AVL Tree

Anoop S Babu

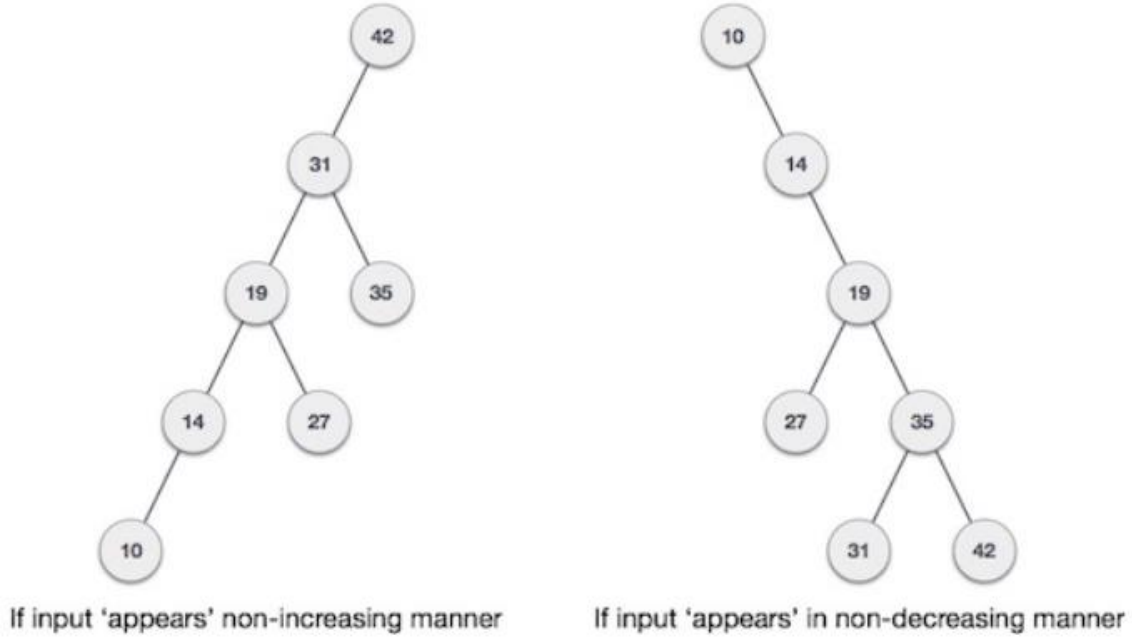
Faculty Associate

Dept. of Computer Science & Engineering

bsanoop@am.amrita.edu

BST's Worst-Case Performance

What if the input to binary search tree comes in a sorted (ascending or descending) manner?

