AVL TREE

Height balanced binary search tree or AVL tree

An AVL tree is a binary search tree in which the balance factor of every node, which is defined as the difference b/w the heights of the node's left & right sub trees is either 0 or +1 or -1 .

Balance factor = height of left sub tree – height of right sub tree.

The balance factor of any node in an AVL tree is either **-1**, **0** or **+1**. If after any modification in the tree, the balance factor becomes less than -1 or greater than +1, the subtree rooted at that node is said to be unbalanced, and a rotation is needed.

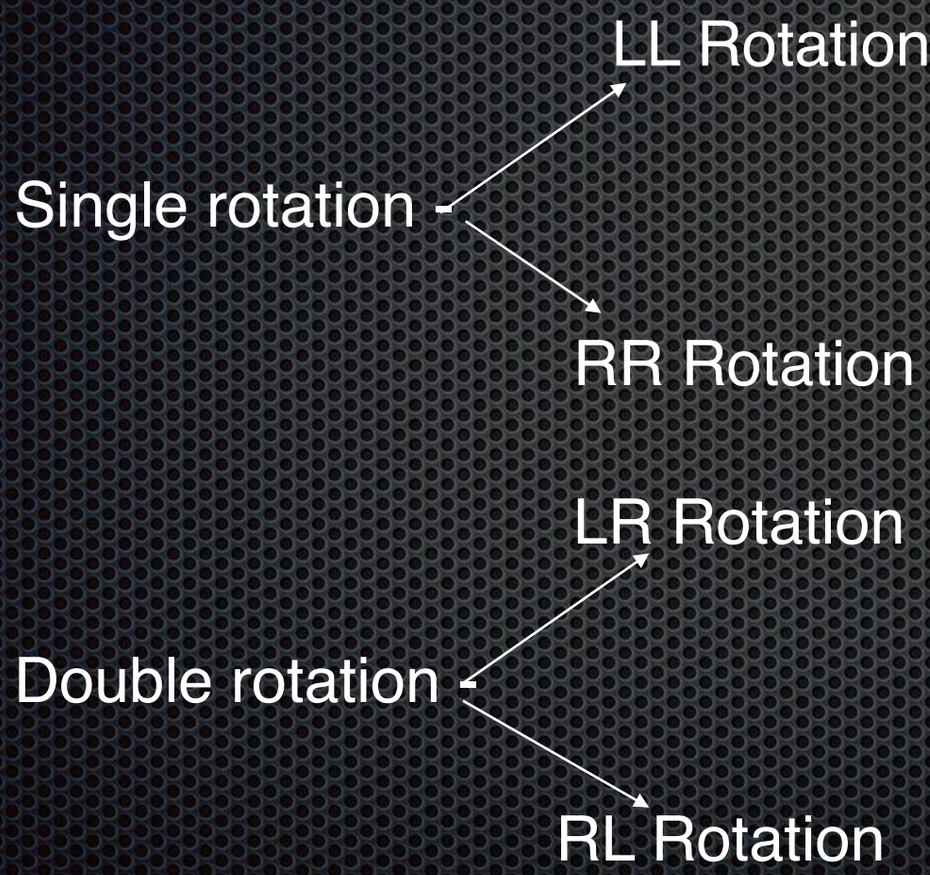
AVL tree Condition

1) Should be BST
2) Height of left subtree - height of right sub tree={-1,0,1}(known as balance factor)

AVL Tree Rotations

In AVL tree, after performing every operation like insertion and deletion we need to check the balance factor of every node in the tree. If every node satisfies the balance factor condition then we conclude the operation otherwise we must make it balanced. We use rotation operations to make the tree balanced whenever the tree is becoming imbalanced due to any operation.

Rotation operations are used to make a tree balanced.There are four rotations and they are classified into two types:



Single Left Rotation (LL Rotation)

In LL Rotation every node moves one position to left from the current position.

Single Right Rotation (RR Rotation)

In RR Rotation every node moves one position to right from the current position.

