Sponsored by Telegram

Codeforces Round #556 (Div. 2)

A. Stock Arbitraging

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

Welcome to Codeforces Stock Exchange! We're pretty limited now as we currently allow trading on one stock, Codeforces Ltd. We hope you'll still be able to make profit from the market!

In the morning, there are n opportunities to buy shares. The i-th of them allows to buy as many shares as you want, each at the price of s_i bourles.

In the evening, there are m opportunities to sell shares. The i-th of them allows to sell as many shares as you want, each at the price of b_i bourles. You can't sell more shares than you have.

It's morning now and you possess r bourles and no shares.

What is the maximum number of bourles you can hold after the evening?

Input

The first line of the input contains three integers n, m, r ($1 \leq n \leq 30$, $1 \leq m \leq 30$, $1 \leq r \leq 1000$) — the number of ways to buy the shares on the market, the number of ways to sell the shares on the market, and the number of bourles you hold now.

The next line contains n integers $s_1, s_2, \dots, s_n$ ($1 \leq s_i \leq 1000$); s_i indicates the opportunity to buy shares at the price of s_i bourles.

The following line contains m integers $b_1, b_2, \dots, b_m$ ($1 \leq b_i \leq 1000$); b_i indicates the opportunity to sell shares at the price of b_i bourles.

Output

Output a single integer — the maximum number of bourles you can hold after the evening.

Examples

input

```
3 4 11
4 2 5
4 4 5 4
```

output

```
26
```

input

```
2 2 50
5 7
4 2
```

output

```
50
```

Note

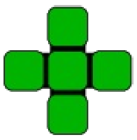
In the first example test, you have 11 bourles in the morning. It's optimal to buy 5 shares of a stock at the price of 2 bourles in the morning, and then to sell all of them at the price of 5 bourles in the evening. It's easy to verify that you'll have 26 bourles after the evening.

In the second example test, it's optimal not to take any action.

B. Tiling Challenge

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

One day Alice was cleaning up her basement when she noticed something very curious: an **infinite** set of wooden pieces! Each piece was made of five square tiles, with four tiles adjacent to the fifth center tile:



By the pieces lay a large square wooden board. The board is divided into n^2 cells arranged into n rows and n columns. Some of the cells are already occupied by single tiles stuck to it. The remaining cells are free.
Alice started wondering whether she could fill the board completely using the pieces she had found. Of course, each piece has to cover exactly five distinct cells of the board, no two pieces can overlap and every piece should fit in the board entirely, without some parts laying outside the board borders. The board however was too large for Alice to do the tiling by hand. Can you help determine if it's possible to fully tile the board?

Input

The first line of the input contains a single integer n ($3 \leq n \leq 50$) — the size of the board.

The following n lines describe the board. The i-th line ($1 \leq i \leq n$) contains a single string of length n. Its j-th character ($1 \leq j \leq n$) is equal to "." if the cell in the i-th row and the j-th column is free; it is equal to "#" if it's occupied.

You can assume that the board contains at least one free cell.

Output

Output YES if the board can be tiled by Alice's pieces, or NO otherwise. You can print each letter in any case (upper or lower).

Examples

input

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

output

```
YES
```

input

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

output

```
NO
```

input

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

output

```
YES
```

input

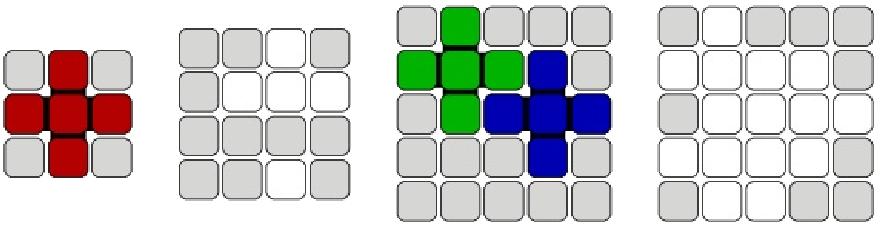
```
5
#.###
....#
#....
....#
#..##
```

output

```
NO
```

Note

The following sketches show the example boards and their tilings if such tilings exist:



C. Prefix Sum Primes

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

We're giving away nice huge bags containing number tiles! A bag we want to present to you contains n tiles. Each of them has a single number written on it — either 1 or 2.