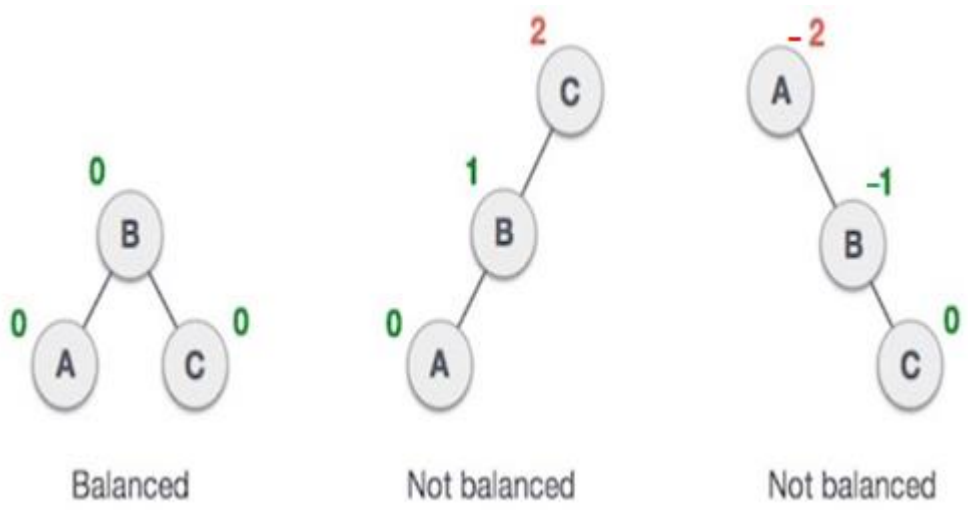


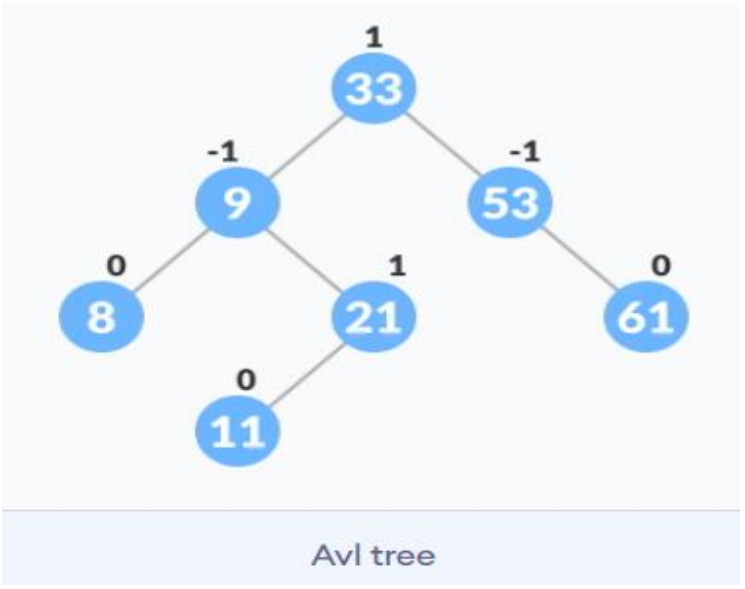
It is observed that BST's worst-case performance is closest to linear search algorithms, that is O(n).

AVL Tree

- AVL trees are **height balancing** binary search tree, in which each node having a **balance factor** not more than 1 (that is either **-1, 0 or +1**).
- **Balance Factor**: The difference in the height of left and right sub-trees.

```
BalanceFactor = height(left-sutree) – height(right-sutree)
```





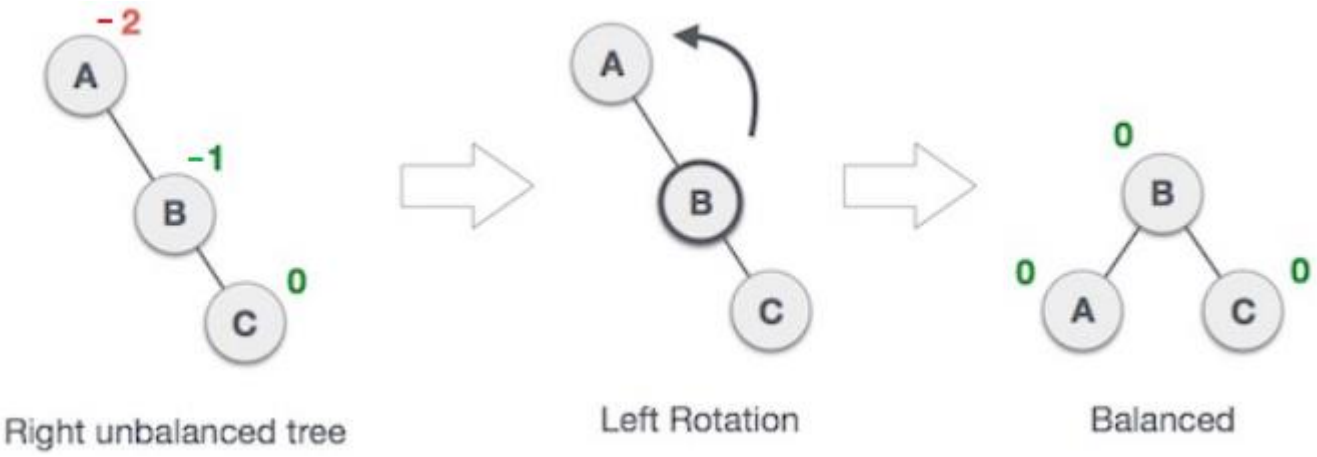
AVL Rotations

- To balance itself, an AVL tree may perform the following four kinds of rotations.
  - Left rotation
  - Right rotation
  - Left-Right rotation
  - Right-Left rotation
- The first two rotations are single rotations and the next two rotations are double rotations.