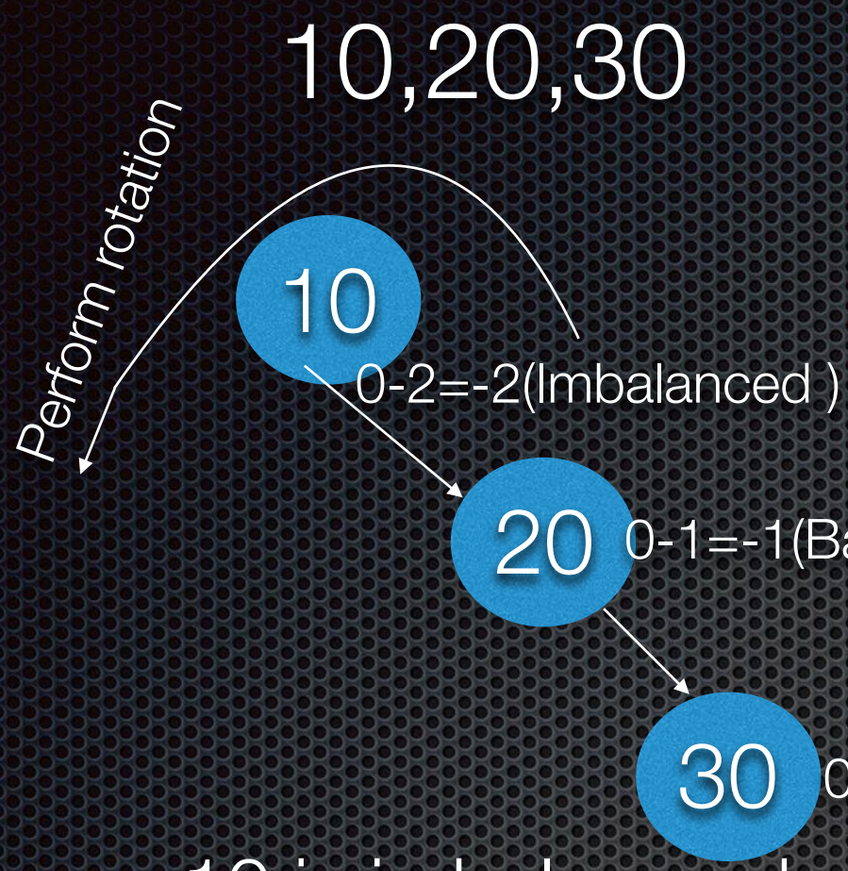
Left Right Rotation (LR Rotation)

The LR Rotation is combination of single left rotation followed by single right rotation. In LR Rotation, first every node moves one position to left then one position to right from the current position.

Right Left Rotation (RL Rotation)

The RL Rotation is combination of single right rotation followed by single left rotation. In RL Rotation, first every node moves one position to right then one position to left from the current position.

Single Left Rotation (LL Rotation)



