CS F211
Data Structures and Algorithms
Assignment - 8

Allowed languages: **C**

March 26, 2021

General Tips

- Try to use functions as much as possible in your code. Functions increase reusability and the pass-by-value feature provides a significant help sometimes. Modularizing your code also helps you to debug efficiently.
- Use `scanf` to read characters/strings from STDIN. Avoid using `getchar`, `getc` or `gets`. Try to read up about character suppression in `scanf` as it will be very helpful in some of the problems.
- Use `printf` instead of `putc`, `putchar` or `puts` to print character/string output on STDOUT.
- Indent your code appropriately and use proper variable names. These increase readability and writability of the code. Also, use comments wherever necessary.
- Use a proper IDEs like Sublime Text or VSCode as they help to run and test your code on multiple test-cases easily. You can install Windows Subsystem Linux (WSL) or MinGW 7.3.0, if you are Windows user to compile and run your programs. Alternatively, you can run and test your codes on Online GDB. If you are using WSL or Linux to run your programs, make sure that the gcc version is `gcc 5.4.1 c99`.

Special Instructions

For binary-tree problems A, B, C, D, F you **must** use a pointer-based approach to solve these problems (syntax below). After inputting and constructing the tree,all subsequent operations must be performed **only on** the struct Node* objects. **Your code will be manually checked to ensure you did this, and you will be awarded zero score if any other approach is used.**

```
struct Node {
        int val;
        struct Node *left;
        struct Node *right;
};
```

Similarly, Question G also requires implementation of the solution using a specified data structure. Any other approaches will lead to a zero score.

A: Binary Tree Blues - I

Given a binary tree, write a program to print the *zig-zag level order traversal*. To do this:

1. The *level* of a node is defined as the distance from the root node. Consequently, the root node has a level 0, and the children of the root would have level 1.
2. Start with level 0, & print all nodes in that level. Then print all nodes of level 1, and so on...
3. For all even levels, print the nodes of that level from left to right. For odd levels, print the nodes from right to left.

See the example and figure for clearer understanding.

Input

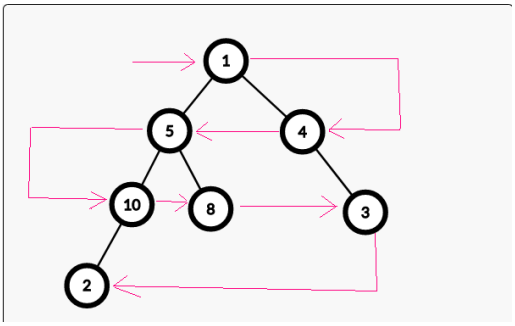
The first line contains a single integer N ($1 \leq N \leq 10^3$), the number of nodes in the binary tree. The next line contains N space-separated integers denoting the **values** of the i^{th} node (zero-indexed). The 0^{th} element in this list is guaranteed to be the root node. The next $N-1$ lines contain two integers (x and y respectively) and a single character. This means that the x^{th} element in the list of nodes is a parent of the y^{th} element in the list of nodes, and the character (either 'L' or 'R') denotes whether it is the left child or right child. All values in the binary tree are in $[1, 10^9]$.

Output

Print N space separated integers representing the zig-zag level order traversal of the tree.

---

input
7
1 5 4 3 8 10 2
0 1 L
0 2 R
2 3 R
1 4 R
1 5 L
5 6 L
output
1 4 5 10 8 3 2
explanation
Follow the arrows.



B: Binary Tree Blues - II

Given a binary tree with N nodes, construct the tree and perform the following operations on it:

- Perform a **preorder** traversal and store it in an array A. Perform a **postorder** traversal and store it in array B. Perform an **inorder** traversal and store it in array C.
- Calculate $S = \sum_{i=0}^{N}\{abs(A_i - B_i) * C_i\}$ and since this number might be large, output it modulo $10^9 + 7$.

Input

The first line contains a single integer N ($1 \leq N \leq 10^3$), the number of nodes in the binary tree. The next line contains N space-separated integers denoting the **values** of the i^{th} node (zero-indexed). The 0^{th} element in this list is guaranteed to be the root node. The next $N - 1$ lines contain two integers (x and y respectively) and a single character. This means that the x^{th} element in the list of nodes is a parent of the y^{th} element in the list of nodes, and the character (either 'L' or 'R') denotes whether it is the left child or right child. All values in the binary tree are in $[1, 10^9]$.