AVL Rotations

Left Rotation

Suppose we have A, B, C

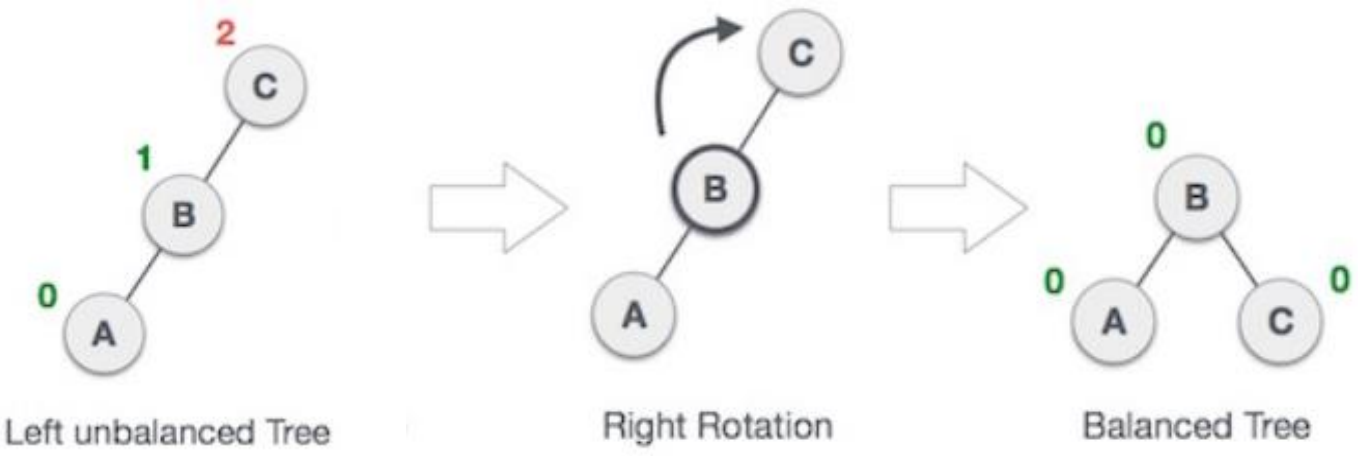


Node **A** has become unbalanced as a node is inserted in the right subtree of A's right subtree. We perform the left rotation by making **A** the left-subtree of B.

AVL Rotations

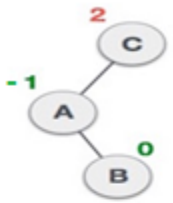
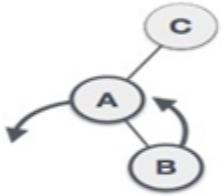
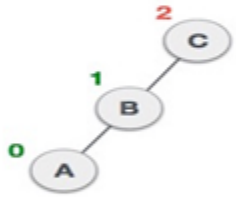
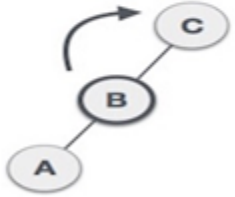
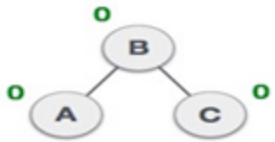
Right Rotation