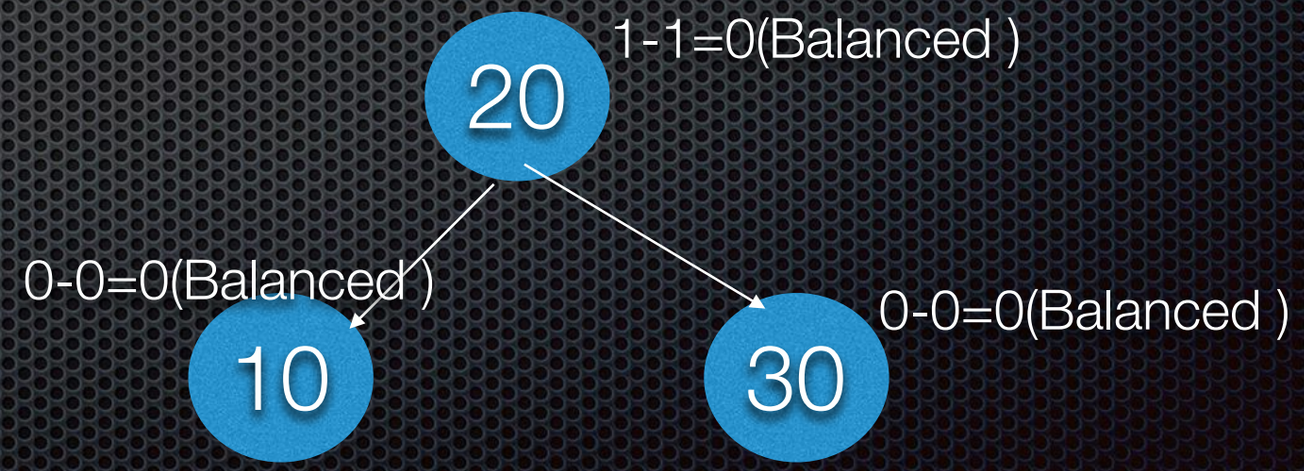
Check balance factor of each node

Height of left subtree - height of right subtree

To make balanced we use LL rotation which moves nodes one position left

10 is imbalance due to Right of Right insertion so this type of imbalance is called RR imbalance , perform left rotation to balance

After rotation →



Single Right Rotation (RR Rotation)

30,20,10

Perform rotation

Check balance factor of each node
Height of left subtree - height of right subtree

30 2-0=2(Imbalanced)

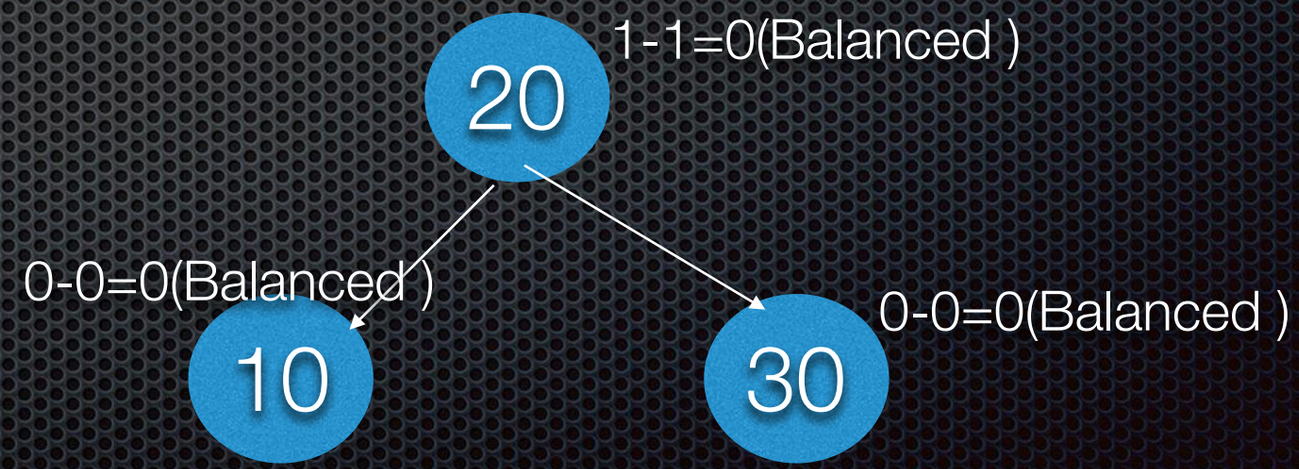
20 1-0=1(Balanced)

10 0-0=0(Balanced)