Output

Output the value S as described in the question.

```
input
5
1 3 5 79 8
0 1 L
0 2 R
2 3 L
3 4 R
output
6484
explanation
Preorder traversal is [1, 3, 5, 79, 8].
Postorder traversal is [3, 8, 79, 5, 1].
Inorder traversal is [3, 1, 79, 8, 5]
```

C: Binary Tree Blues - III

Perform an **inorder traversal** of a binary tree **without using recursion**. Your program **must** have exactly **one** function definition only - main() and no other functions. *Hint: You might need to use a stack. Additional thinking: can you do postorder, or pre-order traversals without using recursion?*

Input

The first line contains a single integer N ($1 \leq N \leq 10^3$), the number of nodes in the binary tree. The next line contains N space-separated integers denoting the **values** of the i^{th} node (zero-indexed). The 0^{th} element in this list is guaranteed to be the root node. The next $N-1$ lines contain two integers (x and y respectively) and a single character. This means that the x^{th} element in the list of nodes is a parent of the y^{th} element in the list of nodes, and the character (either 'L' or 'R') denotes whether it is the left child or right child. All values in the binary tree are in $[1, 10^9]$.

Output

Output N space separated integers represnting the inorder traversal of the given tree.

```
input
5 1
1 3 5 79 8
0 1 L
0 2 R
2 3 L
3 4 R
output
3 1 79 8 5
```

D: Binary Tree Blues - IV

Given a binary tree with N nodes perform T operations on it. The format of each query is:

- "**DELETE A**": Find a node which has value A stored in it, and delete that node and all it's descendants.
- "**SWAP A B**": Find nodes that have values A and B in them respectively. Swap the subtrees rooted at A and B. Swap A (and all its children) with B (and all its children).

After completing all T operations, print the **diameter** of the final binary tree. To calculate the diameter, identify the longest path (without cycles) between *any two leaves* in the binary tree. The path length between two nodes is the number of edges between them.

Input

The first line contains space-separated integers N ($1 \leq N \leq 10^3$) and T ($0 \leq T \leq 10$). The next line contains N space-separated **unique** integers denoting the **values** of the i^{th} node (0-indexed). The 0^{th} element in this list is the root node. The next $N-1$ lines contain two integers (x and y respectively) and a single character, meaning that the x^{th} element in the list of nodes is a parent of the y^{th} element in the list of nodes, and the character ('L' or 'R') denotes whether it is the left or right child. The next T lines contain one query each.

Output

Print the diameter of the final tree.

---

input

```
8 2
2 6 5 1 3 4 99 128
0 1 L
0 2 R
1 3 L
2 4 L
4 5 R
5 6 L
6 7 R
DELETE 99
SWAP 1 3
```

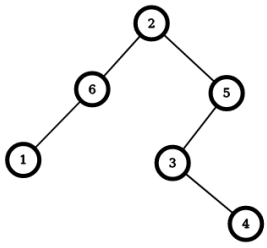
output

```
5
```

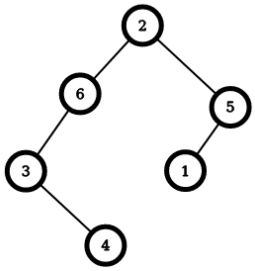
explanation

```
The diameter is from
node 4 to node 1.
```

After delete operation:



After swap operation:



E: Binary Tree Blues - V

Given the inorder and preorder traversals of a binary tree, print its' postorder traversal.

Input

The first line contains an integer N representing the number of elements in the binary tree ($1 \leq N \leq 10^3$). The second line contains N space separated integers representing the inorder traversal of the tree. The third line, which is similarly formatted, contains the preorder traversal of the tree.

Output

Print N space-separated integers representing the postorder traversal of the tree.

```
input
3
1 3 2
1 2 3

output
3 2 1
```

F. Binary Tree Blues - VI: Binary Possibilities

You are given a Binary Search Tree and q queries. Each query consists of two nodes. Find the Lowest Common Ancestor of these nodes. Let T be a rooted tree. The lowest common ancestor between two nodes n_1 and n_2 is defined as the node in T that has both n_1 and n_2 as descendants (where we allow a node to be a descendant of itself) and is farthest possible from the root while still satisfying the previous condition.

Input

The first line contains two space-separated integers N ($1 \leq N \leq 10^3$), the number of nodes in the BST and q ($0 \leq q \leq 100$), the number of queries. The next line contains N space-separated integers denoting the **values** of the i^{th} node (zero-indexed). The 0^{th} element in this list is guaranteed to be the root node. The next $N-1$ lines contain two integers (x and y respectively) and a single character. This means that the x^{th} element in the list of nodes is a parent of the y^{th} element in the list of nodes, and the character (either 'L' or 'R') denotes whether it is the left child or right child. All values in the binary tree are in $[1, 10^9]$. The next q lines each contain two integers a and b representing the nodes for which you have to find the least common ancestor.