Suppose we have C, B, A

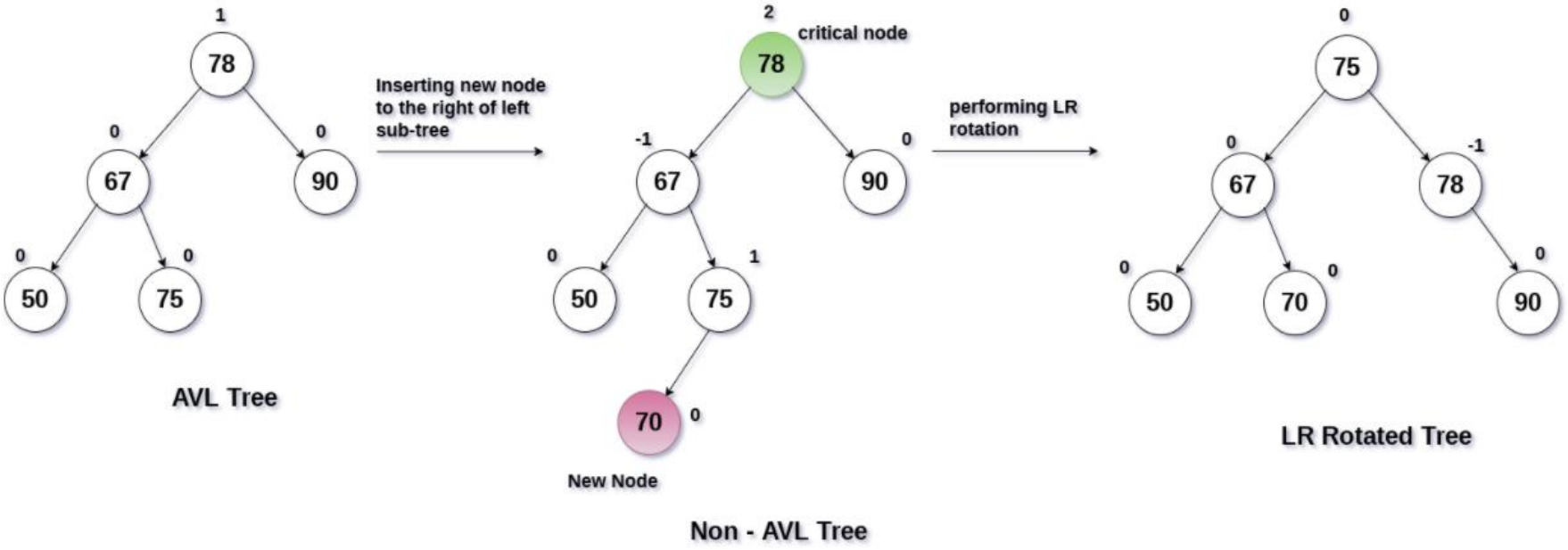


Node **C** has become unbalanced as a node is inserted in the left subtree of C's left subtree. We perform the right rotation by making **C** the right-subtree of B.

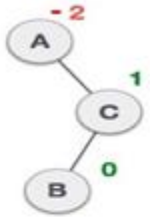
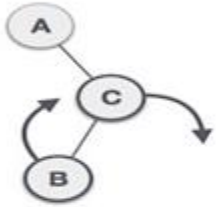
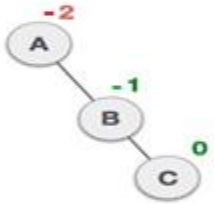
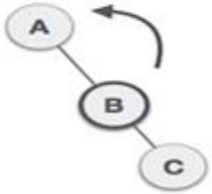
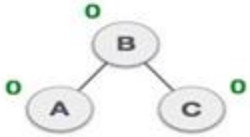
Left Right Rotation

State	Action
	A node has been inserted into the right subtree of the left subtree. This makes **C** an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.
	We first perform the left rotation on the left subtree of **C**. This makes **A**, the left subtree of **B**.
	Node **C** is still unbalanced, however now, it is because of the left-subtree of the left-subtree.
	We shall now right-rotate the tree, making **B** the new root node of this subtree. **C** now becomes the right subtree of its own left subtree.
	The tree is now balanced.

