Lecture

NOTE: FOR FURTHER DETAILS AND MORE COMPREHENSIVE STUDY, PLEASE SEE RECOMMENDED BOOKS OR INTERNET.

AVL Trees

An ***AVL*** tree (named after **A**delson-**V**elsky and **L**andis) is a ***self-balancing BST*** where the difference between heights of left and right sub-trees ***cannot be more than one*** for all nodes. These trees are introduced to overcome the degenerate ***drawback of BST.***
In an ***AVL*** tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property.
AVL trees are often compared with ***red–black trees*** because both support the same set of operations and take ***O(log n)*** time for the basic operations. For lookup-intensive applications, AVL trees are faster than red–black trees because they are more strictly balanced.

Balance factor

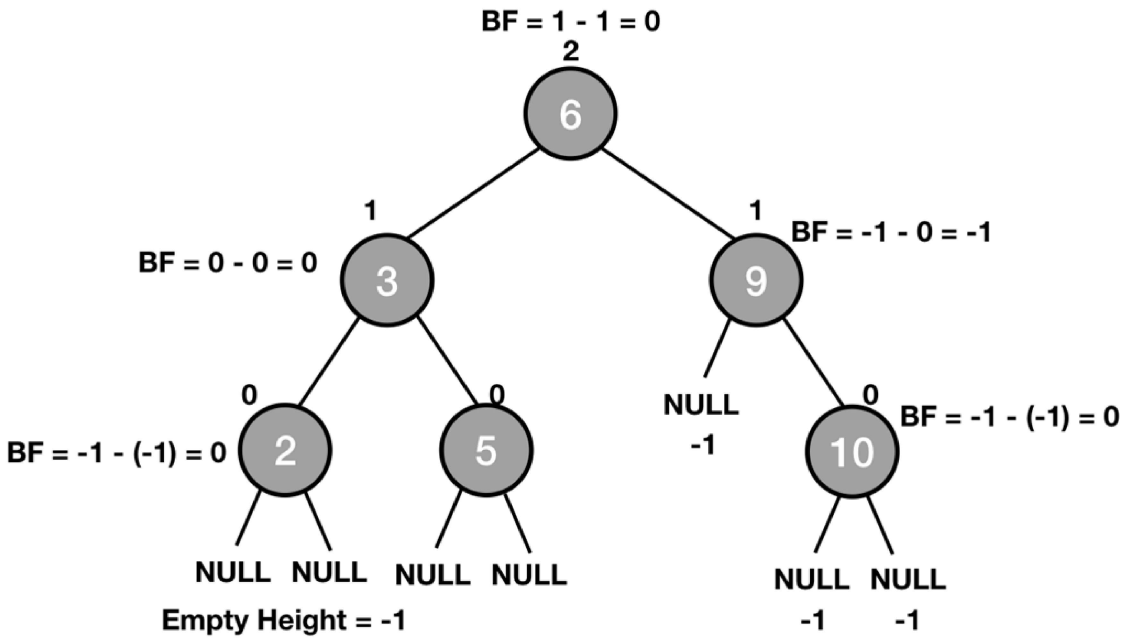
In a binary tree the balance factor of a ***node*** is defined to be the height difference between left and right sub tree.

Balance Factor (Node) = Height of Left Node – Height of Right Node

A binary tree is defined to be an AVL tree if the invariant when

Balance Factor (Node) $\in$ {-1,0,1}

This property apply on the all the nodes in a tree. A ***node*** with ***Balance Factor < 0*** is called ***"Left Heavy"***, a ***node*** with ***Balance Factor > 0*** is called ***"Right Heavy"***, and a ***node*** with ***Balance Factor = 0*** is simply called ***"Balanced"***.



AVL Rotations

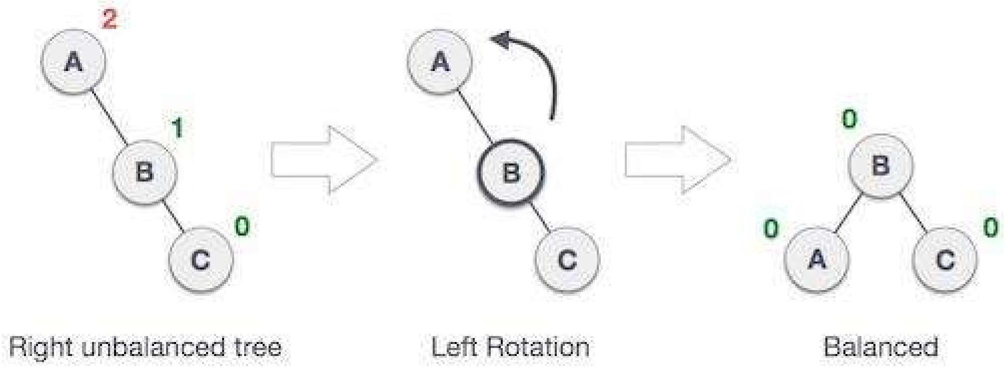
To balance itself, an AVL tree may perform the following four kinds of rotations:

- *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. To have an unbalanced tree, we at least need a tree of height 2. With this simple tree, let's understand them one by one.

