A. Implement Stack

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

You have to implement a stack with two operations. Please, don't use stack implementation from the standard library. Implement your own stack using an array or pointer implementation.

The first operation, `push`, adds an element to the stack. The second operation, `pop`, retrieves the element. For each operation of the second type, you should return the retrieved element. It is guaranteed that there always exists an element to retrieve.

Input

The first line of the input contains the number of operations n ($1 \le n \le 10^5$). The next n lines describe n operations, one per line. The first integer in line is the identifier: `1` for `push`, `2` for `pop`. The second integer is only for `push` operation — the added element (positive, does not exceed 10^5).

Output

Output all the retrieved elements in the corresponding order, one per line.

Examples

input Copy

```
6
1 3
1 4
1 6
2
1 8
2
```

output Copy

```
6
8
```

input Copy

```
1
1 2
```

output Copy

input Copy

```
6
1 4
1 15
1 2
1 3
2
2
```

output Copy

```
3
2
```

B. Implement Queue

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

You have to implement a queue with two operations. Please, don't use queue implementation from the standard library. Implement your own queue using an array or pointer implementation.

You have to implement a queue with two operations:

- "+ x" — add an element x to the queue;
- "-" — retrieve an element from the queue.

For each retrieval operation output the result of the operation.

Input

The first line of the input contains the number of operations — n ($1 \le n \le 10^5$). The next n lines contain the description of operations one per line. The added element cannot exceed 10^9 by absolute value.

Output

Output the results of all retrieve operations in the corresponding order, one per line.

Example

input Copy

```
4
+ 1
+ 10
-
-
```

output Copy

```
1
10
```

C. Brackets

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

A bracket sequence is called regular (correct) if it can be obtained from any mathematical expression by erasing all symbols except brackets.

The formal definition of the correct bracket sequence (of round, square and curly brackets) is:

- empty sequence is correct;
- if `A` is a regular bracket sequence, then `(A)`, `[A]` and `{A}` are regular bracket sequences;
- if `A` and `B` are regular bracket sequences, then `AB` is a regular bracket sequence.

Determine whether the given sequence of round, square and curly brackets is regular (correct)?

Input

The only line of the input contains n brackets ($1 \leq n \leq 10^5$) without any delimiters. The allowed characters are: round, square and curly brackets.

Output

Output "`YES`", if the given sequence is a regular (correct) bracket sequence correct, and "`NO`" otherwise.

Examples

input	Copy
`()`	
output	Copy
`YES`	

input	Copy
`([]){}`	
output	Copy
`YES`	

input	Copy
`[]([)]`	
output	Copy
`NO`	

D. Stack-Sortable Permutation

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

A permutation of length n is array $p = [p_0, p_1, \ldots, p_{n-1}]$ where all elements are district numbers between 0 and $n-1$. For example, $p = [0, 2, 1]$, $p = [0]$ and $p = [4, 3, 2, 1, 0]$ are permutations.

Stack-sortable permutation is a permutation whose elements may be sorted by an algorithm whose internal storage is limited to a single stack data structure. In other words for given permutation the only allowed operations are:

- `A`: extract the leftmost element of p and push it into the stack s (after this operation the length of p is decreased by 1);
- `B`: extract an element from s and put it to the output.

For example $p = [1, 0, 2, 3]$ is stack-sortable:

- apply step `A`: after it $p = [0, 2, 3]$ and $s = [1]$;
- apply step `A`: after it $p = [2, 3]$ and $s = [1, 0]$;
- apply step `B`: after it $p = [2, 3]$ and $s = [1]$, output is `0`;
- apply step `B`: after it $p = [2, 3]$ and $s = []$, output is `0, 1`;
- apply step `A`: after it $p = [3]$ and $s = [2]$, output is `0, 1`;
- apply step `B`: after it $p = [3]$ and $s = []$, output is `0, 1, 2`;
- apply step `A`: after it $p = []$ and $s = [3]$, output is `0, 1, 2`;
- apply step `B`: after it $p = []$ and $s = []$, output is `0, 1, 2, 3`.
- the permutation is sorted!

Check the given permutation p if it is a stack-sortable permutation.

Input

The first line contains t ($1 \le t \le 10^4$) — number of test cases. Then t test cases follow.

Each test case starts with line containing integer n ($1 \le n \le 4 \cdot 10^5$) — length of p. The next lines contains district integers $p_0, p_1, \ldots, p_{n-1}$ ($0 \le p_i < n$).

The sum of values n in each test doesn't exceed $4 \cdot 10^5$.

Output

For each test case print `YES` or `NO`.