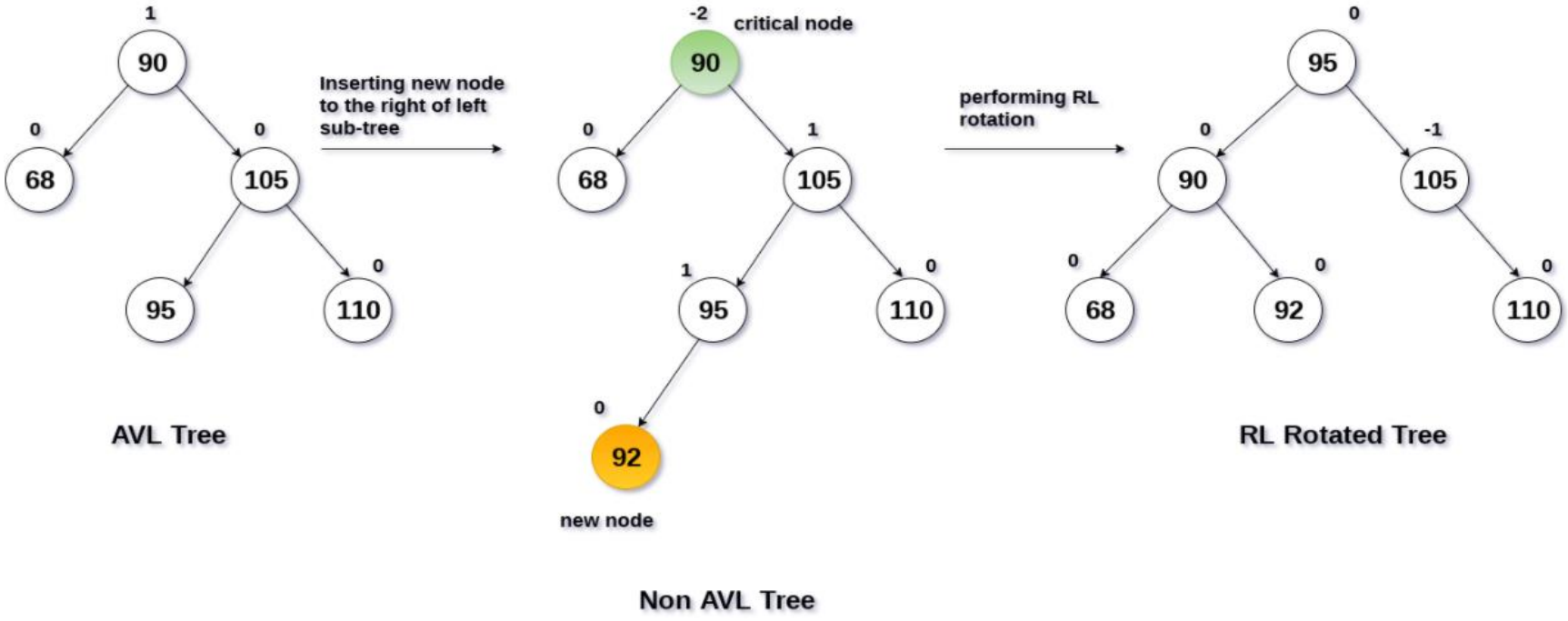
Left Right Rotation: Example



Right Left Rotation

State	Action
	A node has been inserted into the left subtree of the right subtree. This makes **A**, an unbalanced node with balance factor 2.
	First, we perform the right rotation along **C** node, making **C** the right subtree of its own left subtree **B**. Now, **B** becomes the right subtree of **A**.
	Node **A** is still unbalanced because of the right subtree of its right subtree and requires a left rotation.
	A left rotation is performed by making **B** the new root node of the subtree. **A** becomes the left subtree of its right subtree **B**.
	The tree is now balanced.

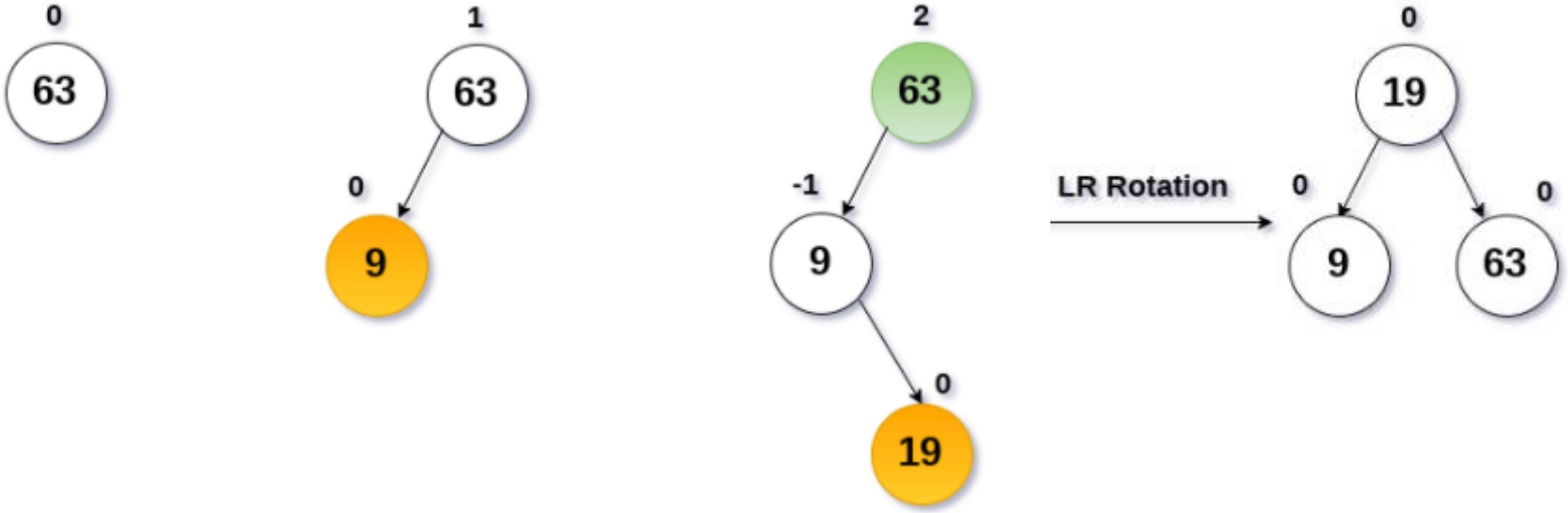
Right Left Rotation: Example



AVL Tree: Insertion

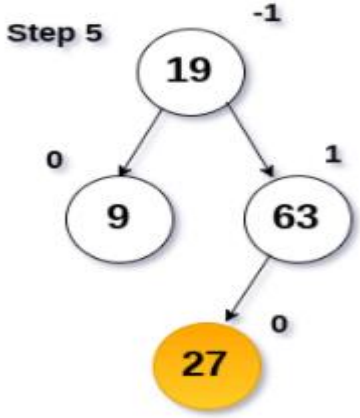
- Insertion in AVL tree is performed in the same way as it is performed in a BST.
- Check the balance factor and if needed, the tree can be balanced by applying rotations.