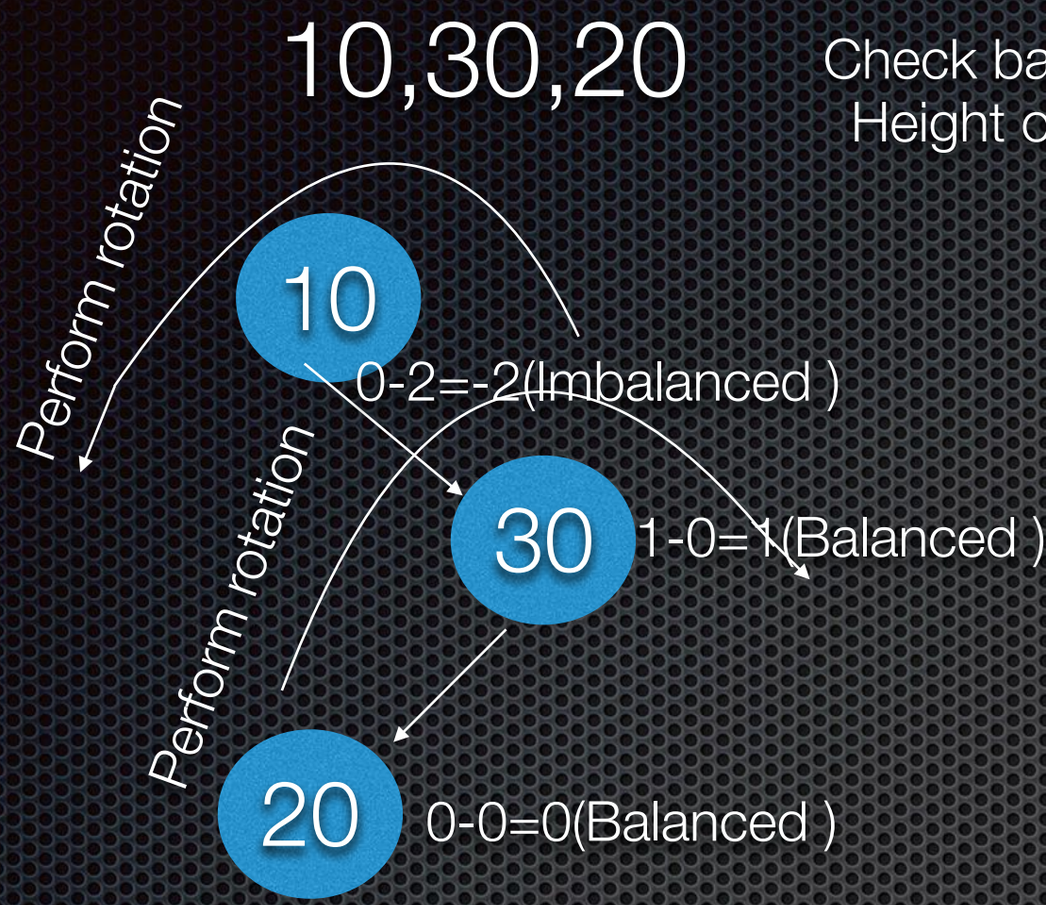
To make balanced we use RR rotation which moves nodes one position right

30 is imbalance due to left of left insertion so this type of imbalance is called LL imbalance , perform right rotation to balance

