Programming Part

Red-Black Tree (20 pts)

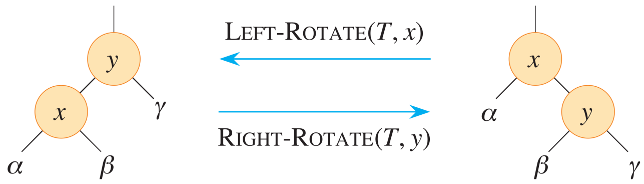
Problem Description

In this problem, we ask you to implement the LEFT-ROTATE and RIGHT-ROTATE operations on a red-black tree. The operations are illustrated in the figure below and are identical to the ones introduced in the lecture. The pseudo code of LEFT-ROTATE is given as follows, and you will need to implement your own RIGHT-ROTATE (which is a with-lecture quiz, remember?). Note that after the rotations, some red-black tree properties might not be satisfied anymore. In addition, the problem does not require your program to insert or delete a tree node, and thus there is no need for the implementations of these operations.

Here we define the *depth* of a tree node x as the number of nodes in the path from node x to the root of the tree, excluding x. For example, the root has a depth of 0 as the root itself is excluded. The child nodes of the root has a depth of 1, counting only the root node in the path. We can also define *black-depth* of a red-black tree node y as the number of *black* nodes in the path from node y to the root of the tree, excluding y. For example, the black-depth of the root of a red-black tree is always 0, and the black-depth of the child nodes of the root is always 1.

In this problem, we will ask you to output the depth and the black-depth of certain nodes in the tree.

```
LEFT-ROTATE(T, x)
y = x.right
x.right = y.left
if y.left ≠ T.nil
    y.left.p = x
y.p = x.p
if x.p == T.nil
    T.root = y
elseif x == x.p.left
    x.p.left = y
else x.p.right = y
y.left = x
x.p = y
```



Input

The first line has a single positive integer number N, which specifies the number of internal nodes (non-NIL nodes) in the given red-black tree.

Each of the following N lines includes the information of one node, and the nodes in these N lines are given in the order of the pre-order traversal of the given red-black tree. In each line, we have one character C and one integer number k. C represents the color of the node, and is either `R`, representing red, or `B`, representing black. k is the key of the node. Note that since a legal red-black tree is also a legal binary search tree, the list of nodes with the order of a pre-order traversal can uniquely determine a red-black tree.

The next line has a single integer number M, which specifies the number of operations to be performed on the given red-black tree. Each of the following M lines is in one of these three formats.

1. `L k` – You should execute LEFT-ROTATE(T, x), where x is the node with key k.
2. `R k` – You should execute RIGHT-ROTATE(T, y), where y is the node with key k.
3. `P k` – You should output information about the node with the given key k, in the output format specified below.

A single space character is used to separate different items in the same line.

You should execute all operations according to the given order. We ensure that all keys of the given red-black tree are *distinct*.

Output

For each given `P` operation with key k, output one line in this format – `LK RK BD D`, with a single space character separating consecutive numbers. Assume node x is the node with key k. LK is the key of the left child node of x, RK is the key of the right child node of x. If the left child of x is NIL, then LK is -1. The same applies to RK. Finally, BD is the black-depth of x and D is the depth of x.

Constraints

- $2 \leq N \leq 3 \times 10^5$
- $1 \leq M \leq 3 \times 10^5$
- $0 \leq k \leq 10^9$
- We ensure that all rotation and output operations can be executed successfully.
- Although it is possible that a search in the given tree have a worst-case running time of $O(n)$, we ensure that the judge will not produce a Time Limit Exceeded (TLE) error in our test cases if you execute the operations in the given order.

Sample Testcases

Sample Input 1

```
5
B 2
B 1
B 4
R 3
R 5
5
P 4
L 2
P 4
R 4
P 4
```

Sample Output 1

```
3 5 1 1
2 5 0 0
3 5 1 1
```

Sample Input 2

```
2
B 2
R 1
3
P 2
R 2
P 2
```

Sample Output 2

```
1 -1 0 0
-1 -1 0 1
```