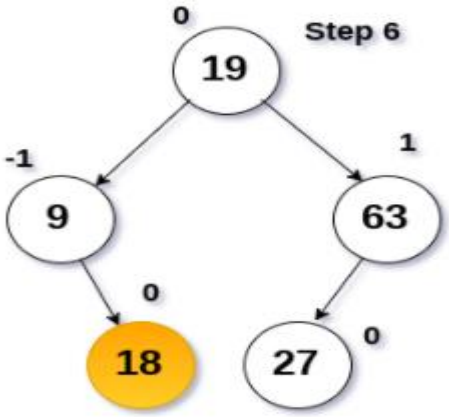
Construct an AVL tree by inserting the elements in the order (**63, 9, 19, 27, 18, 108, 99, 81**)

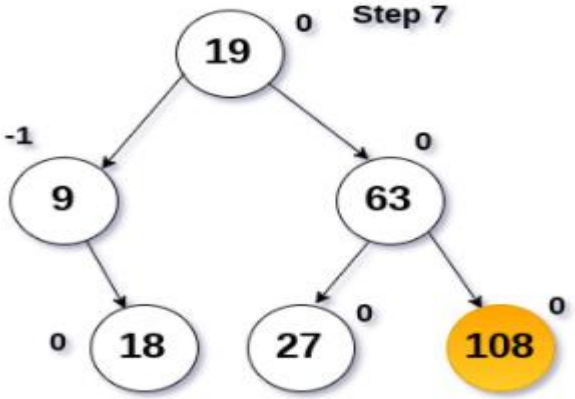


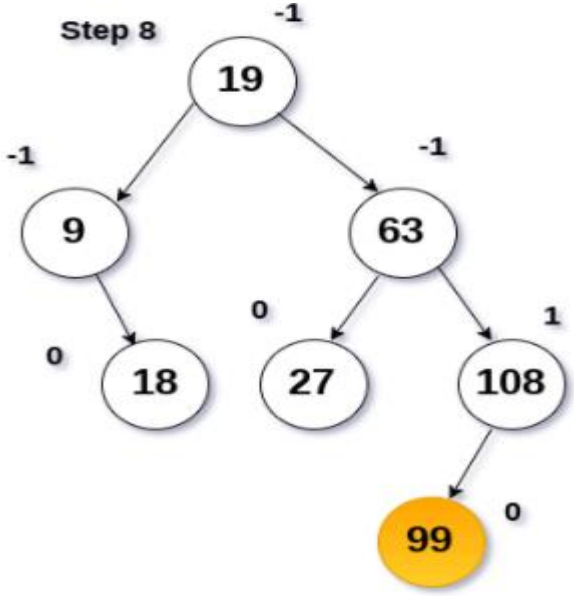
AVL Tree: Insertion

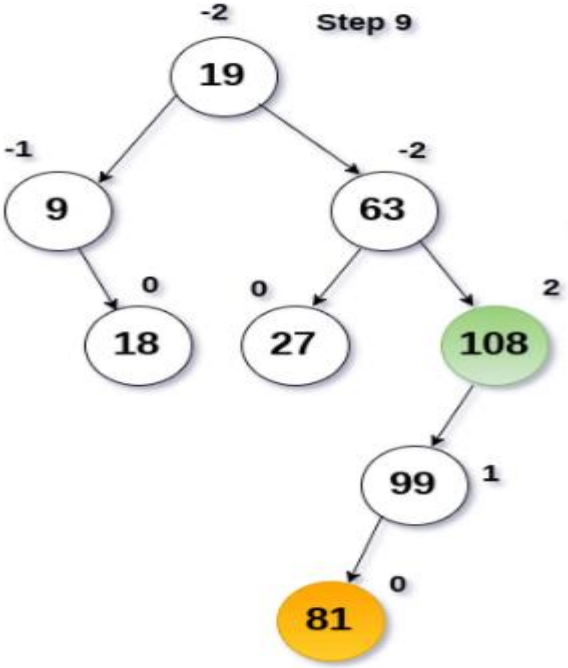
(63, 9, 19, 27, 18, 108, 99, 81)