After rotation



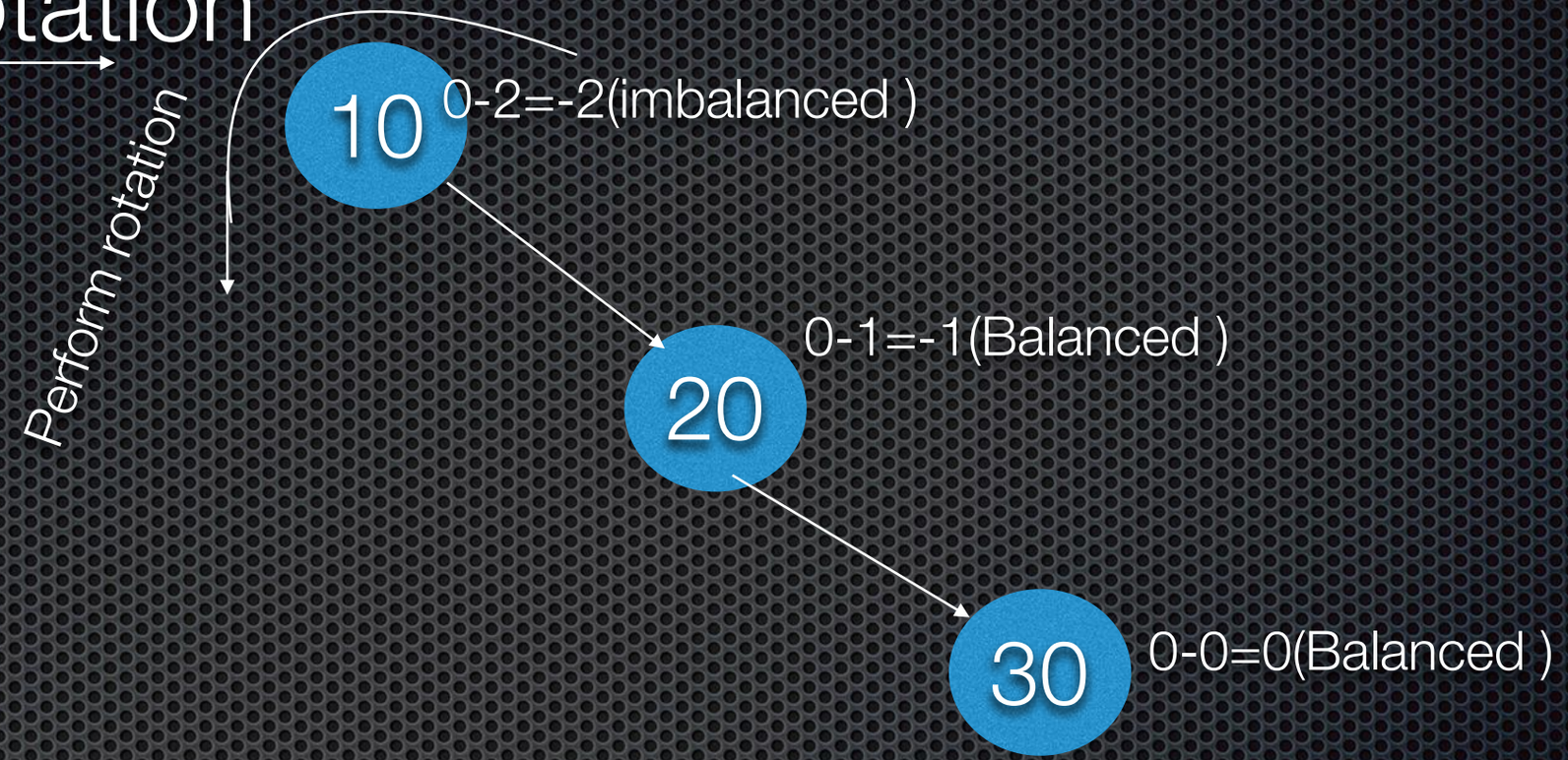
Right Left Rotation (RL Rotation)



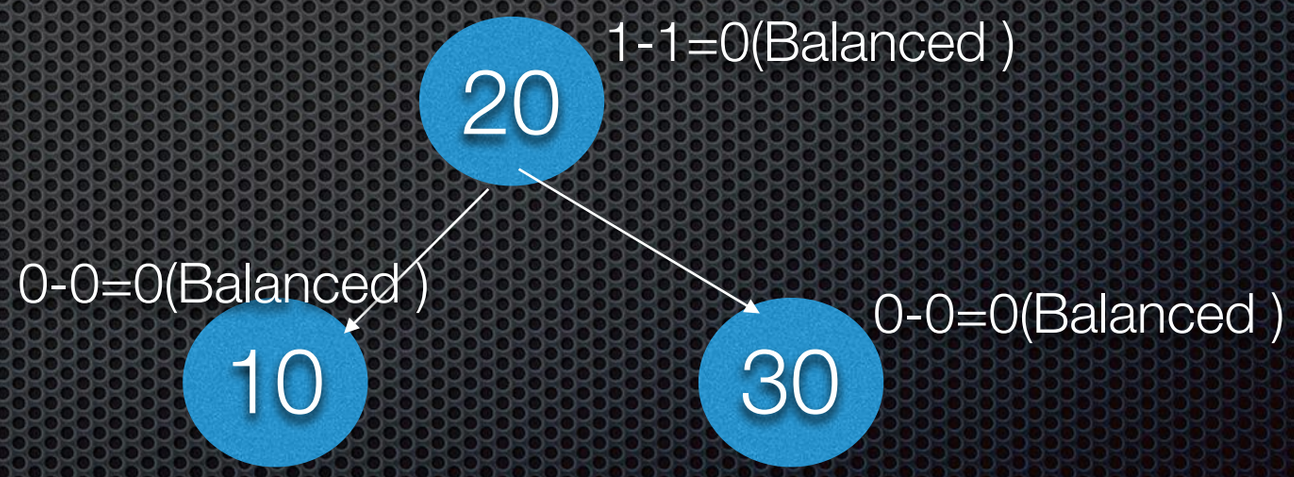
10 is imbalance due to Right and then Left insertion . RL imbalance

