An AVL tree defined as a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees for any node cannot be more than one.
The absolute difference between the heights of the left subtree and the right subtree foar any node is known as the balance factor of the node. The balance factor for all nodes must be less than or equal to 1.
Every AVL tree is also a Binary Search Tree (Left subtree values Smaller and Right subtree values grater for every node), but every BST is not AVL Tree. For example, the second diagram below is not an AVL Tree.
The main advantage of an AVL Tree is, the time complexities of all operations (search, insert and delete, max, min, floor and ceiling) become O(Log n). This happens because height of an AVL tree is bounded by O(Log n). In case of a normal BST, the height can go up to O(n).
An AVL tree maintains its height by doing some extra work during insert and delete operations. It mainly uses rotations to maintain both BST properties and height balance.
There exist other self-balancing BSTs also like Red Black Tree. Red Black tree is more complex, but used more in practice as it is less restrictive in terms of left and right subtree height differences.
Advantages of AVL Tree:
AVL trees can self-balance themselves and therefore provides time complexity as O(Log n) for search, insert and delete.
It is a BST only (with balancing), so items can be traversed in sorted order.
Since the balancing rules are strict compared to Red Black Tree, AVL trees in general have relatively less height and hence the search is faster.
AVL tree is relatively less complex to understand and implement compared to Red Black Trees.
Disadvantages of AVL Tree:
It is difficult to implement compared to normal BST and easier compared to Red Black
Less used compared to Red-Black trees. Due to its rather strict balance, AVL trees provide complicated insertion and removal operations as more rotations are performed.
Applications of AVL Tree:
AVL Tree is used as a first example self balancing BST in teaching DSA as it is easier to understand and implement compared to Red Black
Applications, where insertions and deletions are less common but frequent data lookups along with other operations of BST like sorted traversal, floor, ceil, min and max.
Red Black tree is more commonly implemented in language libraries like map in C++, set in C++, TreeMap in Java and TreeSet in Java.
AVL Trees can be used in a real time environment where predictable and consistent performance is required.