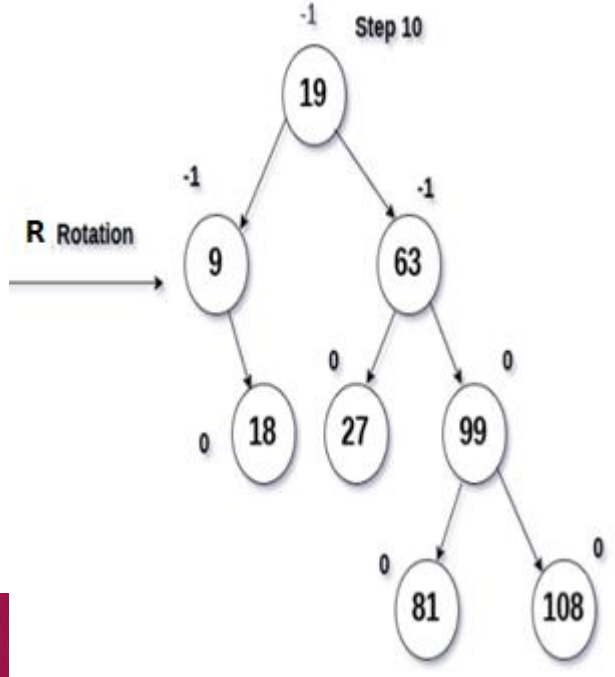






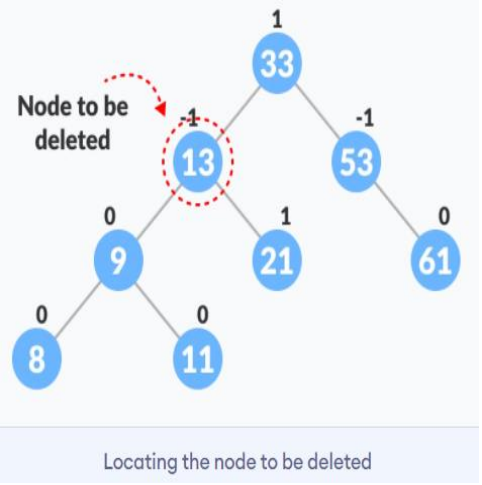




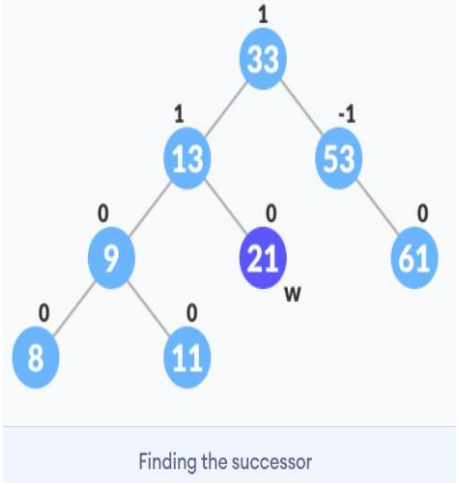


AVL Tree: Deletion

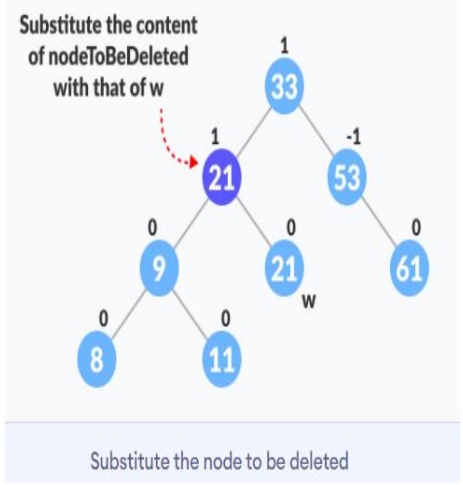
- Deletion in AVL tree is performed in the same way as it is performed in a BST.
- Check the balance factor and if needed, the tree can be balanced by applying rotations.



