Train

This is a figure that shows the structure of a station for train dispatching.

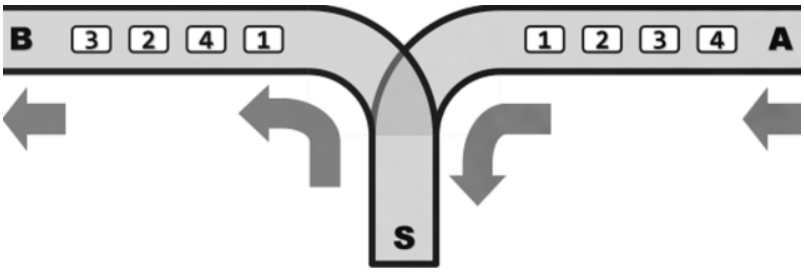

Figure 1: train station diagram for Example 1

In this station, A is the entrance for each train and B is the exit. S is the switching track. The coaches of a train can enter the switching track from direction A and must leave in direction B. Individual coaches can be disconnected from the rest of the train as they enter the switching track, so that they can be reorganized before they continue in direction B. If a coach enters the switching track from direction A, it must leave in direction B (i.e., it cannot return towards A). If a coach leaves in direction B, it cannot return back to the switching track.

Assume that a train consist of n coaches labeled {1, 2, . . . , n}. A dispatcher wants to know whether these coaches can pull out at B in the order of {a1, a2, . . . , an}.

Input

The 1st line contains an integer n (n <= 1,000) equal to the number of coaches, as described above. In each of the next lines of the input, except the last one, there is a permutation of 1, 2, . . . , n, this is the sequence {a1, a2, . . . , an} that the dispatcher would like to achieve as the coaches leave the switching track in direction B. The last line of the block contains just '0' (to indicate the end of input).

Output

You should output the result for each permutation. If the sequence is feasible, output a "Yes", followed by a newline. If the sequence is infeasible, output a "No", followed by a newline.

Example 1

```
Input:
4
3 2 4 1
0

Output:
Yes
```

This is done following steps: - coaches 1, 2, and 3 enter the switching track from track A - coaches 3 and 2 leave the switching track in direction leading to track B - the coach 4 enters the switching track - coaches 4 and 1 leave the switching track to track B

Example 2

```
Input:
5
1 2 3 4 5
0

Output:
Yes
```

Each coach enters the switching track and immediately leaves it before any other coach arrives in the switching track.

Example 3

```
Input:
5
4 5 1 3 2
0

Output:
No
```

For the coach 4 to be the first one to leave the switching track, the coaches 1, 2, and 3 have to enter the switching track and remain there so the coach 4 can enter and then leave. But this means that the coach 1 can no longer leave the switching track before coaches 3 and 2.

Buggy Keyboard

Gabrielle has an assignment for her Algorithmic Problem Solving course due in a couple of hours. She opened up her computer which she purchased recently and started working using her favorite editor when she discovered something strange. Generally, pressing the backspace key should erase a character to the left of the cursor. But on her new computer, pressing that key produced a character `<`. Since the assignment is due in couple of hours, she does not have time to call customer support or replace the computer so she decides to temporarily find a way around the problem with the help of a program.

Help Gabrielle write a program which takes the string written on her computer and outputs the string that Gabrielle actually intended to write. It can be assumed that she never needs to output the character `<`. Also you can be assured that she will never press backspace on an empty line.

Figure 1: Computer keyboard

Input

A string `s` containing text written on Gabrielle's computer. The length of `s` is less than 10^6 The string will contain only lower case letters, spaces, and the character `<`.

Output

A string containing text that was actually intended. Note that there should be no spaces or new line after the last character in the output

Example 1

```
Input:
a<bcd<

Output:
bc
```

Example 2

```
Input:
prind<tf

Output:
printf
```

Example 3

```
Input:
as<d<<

Output:
```

Note: there is no output in the last example.

