Order Statistics on an AVL Tree

Note: You should complete the AVL tree question first BEFORE attempting this question

One of the operations not provided by the Java API for **TreeSet** is the **getRank**() operation. This allows to get the rank of an element in a BST of **N** elements where 1 is rank of the *smallest* element and **N** is the rank of the *largest* element. However, it is very simple to extend an implementation of AVL tree to support this operation.

Thus, you are asked to implement an AVL tree that supports the following 2 operations:

Operation	Description
`I [x]`	Insert **[x]** into the AVL tree. It Is guaranteed the element does not currently exist in the AVL tree.
`R [x]`	Get the rank **[x]** in the AVL tree. It is guaranteed that **[x]** currently exists in the AVL tree.

All the operations should run in O(log **N**) time where **N** is the number of elements in the BST. In addition, print all the elements left in the BST at the end of all the operations in **increasing** order. It is guaranteed there will be at least one element in the BST at the end of all the operations.

Input
The first line of input contains an integer **Q**. **Q** lines will follow, representing an operation each. The operations should be executed in order and the format would be as described in the table above. (See sample)

Output
For each **R** operation, output the rank. At the end of all **Q** operations, output all the elements left in the BST at the end of all the operations in **increasing** order. Add a single space between two consecutive values. **Print a space before the first value and do not print a space after the last value.** Instead, remember to print an end-line character at the end of the output.

Limits

- $1 \leq \mathbf{Q} \leq 200{,}000$
- All the values will range from 1 to 10^9 inclusive.
- It is guaranteed that the element inserted in the **I** operation does not currently exist in the AVL tree.
- It is guaranteed that the element **[x]** in the **R** operation currently exists in the AVL tree.
- It is guaranteed there will be at least one element in the BST at the end of all the operations.

Sample Input (**rank1.in**)	Sample Output (**rank1.out**)
10 I 5 I 3 R 5 R 3 I 6 I 2 I 4 R 5 R 3 R 6	2 1 4 2 5 2 3 4 5 6

Explanation

After the first 2 **I** operations, the BST will contain [3, 5] in increasing order. Thus, the rank of 5 is 2 and the rank of 3 is 1. The next 3 **I** operations will change the BST to contain [2, 3, 4, 5, 6] in increasing order. Then the rank of 5 is 4, the rank of 3 is 2 and the rank of 6 is 5.

Notes:

1. **You should develop your program by extending the BST class that you implemented in the question on AVL tree.** The solution to that should be developed using the BST class used in lecture.
2. You are not allowed to import any additional Java classes.
3. You may add any helper classes or methods as needed. You will also likely need to modify the implementations of the other methods to support this new method. However, you should not modify the function signatures and functionalities of **insert, predecessor, successor and inorder**. You should also not modify the function signature of **rank**.
4. You should **NOT** modify `Grader.java` and you should submit only `BST.java` to CodeCrunch
5. To test your code, you can run **`bash check.sh Grader`**