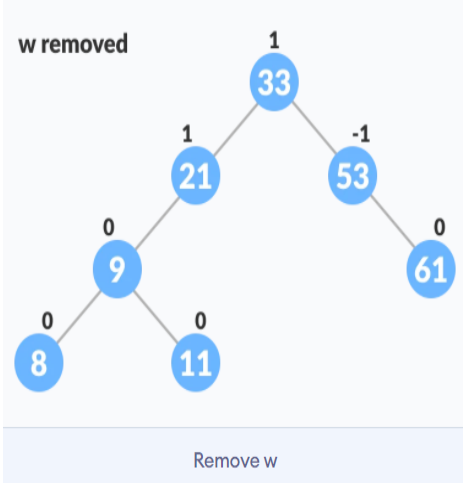
Locating the node to be deleted



Finding the successor

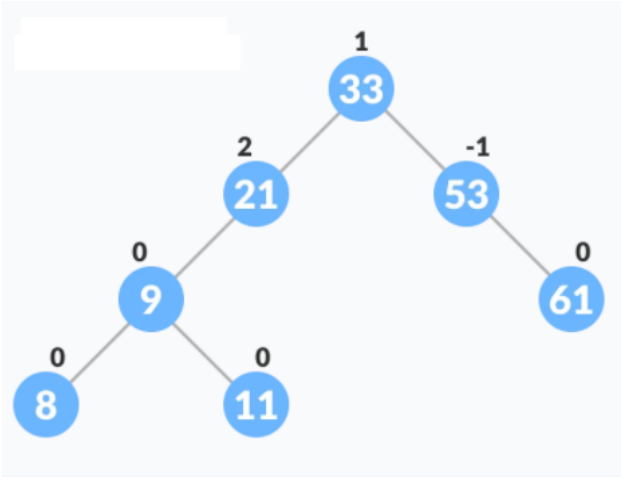


Substitute the node to be deleted



Remove w

AVL Tree: Deletion



Update balance factor of the nodes

Rebalance the tree if the balance factor of any of the nodes is not equal to -1, 0 or 1.

If balanceFactor of currentNode > 1,

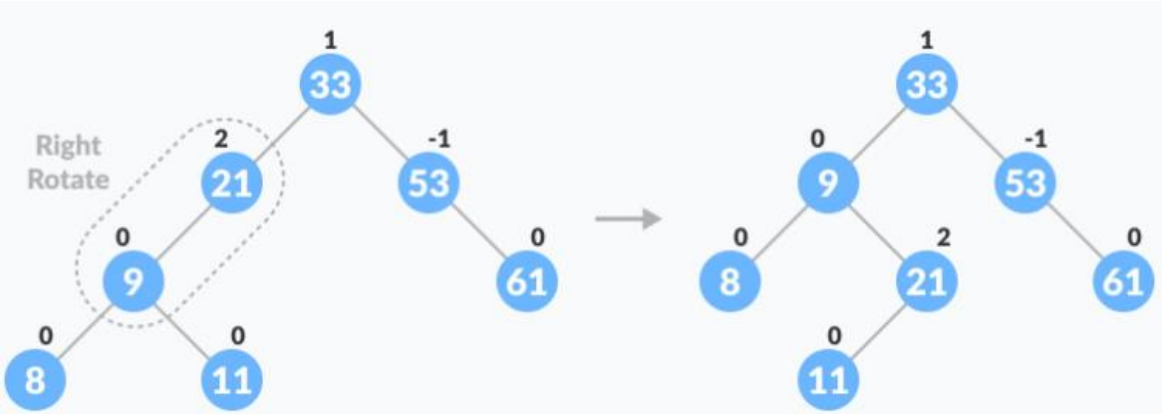
a) If balanceFactor of leftChild >= 0, do right rotation.

b) Else do left-right rotation.

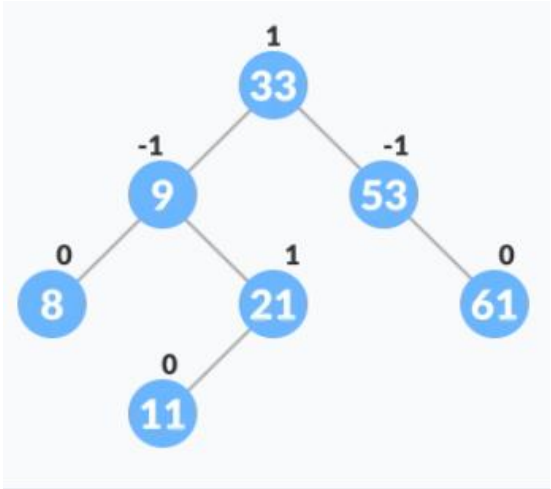
If balanceFactor of currentNode < -1,

a) If balanceFactor of rightChild <= 0, do left rotation.

b) Else do right-left rotation.



Right-rotate for balancing the tree



Avl tree final

AVL Tree: Deletion