Implementing an AVL tree

In Lecture, a basic Binary Search Tree (BST) implementation and the various cases of rotations in an AVL tree was covered. However, the implementation of those rotations was *left as an exercise to the reader*. This question is the *exercise*. In this question, you will have to implement the rotations for an AVL tree. For simplicity, you do not need to support any delete operations.

In summary, you need to implement an AVL tree that supports the following operations:

Operation	Description
`I [x]`	Insert **[x]** into the AVL tree. It Is guaranteed the element does **not currently exist** in the AVL tree.
`P [x]`	Get the largest element in the AVL tree smaller than **[x]**, also known as the predecessor of **[x]** and output the value. It is guaranteed that **[x] currently exists** in the AVL tree and is **not the smallest element**.
`S [x]`	Get the smallest element in the AVL tree larger than **[x]**, also known as the successor of **[x]** and output the value. It is guaranteed that **[x] currently exists** in the AVL tree and is **not the largest element**.

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 **P** and **S** operation, output the element. 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 **P** operation currently exists in the AVL tree and is not the smallest element.
- It is guaranteed that the element **[x]** in the **S** operation currently exists in the AVL tree and is not the largest element.
- It is guaranteed there will be at least one element in the BST at the end of all the operations.

Sample Input (**avl1.in**)	Sample Output (**avl1.out**)
9 I 5 I 3 I 7 P 5 S 5 I 4 I 6 P 5 S 5	3 7 4 6 3 4 5 6 7

Explanation

After the first 3 **I** operations, the BST will contain [3, 5, 7] in increasing order. In that case, the predecessor of 5 is 3 and the successor of 5 is 7. The next 2 **I** operations will change the BST to contain [3, 4, 5, 6, 7] in increasing order. Then, the predecessor of 5 is 4 and the successor of 5 is 6.

Notes:

1. **You should develop your program by extending the BST class given in lecture.** This has been provided to you as `BST.java`.
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.**
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`**