Example

input Copy

```
6
4
1 0 2 3
2
1 0
4
3 1 2 0
2
0 1
7
6 2 1 0 5 3 4
7
3 5 6 4 1 0 2
```

output Copy

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

E. Minimum on Stack

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

You have to implement the data structure that supports the following operations:

1. `1` x — add x to an end of the data structure.
2. `2` — retrieve the last element from the data structure.
3. `3` — find the minimal element in the data structure.

Input

The first line of the input contains one integer n ($1 \leq n \leq 10^6$) — the number of operations. Next n lines contain the description of operations, one per line. The argument x in an operation of the first type lies in $[-10^9, 10^9]$). It is guaranteed that before retrieval the data structure is not empty.

Output

Output the result for each operation of the third type, one per line.

Example

input Copy

```
8
1 2
1 3
1 -3
3
2
3
2
3
```

output Copy

```
-3
2
2
```

F. Plug-in

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

Polycarp thinks about the meaning of life very often. He does this constantly, even when typing in the editor. Every time he starts brooding he can no longer fully concentrate and repeatedly presses the keys that need to be pressed only once. For example, instead of the phrase "how are you" he can type "hhoow aaaare yyoouu".

Polycarp decided to automate the process of correcting such errors. He decided to write a plug-in to the text editor that will remove pairs of identical consecutive letters (if there are any in the text). Of course, this is not exactly what Polycarp needs, but he's got to start from something!

Help Polycarp and write the main plug-in module. Your program should remove from a string all pairs of identical letters, which are consecutive. If after the removal there appear new pairs, the program should remove them as well. Technically, its work should be equivalent to the following: while the string contains a pair of consecutive identical letters, the pair should be deleted. Note that deleting of the consecutive identical letters can be done in any order, as any order leads to the same result.

Input

The input data consists of a single line to be processed. The length of the line is from 1 to $2 \cdot 10^6$ characters inclusive. The string contains only lowercase Latin letters.

Output

Print the given string after it is processed. It is guaranteed that the result will contain at least one character.

Examples

input	Copy
hhoowaaaareyyoouu	
output	Copy
wre	

input	Copy
reallazy	
output	Copy
rezy	

input	Copy
abacabaabacabaa	
output	Copy
a	

G. Minus and Minus Give Plus

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

Everyone knows that two consecutive (adjacent) "minus" signs can be replaced with a single "plus" sign.

You are given the string s, consisting of "plus" and "minus" signs only. Zero or more operations can be performed with it. In each operation you can choose any two adjacent "minus" signs, and replace them with a single "plus" sign. Thus, in one operation, the length of the string is reduced by exactly 1.

You are given two strings s and t. Determine if you can use 0 or more operations to get the string t from the string s.

Input

The first line of the input contains an integer k ($1 \le k \le 10^5$), denoting the number of test cases in the input. The following lines contain descriptions of the test sets, each set consists of two lines. First comes the line containing s (the length of the line s does not exceed $2 \cdot 10^5$), then comes the line containing t (the length of the line t does not exceed $2 \cdot 10^5$). The lines s and t are non-empty, and they contain only "plus" and "minus" signs.

The sum of the lengths of lines s over all test cases in the input does not exceed $2 \cdot 10^5$. Similarly, the sum of the lengths of lines t over all test cases in the input does not exceed $2 \cdot 10^5$.

Output

Print k lines: the i-th line must contain `YES` if the answer to the i-th test case is positive, otherwise `NO`. Print `YES` and `NO` using uppercase letters only.

Example

input Copy

```
5
-+--+
-+++
--------
-+--+-
-
+
--
---
+++
+++
```

output Copy

```
YES
YES
NO
NO
YES
```

H. Replace To Make Regular Bracket Sequence

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

You are given string s consists of opening and closing brackets of four kinds <>, {}, [], (). There are two types of brackets: opening and closing. You can replace any bracket by another of the same type. For example, you can replace < by the bracket {, but you can't replace it by) or >.

The following definition of a regular bracket sequence is well-known, so you can be familiar with it.

Let's define a regular bracket sequence (RBS). Empty string is RBS. Let s_1 and s_2 be a RBS then the strings <s_1>s_2, {s_1}s_2, [s_1]s_2, (s_1)s_2 are also RBS.

For example the string "[[(){}]<>]" is RBS, but the strings "[](]" and "][()()" are not.

Determine the least number of replaces to make the string s RBS.

Input

The only line contains a non empty string s, consisting of only opening and closing brackets of four kinds. The length of s does not exceed 10^6.

Output

If it's impossible to get RBS from s print Impossible.

Otherwise print the least number of replaces needed to get RBS from s.

Examples

input	Copy
[<}){}	
output	Copy
2	

input	Copy
{()}[]	
output	Copy
0	

input	Copy
]]	
output	Copy
Impossible	