Codeforces Round #293 (Div. 2)

A. Vitaly and Strings

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Vitaly is a diligent student who never missed a lesson in his five years of studying in the university. He always does his homework on time and passes his exams in time.

During the last lesson the teacher has provided two strings s and t to Vitaly. The strings have the same length, they consist of lowercase English letters, string s is lexicographically smaller than string t. Vitaly wondered if there is such string that is lexicographically larger than string s and at the same is lexicographically smaller than string t. This string should also consist of lowercase English letters and have the length equal to the lengths of strings s and t.

Let's help Vitaly solve this easy problem!

Input

The first line contains string s ($1 \le |s| \le 100$), consisting of lowercase English letters. Here, $|s|$ denotes the length of the string.

The second line contains string t ($|t| = |s|$), consisting of lowercase English letters.

It is guaranteed that the lengths of strings s and t are the same and string s is lexicographically less than string t.

Output

If the string that meets the given requirements doesn't exist, print a single string "`No such string`" (without the quotes).

If such string exists, print it. If there are multiple valid strings, you may print any of them.

Sample test(s)

input
```
a
c
```
output
```
b
```

input
```
aaa
zzz
```
output
```
kkk
```

input
```
abcdefg
abcdefh
```
output
```
No such string
```

Note

String $s = s_1s_2... s_n$ is said to be lexicographically smaller than $t = t_1t_2... t_n$, if there exists such i, that $s_1 = t_1, s_2 = t_2, ... s_{i-1} = t_{i-1}, s_i < t_i$.

B. Tanya and Postcard

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Little Tanya decided to present her dad a postcard on his Birthday. She has already created a message — string s of length n, consisting of uppercase and lowercase English letters. Tanya can't write yet, so she found a newspaper and decided to cut out the letters and glue them into the postcard to achieve string s. The newspaper contains string t, consisting of uppercase and lowercase English letters. We know that the length of string t greater or equal to the length of the string s.

The newspaper may possibly have too few of some letters needed to make the text and too many of some other letters. That's why Tanya wants to cut some n letters out of the newspaper and make a message of length exactly n, so that it looked as much as possible like s. If the letter in some position has correct value and correct letter case (in the string s and in the string that Tanya will make), then she shouts joyfully "`YAY!`", and if the letter in the given position has only the correct value but it is in the wrong case, then the girl says "`WHOOPS`".

Tanya wants to make such message that lets her shout "`YAY!`" as much as possible. If there are multiple ways to do this, then her second priority is to maximize the number of times she says "`WHOOPS`". Your task is to help Tanya make the message.

Input

The first line contains line s ($1 \le |s| \le 2 \cdot 10^5$), consisting of uppercase and lowercase English letters — the text of Tanya's message.

The second line contains line t ($|s| \le |t| \le 2 \cdot 10^5$), consisting of uppercase and lowercase English letters — the text written in the newspaper.

Here $|a|$ means the length of the string a.

Output

Print two integers separated by a space:

- the first number is the number of times Tanya shouts "`YAY!`" while making the message,
- the second number is the number of times Tanya says "`WHOOPS`" while making the message.

Sample test(s)

input

```
AbC
DCbA
```

output

```
3 0
```

input

```
ABC
abc
```

output

```
0 3
```

input

```
abacaba
AbaCaBA
```

output

```
3 4
```

C. Anya and Smartphone

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Anya has bought a new smartphone that uses *Berdroid* operating system. The smartphone menu has exactly n applications, each application has its own icon. The icons are located on different screens, one screen contains k icons. The icons from the first to the k-th one are located on the first screen, from the $(k + 1)$-th to the $2k$-th ones are on the second screen and so on (the last screen may be partially empty).

Initially the smartphone menu is showing the screen number 1. To launch the application with the icon located on the screen t, Anya needs to make the following gestures: first she scrolls to the required screen number t, by making $t - 1$ gestures (if the icon is on the screen t), and then make another gesture — press the icon of the required application exactly once to launch it.

After the application is launched, the menu returns to the first screen. That is, to launch the next application you need to scroll through the menu again starting from the screen number 1.