Left Rotation

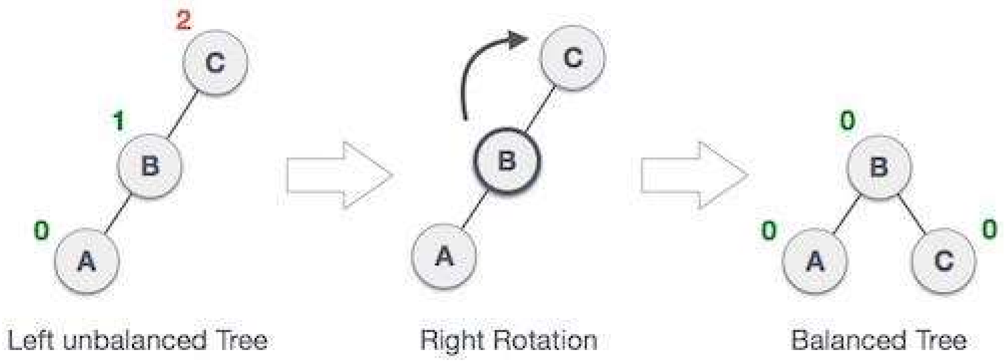
If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation:



In our example, 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***.

Right Rotation

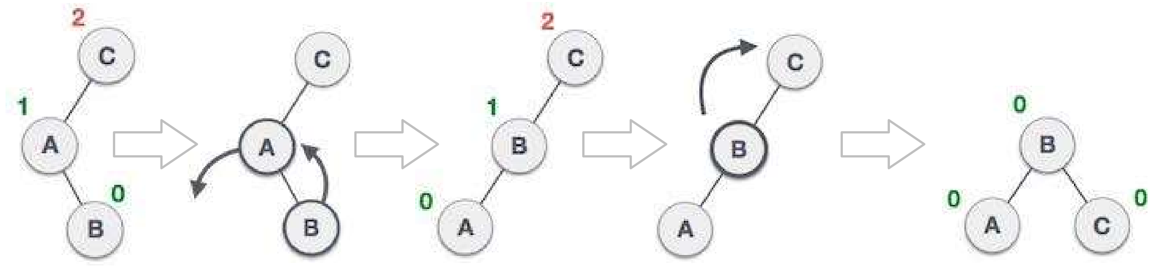
AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.



As depicted, the unbalanced node becomes the right child of its left child by performing a right rotation.

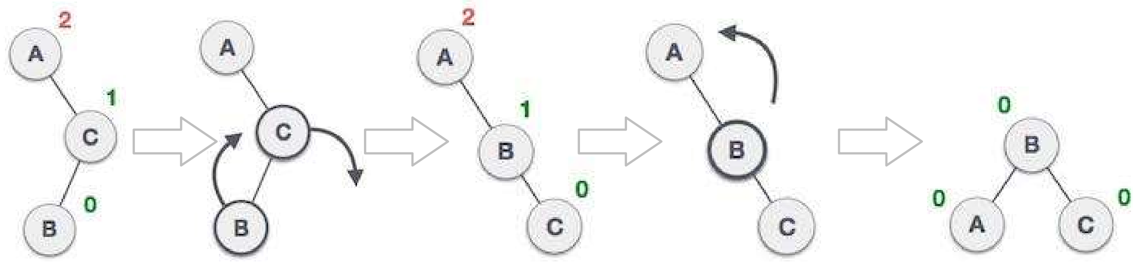
Left-Right Rotation

Double rotations are slightly complex version of already explained versions of rotations. To understand them better, we should take note of each action performed while rotation. Let's first check how to perform Left-Right rotation. A left-right rotation is a combination of left rotation followed by right rotation.



Right-Left Rotation

The second type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.


A
2
1
C
B
0
A
C
B
A
2
1
B
0
C
A
B
C
0
B
0
A
C
0