First perform right rotation then perform left rotation to balance

After first (right) rotation



After second(left) rotation

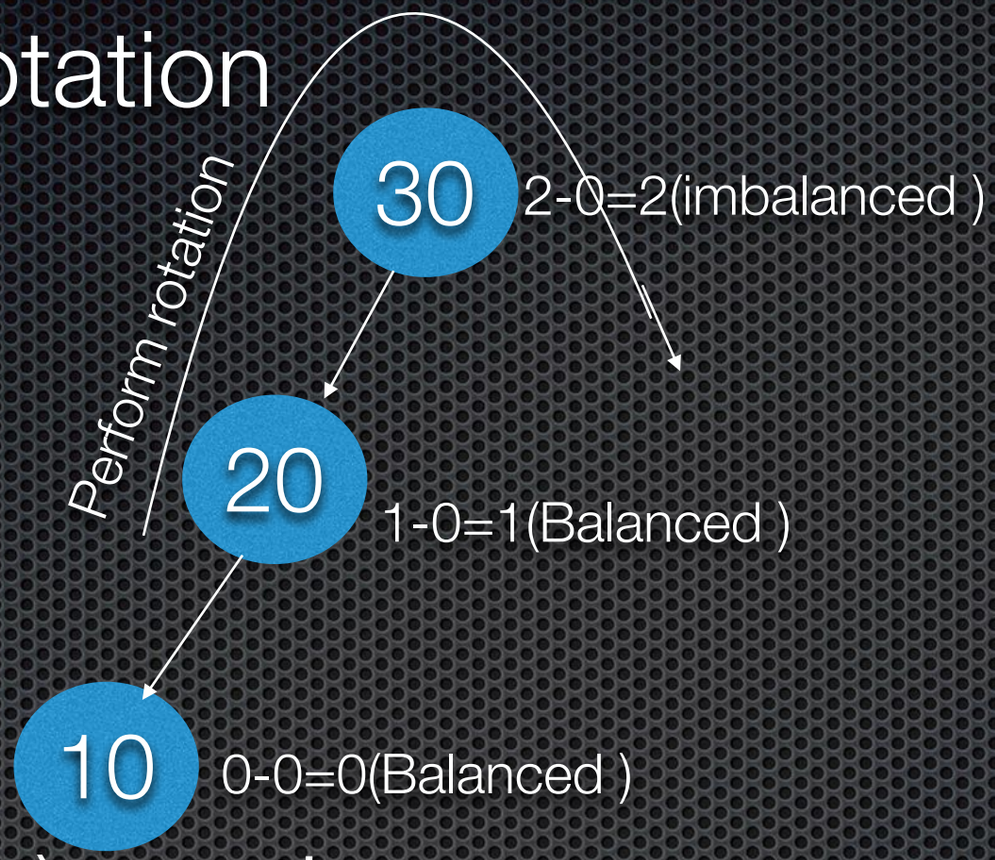


Left Right Rotation (LR Rotation)

30 is imbalance due to Left and then Right insertion . LR imbalance

First perform left rotation then perform right rotation to balance

After first (left) rotation



After second(right) rotation