All applications are numbered from 1 to n. We know a certain order in which the icons of the applications are located in the menu at the beginning, but it changes as long as you use the operating system. *Berdroid* is intelligent system, so it changes the order of the icons by moving the more frequently used icons to the beginning of the list. Formally, right after an application is launched, Berdroid swaps the application icon and the icon of a preceding application (that is, the icon of an application on the position that is smaller by one in the order of menu). The preceding icon may possibly be located on the adjacent screen. The only exception is when the icon of the launched application already occupies the first place, in this case the icon arrangement doesn't change.

Anya has planned the order in which she will launch applications. How many gestures should Anya make to launch the applications in the planned order?

Note that one application may be launched multiple times.

Input

The first line of the input contains three numbers n, m, k $(1 \le n, m, k \le 10^5)$ — the number of applications that Anya has on her smartphone, the number of applications that will be launched and the number of icons that are located on the same screen.

The next line contains n integers, permutation $a_1, a_2, ..., a_n$ — the initial order of icons from left to right in the menu (from the first to the last one), a_i — is the id of the application, whose icon goes i-th in the menu. Each integer from 1 to n occurs exactly once among a_i.

The third line contains m integers $b_1, b_2, ..., b_m (1 \le b_i \le n)$ — the ids of the launched applications in the planned order. One application may be launched multiple times.

Output

Print a single number — the number of gestures that Anya needs to make to launch all the applications in the desired order.

Sample test(s)

input

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

output

```
7
```

input

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

output

```
8
```

Note

In the first test the initial configuration looks like $(123)(456)(78)$, that is, the first screen contains icons of applications $1, 2, 3$, the second screen contains icons $4, 5, 6$, the third screen contains icons $7, 8$.

After application 7 is launched, we get the new arrangement of the icons — $(123)(457)(68)$. To launch it Anya makes 3 gestures.

After application 8 is launched, we get configuration $(123)(457)(86)$. To launch it Anya makes 3 gestures.

After application 1 is launched, the arrangement of icons in the menu doesn't change. To launch it Anya makes 1 gesture.

In total, Anya makes 7 gestures.

D. Ilya and Escalator

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Ilya got tired of sports programming, left university and got a job in the subway. He was given the task to determine the escalator load factor.

Let's assume that n people stand in the queue for the escalator. At each second one of the two following possibilities takes place: either the first person in the queue enters the escalator with probability p, or the first person in the queue doesn't move with probability $(1 - p)$, paralyzed by his fear of escalators and making the whole queue wait behind him.

Formally speaking, the i-th person in the queue cannot enter the escalator until people with indices from 1 to $i - 1$ inclusive enter it. In one second only one person can enter the escalator. The escalator is infinite, so if a person enters it, he never leaves it, that is he will be standing on the escalator at any following second. Ilya needs to count the expected value of the number of people standing on the escalator after t seconds.

Your task is to help him solve this complicated task.

Input

The first line of the input contains three numbers n, p, t $(1 \leq n, t \leq 2000,\ 0 \leq p \leq 1)$. Numbers n and t are integers, number p is real, given with exactly two digits after the decimal point.

Output

Print a single real number — the expected number of people who will be standing on the escalator after t seconds. The absolute or relative error mustn't exceed 10^{-6}.

Sample test(s)

input

```
1 0.50 1
```

output

```
0.5
```

input

```
1 0.50 4
```

output

```
0.9375
```

input

```
4 0.20 2
```

output

```
0.4
```

E. Arthur and Questions

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

After bracket sequences Arthur took up number theory. He has got a new favorite sequence of length n $(a_1, a_2, ..., a_n)$, consisting of integers and integer k, not exceeding n.

This sequence had the following property: if you write out the sums of all its segments consisting of k consecutive elements $(a_1 + a_2 ... + a_k, a_2 + a_3 + ... + a_{k+1}, ..., a_{n-k+1} + a_{n-k+2} + ... + a_n)$, then those numbers will form strictly increasing sequence.

For example, for the following sample: $n = 5, k = 3, a = (1, 2, 4, 5, 6)$ the sequence of numbers will look as follows: $(1 + 2 + 4, 2 + 4 + 5, 4 + 5 + 6) = (7, 11, 15)$, that means that sequence a meets the described property.

