**Programming Task P1.** / 20 P

**Passwort für Einschreibung:** `asymptotic`

**Einreichung:** `https://judge.inf.ethz.ch/team/websubmit.php?cid=28784&problem=AVLAugme`

## AVL Tree Augmentation

Your task is to augment an AVL tree to support the following `rank(x)` operation:

`rank(`$x$`)` : Given an integer $x$, returns the number of values $y \leq x$ stored in the AVL tree.

Most of the implementation of the AVL tree is already provided by the template (reading the input, inserting a new element, writing the output).

The tree is stored as a group of `Node` objects. Each `Node` object $v$ has five fields:

`parent`: a pointer to the parent of $v$ in the tree (or `null` if $v$ is the root of the tree).

`leftChild`: a pointer to the left child of $v$ (or `null` if no such child exists).

`rightChild`: a pointer to the left child of $v$ (or `null` if no such child exists).

`value`: the integer value associated with $v$.

`balanceFactor`: the balance factor of $v$, i.e., the height of the subtree rooted at the right child of $v$ minus the height of the subtree rooted at the left child of $v$.

Notice that for every pointer, `leftChild`, `rightChild`, or `parent`, from one vertex $v$ to another vertex $u$, there is a corresponding pointer from $u$ to $v$. The provided AVL tree implementation also contains an additional pointer, named `root`, to the current root node of the tree (`root` is `null` when the tree is empty).

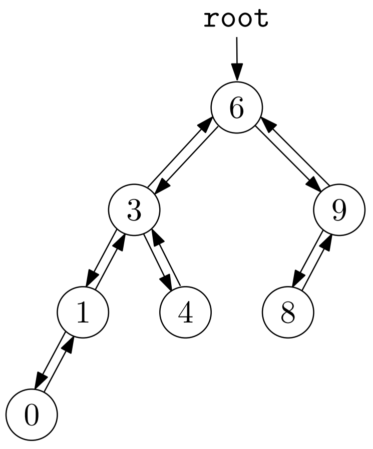
To solve the task you will need to edit the provded code so that the `rank(`$x$`)` operation can be implemented in $O(\log n)$ time, where $n$ is the number of nodes in the AVL tree. The asymptotic complexity of the insert operation must remain unchanged.

The values inserted are distinct integers between 0 and 1 000 000.

(Continues on the next page)

**Example**

The following figure shows the structure of an AVL tree in which `rank`(5)=4 and `rank`(8)=6.



**Grading** You can get up to 20 judge points. The program should implement the insertion and `rank(`$x$`)` operations in $O(\log n)$ time per operation (with reasonable hidden constants), where $n$ is the number of nodes in the AVL tree. Less efficient solutions can obtain up to 10 points.

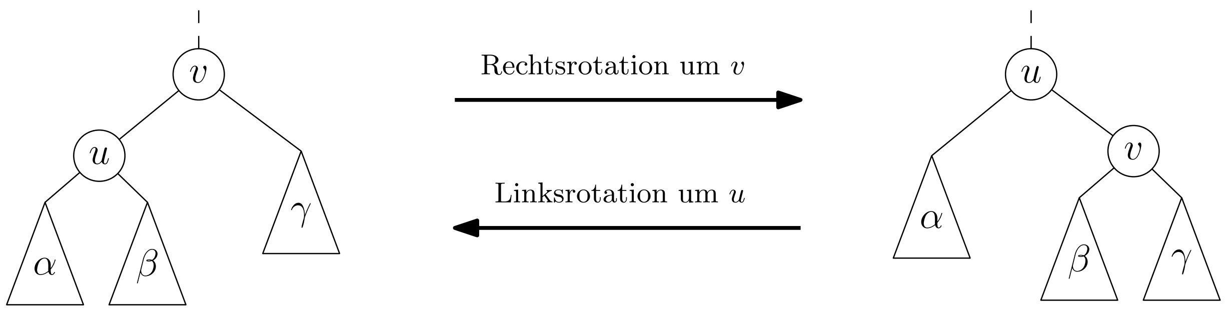
**Instructions** For this exercise, we provide a program template as an Eclipse project in your workspace, and the template already implements most of the functionality, exept for the modifications needed to support the `rank(`$x$`)` operation.

The project also contains data for your local testing and a `Judge.java` program that runs your `Main.java` on all the local tests – just open and run `Judge.java` in the project. The local test data are different from the data that are used in the online judge.

Submit only your `Main.java`.

**Notes**

For your convenience, the following figure shows a generic right (resp. left) tree roation around vertex $v$ (resp. $u$).



*The input and output are handled by the template – you should not need the rest of this text.*

**Input** The input of this problem consists of a number of test cases. The first line of the input contains the number of test cases. The first line of each test case contains the number $m$ of operations to perform. The next $m$ lines each contain a character $C$ and an integer $x$, separated by a space. The character $C$ can be either "I" or "R". If $C$ is "I" then $x$ must be inserted into the AVL tree. If $C$ is "R", then a `rank`$(x)$ operation must be performed.

**Output** The output contains one line for each `rank`$(x)$ operation. More precisely, the $i$-th line of the output contains a single integer corresponding to the result of the $i$-th `rank`$(x)$ operation in the input.

*Example input.*

```
1
10
I 6
I 3
I 9
R 10
I 4
I 8
I 1
I 0
R 5
R 8
```

*Example output:*

```
3
4
6
```

*Space for your notes. These will not be graded. Only what was submitted to the judge counts for this exercise.*