Output

Print q space separated integers, one for each query print a single integer representing the node that is the least common ancestor of the two nodes given in the query.

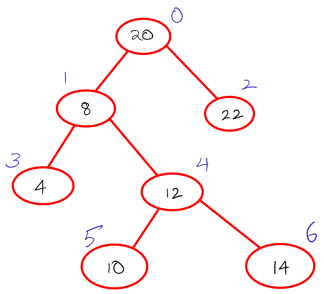
---

input

```
7 5
20 8 22 4 12 10 14
0 1 L
0 2 R
1 3 L
1 4 R
4 5 L
4 6 R
1 2
5 2
1 6
5 6
1 1
```

output

```
0 0 1 4 1
```



explanation

For the first case we can see that the LCA is 0. In the second case the LCA is also 0. In the third case, the LCA is 1 because the LCA can be one of the nodes themselves. The LCA in the fourth case is 4. In the last case, they are both the same node so the answer is 1 itself.

G. Stack Overkill

Implement a simple stack that performs two operations: push and pop. push should push an element on to the top of the stack and pop removes the element on the top of the stack. There is one catch though. While implementing this stack, you must do so using **two queues**. You will have N queries. All of them will be of three types:

1. "PUSH X ": push the number X on top of the stack
2. "POP": pop the topmost element from the stack.
3. "PRINT": print the value of the element on top of the stack

*Note that you have to use exactly two queues to implement this stack. Your code will be **manually checked** (by humans) to make sure you have done this. Implementing it in any other fashion will lead to a **zero score**.*

Input

The first line of input contains an integer N ($1 \leq N \leq 2 \times 10^5$). The next N lines contain one of the above operations. If the operation is "PRINT" then it will be followed by a single integer "X" ($1 \leq X \leq 10^9$). It is guaranteed that the stack is not empty when the operation is "PRINT" or "POP".

Output

For each "PRINT" function, print a single integer in a single line representing the element at the top of the stack at that point in time.

```
input
7
PUSH 3
PUSH 5
PRINT
POP
PRINT
PUSH 100
PRINT

output
5
3
100

explanation
It's pretty easy to follow the stack operations to see that the sample case is correct
```

H: I Know You'll Google This

Given an array having both positive and negative integers. The task is to compute the length of the largest contiguous subarray with sum 0.

Input

The first line contains one integer N, the sizes of array ($1 \leq N, \leq 2 \times 10^5$). The next line contains contains N integers each integer representing a_i ($-10^9 \leq a_i \leq 10^9$).

Output

Output one integer, the length of the largest such subarray. If no such subarray exists, print 0.

```
input
8
15 -2 2 -8 1 7 10 23

output
5
explanation
The largest subarray is:  -2 2 -8 1 7
```

I: Infuriatingly not Binary Search

You are given an array that was initially sorted, but now it has been cyclically rotated k times to the right. (k is unknown to you). Given q queries where each query contains a number x_i, find whether x_i exists in the array or not. **Your algorithm has to be O(1) space complexity (not including storing the original array). Your code will be checked for this**.
Also note that the array can contain duplicate numbers.

Input

The first line contains the integer N ($1 \leq N \leq 10^5$), the number of elements in the array. The next line contains all the elements in the array ($1 \leq a_i \leq 10^9$), and it is guaranteed that all a_i are distinct. The next line contains an integer q ($1 \leq q \leq 10^5$) representing the number of queries. Each of the next q lines contains a single integer x_i, the number for which you have to check whether or not it is present in the array.

Output

For each query output a single string, "YES" if the number exists in the array or "NO" if it does not. Note that you must print the strings without quotes and all in capital letters.

---

input
8
5 6 7 8 10 1 2 3
5
5
8
12
2
1000000000
output
YES
YES
NO
YES
NO
explanation
In the five queries, you can easily see whether or not the corresponding elements are present in the array or not

---

J: Basic DP

Write an efficient program to find the sum of elements in a contiguous subarray within an array of numbers which has the largest sum.

Input

The first line contains the integer N ($1 \leq N \leq 10^5$), the number of elements in the array. The next line contains N space separated integers, a_i ($-10^9 \leq a_i \leq 10^9$) - representing the series of numbers.

Output

Print a single integer representing the sum of the subarray with that largest sum.

```
input
8
-2 -3 4 -1 -2 1 5 -3

output
7
explanation
Here the largest contiguous subsegment is from indices 3 to 7 inclusive
```