Obviously the sequence of sums will have $n - k + 1$ elements.

Somebody (we won't say who) replaced some numbers in Arthur's sequence by question marks (if this number is replaced, it is replaced by exactly one question mark). We need to restore the sequence so that it meets the required property and also minimize the sum $|a_i|$, where $|a_i|$ is the absolute value of a_i.

Input

The first line contains two integers n and k $(1 \le k \le n \le 10^5)$, showing how many numbers are in Arthur's sequence and the lengths of segments respectively.

The next line contains n space-separated elements a_i $(1 \le i \le n)$.

If $a_i = ?$, then the i-th element of Arthur's sequence was replaced by a question mark.

Otherwise, a_i $(-10^9 \le a_i \le 10^9)$ is the i-th element of Arthur's sequence.

Output

If Arthur is wrong at some point and there is no sequence that could fit the given information, print a single string "`Incorrect sequence`" (without the quotes).

Otherwise, print n integers — Arthur's favorite sequence. If there are multiple such sequences, print the sequence with the minimum sum $|a_i|$, where $|a_i|$ is the absolute value of a_i. If there are still several such sequences, you are allowed to print any of them. Print the elements of the sequence without leading zeroes.

Sample test(s)

input

```
3 2
? 1 2
```

output

```
0 1 2
```

input

```
5 1
-10 -9 ? -7 -6
```

output

```
-10 -9 -8 -7 -6
```

input

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

output

```
Incorrect sequence
```

F. Pasha and Pipe

time limit per test: 4 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

On a certain meeting of a ruling party "A" minister Pavel suggested to improve the sewer system and to create a new pipe in the city.

The city is an $n \times m$ rectangular squared field. Each square of the field is either empty (then the pipe can go in it), or occupied (the pipe cannot go in such square). Empty squares are denoted by character '.', occupied squares are denoted by character '#'.

The pipe must meet the following criteria:

- the pipe is a polyline of width 1,
- the pipe goes in empty squares,
- the pipe starts from the edge of the field, but not from a corner square,
- the pipe ends at the edge of the field but not in a corner square,
- the pipe has at most 2 turns $(90$ degrees),
- the border squares of the field must share **exactly two** squares with the pipe,
- if the pipe looks like a single segment, then the end points of the pipe must lie on distinct edges of the field,
- for each non-border square of the pipe there are **exacly two** side-adjacent squares that also belong to the pipe,
- for each border square of the pipe there is **exactly one** side-adjacent cell that also belongs to the pipe.

Here are some samples of **allowed** piping routes:

```
....#    ....#    .*..#
*****    ****.    .***.
..#..    ..#*.    ..#*.
#...#    #..*#    #..*#
.....    ...*.    ...*.
```

Here are some samples of **forbidden** piping routes:

```
.**.#    *...#    .*.*#
.....    ****.    .*.*.
..#..    ..#*.    .*#*.
#...#    #..*#    #*.*#
.....    ...*.    .***.
```

In these samples the pipes are represented by characters ' $*$ '.

You were asked to write a program that calculates the number of distinct ways to make exactly one pipe in the city.

The two ways to make a pipe are considered distinct if they are distinct in at least one square.

Input

The first line of the input contains two integers n, m $(2 \le n, m \le 2000)$ — the height and width of Berland map.

Each of the next n lines contains m characters — the map of the city.

If the square of the map is marked by character '.', then the square is empty and the pipe can through it.

If the square of the map is marked by character '#', then the square is full and the pipe can't through it.

Output

In the first line of the output print a single integer — the number of distinct ways to create a pipe.

Sample test(s)

input

```
3 3
...
..#
...
```

output

```
3
```

input

```
4 2
..
..
..
```

```
..
```

output

```
2
```

input

```
4 5
#...#
#...#
###.#
###.#
```

output

```
4
```

Note

In the first sample there are 3 ways to make a pipe (the squares of the pipe are marked by characters ' * '):

```
.*.        .*.        ...
.*#        **#        **#
.*.        ...        .*.
```