However, there is one condition you must fulfill in order to receive the prize. You will need to put all the tiles from the bag in a sequence, in any order you wish. We will then compute the sums of all prefixes in the sequence, and then count how many of these sums are prime numbers. If you want to keep the prize, you will need to maximize the number of primes you get.

Can you win the prize? Hurry up, the bags are waiting!

Input

The first line of the input contains a single integer n $(1 \le n \le 200\,000)$ — the number of number tiles in the bag. The following line contains n space-separated integers $a_1, a_2, \dots, a_n$ ($a_i \in \{1, 2\}$) — the values written on the tiles.

Output

Output a permutation $b_1, b_2, \dots, b_n$ of the input sequence $(a_1, a_2, \dots, a_n)$ maximizing the number of the prefix sums being prime numbers. If there are multiple optimal permutations, output any.

Examples

input

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

output

```
1 1 1 2 2
```

input

```
9
1 1 2 1 1 1 2 1 1
```

output

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

Note

The first solution produces the prefix sums $1, \mathbf{2}, \mathbf{3}, \mathbf{5}, \mathbf{7}$ (four primes constructed), while the prefix sums in the second solution are $1, \mathbf{2}, \mathbf{3}, \mathbf{5}, 6, \mathbf{7}, 8, 10, \mathbf{11}$ (five primes). Primes are marked bold and blue. In each of these cases, the number of produced primes is maximum possible.

D. Three Religions

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

During the archaeological research in the Middle East you found the traces of three ancient religions: First religion, Second religion and Third religion. You compiled the information on the evolution of each of these beliefs, and you now wonder if the followers of each religion could coexist in peace.

The Word of Universe is a long word containing the lowercase English characters only. At each moment of time, each of the religion beliefs could be described by a word consisting of lowercase English characters.

The three religions can coexist in peace if their descriptions form disjoint subsequences of the Word of Universe. More formally, one can paint some of the characters of the Word of Universe in three colors: 1, 2, 3, so that each character is painted in at most one color, and the description of the i-th religion can be constructed from the Word of Universe by removing all characters that aren't painted in color i.

The religions however evolve. In the beginning, each religion description is empty. Every once in a while, either a character is appended to the end of the description of a single religion, or the last character is dropped from the description. After each change, determine if the religions could coexist in peace.

Input

The first line of the input contains two integers n, q $(1 \le n \le 100\,000$, $1 \le q \le 1000)$ — the length of the Word of Universe and the number of religion evolutions, respectively. The following line contains the Word of Universe — a string of length n consisting of lowercase English characters.

Each of the following line describes a single evolution and is in one of the following formats:

- + i c ($i \in \{1, 2, 3\}$, $c \in \{\mathtt{a}, \mathtt{b}, \dots, \mathtt{z}\}$: append the character c to the end of i-th religion description.
- - i ($i \in \{1, 2, 3\}$) – remove the last character from the i-th religion description. You can assume that the pattern is non-empty.

You can assume that no religion will have description longer than 250 characters.

Output

Write q lines. The i-th of them should be YES if the religions could coexist in peace after the i-th evolution, or NO otherwise.

You can print each character in any case (either upper or lower).

Examples

input

```
6 8
abdabc
+ 1 a
+ 1 d
+ 2 b
+ 2 c
+ 3 a
+ 3 b
+ 1 c
- 2
```

output

```
YES
YES
YES
YES
YES
YES
NO
YES
```

input

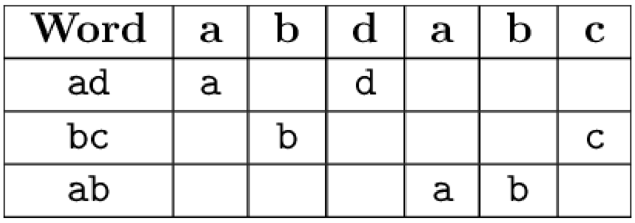
```
6 8
abbaab
+ 1 a
+ 2 a
+ 3 a
+ 1 b
+ 2 b
+ 3 b
- 1
+ 2 z
```

output

```
YES
YES
YES
YES
YES
NO
YES
NO
```

Note

In the first example, after the 6th evolution the religion descriptions are: ad, bc, and ab. The following figure shows how these descriptions form three disjoint subsequences of the Word of Universe:



Word	a	b	d	a	b	c
ad	a		d			
bc		b				c
ab				a	b	

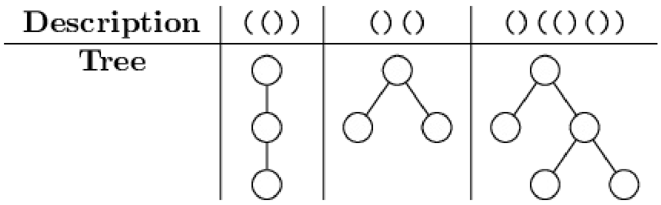
E. Tree Generator™

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

Owl Pacino has always been into trees — unweighted rooted trees in particular. He loves determining the *diameter* of every tree he sees — that is, the maximum length of any simple path in the tree.

Owl Pacino's owl friends decided to present him the Tree Generator™ — a powerful machine creating rooted trees from their *descriptions*. An n-vertex rooted tree can be *described* by a bracket sequence of length $2(n-1)$ in the following way: find any walk starting and finishing in the root that traverses each edge exactly twice — once down the tree, and later up the tree. Then follow the path and write down "(" (an opening parenthesis) if an edge is followed down the tree, and ")" (a closing parenthesis) otherwise.

The following figure shows sample rooted trees and their descriptions:



Owl wrote down the description of an n-vertex rooted tree. Then, he rewrote the description q times. However, each time he wrote a new description, he picked two different characters in the description he wrote the last time, swapped them and wrote down the resulting string. He always made sure that each written string was the description of a rooted tree.

Pacino then used Tree Generator™ for each description he wrote down. What is the diameter of each constructed tree?

Input

The first line of the input contains two integers n, q ($3 \le n \le 100\,000$, $1 \le q \le 100\,000$) — the number of vertices in the tree and the number of changes to the tree description. The following line contains a description of the initial tree — a string of length $2(n-1)$ consisting of opening and closing parentheses.

Each of the following q lines describes a single change to the description and contains two space-separated integers a_i, b_i ($2 \le a_i, b_i \le 2n-3$) which identify the indices of two brackets to be swapped. You can assume that the description will change after each query, and that after each change a tree can be constructed from the description.

Output

Output $q+1$ integers — the diameter of each constructed tree, in the order their descriptions have been written down.

Examples

input

```
5 5
(((())))
4 5
3 4
5 6
3 6
2 5
```

output

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

input

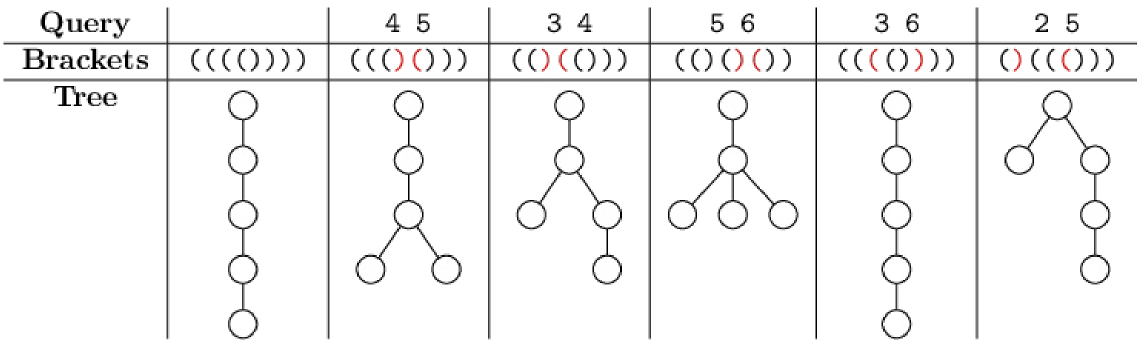
```
6 4
(((())()))
6 7
5 4
6 4
7 4
```

output

```
4
4
4
5
3
```

Note

The following figure shows each constructed tree and its description in the first example test:


Query
4 5
3 4
5 6
3 6
2 5
Brackets
(((())))
((()()))
(()(()))
(()()())
(((())))
()((()))
Tree