Encryption challenge

The Department of Defense is developing a new encryption method that is used for transporting the country's secret military codes. The codes themselves have been *encrypted* by new award winning encryption methodology. A separate decoding key is used to *decrypt* the sequence and return it back to the original. Your task is to write a program that given an encrypted sequence along with the decoding key, recovers the original sequence.

The decoding key contains two different characters encoding two different operations: `R` and `D`.

- the character `R` reverses the numerical sequence;
- the character `D` drops the first element of the sequence.

There are a few cases where the original data is irrecoverable due to transmission errors. Your program should be able to detect that and print `error`, instead of the recovered sequence. The errors should be reported whenever there is an instruction that attempts to drop the first element of an empty sequence. (Note, that reversing an empty sequence should not produce any problems.)

Input

The first line contains the decoding key k(1 <= length(k) <= 10^5) consisting of characters 'R' and 'D'.

The next line contains the number of values in the encrypted sequence, n (0 <= n <= 10^5).

The last line contains the encrypted sequence itself $[y_1, ..., y_n]$, where ($1 \leq y_i \leq 100$). The values are enclosed in a set of square brackets (i.e., start with '[' and end with ']'). The individual values are separated by commas and nothing else.

Output

A single line containing either the decrypted sequence or "error". The resulting list should start with '[' and end with ']'. The integers in the input and output list are separated by commas and nothing else.

Example 1

```
Input:
DD
1
[42]

Output:
error
```

Example 2

```
Input:
RDD
4
[1,2,3,4]

Output:
[2,1]
```

Example 3

```
Input:
RRRDD
4
[1,2,3,4]

Output:
[2,1]
```

Example 4

```
Input:
RRRDDRRDD
4
[1,2,3,4]

Output:
[]
```

Note: there is no output from this sequence since all values are removed. The program should print an empty list.

Parsing Brackets

Sam is working on a new program that is supposed to parse and validate the use of brackets in code programs. He came up with a way to do this for one type of bracket but run into problems when there are multiple types of brackets in the program to be validated. He asks you for help in completing the more advanced program.

A valid sequence of brackets has them correctly nested and sequenced. Here are some general rules:

- valid bracket pairs are `()`, `[]`, and `{}`
- any sequence of characters that does not include any brackets is a valid program (well, a least from the point of view of validating the brackets)
- all programs below are valid:
  - `(VALID)`
  - `[VALID]`
  - `{VALID}`
  - `VALID1VALID2`

  in which `VALID`, `VALID1` and `VALID2` are any valid programs

By these rules all the one-line programs below are valid:

```
a+b
```

```
()
```

```
[a+b]
```

```
if (a + b = c) {x = 50 * y;}
```

```
if (a + b = c) { if (x < y) x = 50 * z[7];}
```

The following programs are not valid:

```
a+b)
```

```
][
```

```
if (a + b = c) { if (x < y) x = 50 * z[7];
```

```
if (a + b = c) { if (x < y} x = 50 * z[7];}
```

Input

For simplicity, the entire input is given on a single line. It may consist of any sequence of characters. Each program is at most 3000 characters.

Output

The program should print `YES` if the program is valid (from the point of view of matching brackets), or `NO` followed by the character position (1-based) at which the first problem was detected if the program is not valid.

Example 1

```
Input:
if( x < y) {print x;}

Output:
YES
```

Example 2

```
Input:
if( x < y) {print x;

Output:
NO 21
```

Example 3

```
Input:
if (a + b = c) { if (x < y} x = 50 * z[7];}

Output:
NO 27
```

AWESOME Number

We say a number is AWESOME if it is prime and has 1 in its ones place. You are given a number N and you are asked to answer how many AWESOME numbers are not greater than N.

Input

The input consists of a single integer N (1 <= N <= 20,000,000).

Output

You should print **one line** containing a single integer, the number of awesome numbers that are no larger than N.

Example 1

```
Input:
6

Output:
0
```

Example 2

```
Input:
11

Output:
1
```

Because 11 is AWESOME.

Example 3

```
Input:
100

Output:
5
```

Because 11, 31, 41, 61, and 71 are all AWESOME.