines consist of the data about the known elements of sequence A, two integers X and Y. That means that the Xth element of sequence A is Y (A[X]=Y). This information will not be **contradictory**.

Output data

The first and only line of the output file should consist of the elements of the sequence A; the numbers should be seperated by the space character.

If there is no sollution, your program should write 'not exist' in the output file.

Note: the sollution doesn't have to be unique.

Examples

numbers.in

```
5 1
0 0 1 0 1
2 3
```

numbers.out

```
2 3 1 5 4
```

numbers.in

```
7 2
0 0 0 1 0 0 1
1 2
5 6
```

numbers.out

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

numbers.in

```
8 3
0 0 0 1 0 0 1 1
1 2
5 6
2 7
```

numbers.out

```
ne postoji
```

There are given two strings, A and B.

An expansion of some string X is a string created by adding or inserting any number (zero, one or more) of blanks **anywhere** in the string, or in the beggining or the end of the string. Eg., if the string X is 'abcbcd', then the strings `'abcb-cd'`, `'-a-bcbcd-'` and `'abcb-cd-'` are expansions of the string X (blanks are denoted by the character '-').

If A_1 is an expansion of the string A, and B_1 is and expansion of the string B, and if A_1 and B_1 are of the same length, then we define the distance of the strings A_1 and B_1 as **the sum of the distances** of the characters on the same positions in these strings. We define the distance of two characters as the **absolute difference** of their ASCII codes, except the distance of the blank and another character, which is **given** (and **equal** for all characters).

You are to write a program which finds the expansions A_1 and B_1 of strings A and B, that have the **smallest** difference.

Input data

The first line of the input file consists of the string A, and the second line of string B. They are consisted only of lower case characters of the english alphabet (a-z), and the number of characters in any of the strings is less than or equal to 2000.

The third line consists of an integer K, $1 \leq K \leq 100$, the distance of the blank and the other characters.

Output data

The first on only line of the input file should consist of the smallest distance as defined in the text of the task.

Examples

blast.in	**blast.in**	**blast.in**

`cmc`
`snmn`
`2`
	`koiv`
	`ua`
	`1`
		`mj`
		`jao`
		`4`

blast.out	**blast.out**	**blast.out**
`10`	`5`	`12`

The city consists of intersections and streets that connect them.

Heavy snow covered the city so the mayor Milan gave to the winter-service a list of streets that have to be cleaned of snow. These streets are chosen such that the number of streets is **as small as possible** but still every two intersections to be **connected** i.e. between every two intersections there will be **exactly one** path. The winter service consists of **two** snow plovers and two drivers, Mirko and Slavko, and their starting position is on one of the intersections.

The snow plover burns **one liter of fuel** per meter (even if it is driving through a street that has already been cleared of snow) and it has to clean **all** streets from the list in such order so the total fuel spent is **minimal**. When all the streets are cleared of snow, the snow plovers are parked on the last intersection they visited. Mirko and Slavko don't have to finish their plowing on the same intersection.

Write a program that calculates the total amount of fuel that the snow plovers will spend.

Input data

The first line of the input file contains two integers: N and S, $1 \leq N \leq 100000$, $1 \leq S \leq N$. N is the total number of intersections; S is ordinal number of the snow plovers starting intersection. Intersections are marked with numbers 1..N.

Each of the next N-1 lines contains three integers: A, B and C, meaning that intersections A and B are directly connected by a street and that street's length is C meters, $1 \leq C \leq 1000$.

Output data

Write to the output file the minimal amount of fuel needed to clean all streets.

Examples

two.in

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

two.out

```
6
```

two.in

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

two.out

```
5
```

two.in

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

two.out

```
11
```