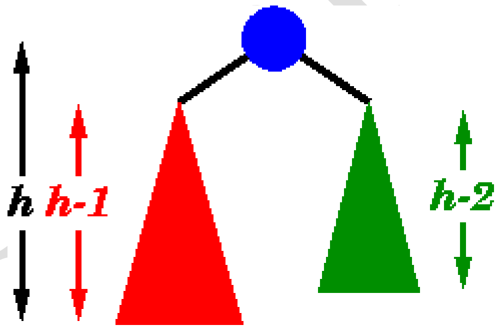
AVL Tree

An **AVL tree** is another balanced binary search tree. Named after their inventors, **A**delson-**V**elskii and **L**andis, they were the first dynamically balanced trees to be proposed. Like red-black trees, they are not perfectly balanced, but pairs of sub-trees differ in height by at most 1, maintaining an ***O*(log*n*)** search time. Addition and deletion operations also take ***O*(log*n*)** time.

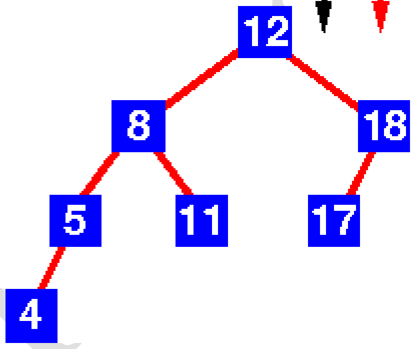
Definition of an AVL tree: An AVL tree is a binary search tree which has the following properties:

1. **The sub-trees of every node differ in height by at most one.**
2. Every sub-tree is an AVL tree.

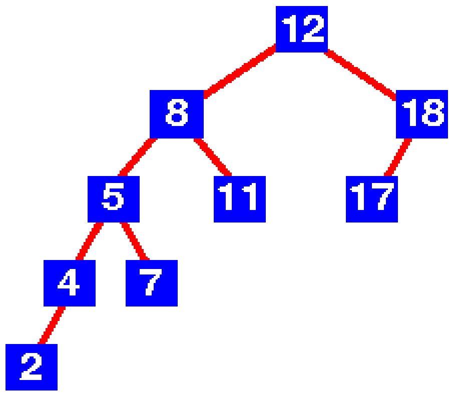
Balance requirement for an AVL tree: the left and right sub-trees differ by at most 1 in height.



For example, here are some trees:



Yes this is an AVL tree. Examination shows that *each* left sub-tree has a height 1 greater than each right sub-tree.

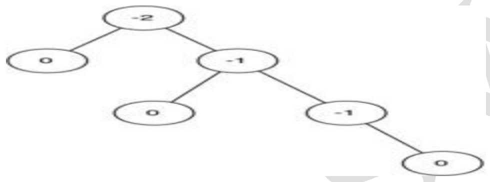


No this is not an AVL tree. Sub-tree with root 8 has height 4 and sub-tree with root 18 has height 2 for node 12.

An AVL tree implements the Map abstract data type just like a regular binary search tree, the only difference is in how the tree performs. To implement our AVL tree we need to keep track of a **balance factor** for each node in the tree. We do this by looking at the heights of the left and right subtrees for each node. More formally, we define the balance factor for a node as the difference between the height of the left subtree and the height of the right subtree.

$$balanceFactor=height(leftSubTree)-height(rightSubTree)$$

Using the definition for balance factor given above we say that a subtree is left-heavy if the balance factor is greater than zero. If the balance factor is less than zero then the subtree is right heavy. If the balance factor is zero then the tree is perfectly in balance. For purposes of implementing an AVL tree, and gaining the benefit of having a balanced tree we will define a tree to be in balance if the balance factor is -1, 0, or 1. Once the balance factor of a node in a tree is outside this range we will need to have a procedure to bring the tree back into balance. Figure shows an example of an unbalanced, right-heavy tree and the balance factors of each node.



Properties of AVL Trees

AVL trees are identical to standard binary search trees except that for every node in an AVL tree, the height of the left and right subtrees can differ by at most 1.
The following is the height differential formula:

$$|\text{Height}(T_L) - \text{Height}(T_R)| \le k$$

When storing an AVL tree, a field must be added to each node with one of three values: 1, 0, or -1. A value of 1 in this field means that the left subtree has a height one more than the right subtree. A value of -1 denotes the opposite. A value of 0 indicates that the heights of both subtrees are the same. Updates of AVL trees require up to $O(\log n)$ rotations, whereas updating red-black trees can be done using only one or two rotations (up to $O(\log n)$ color changes). For this reason, they (AVL trees) are considered a bit obsolete by some.

Sparse AVL trees

Sparse AVL trees are defined as AVL trees of height h with the fewest possible nodes. Figure 3 shows sparse AVL trees of heights 0, 1, 2, and 3.

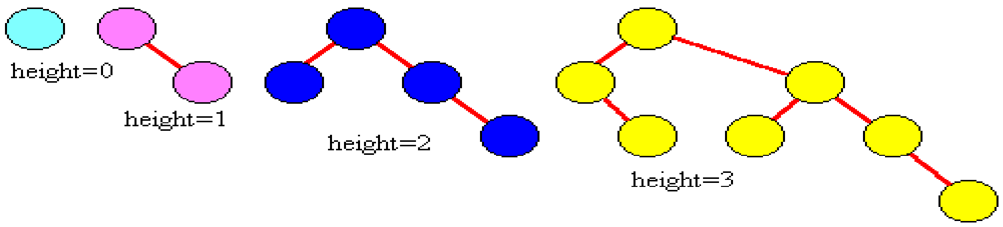
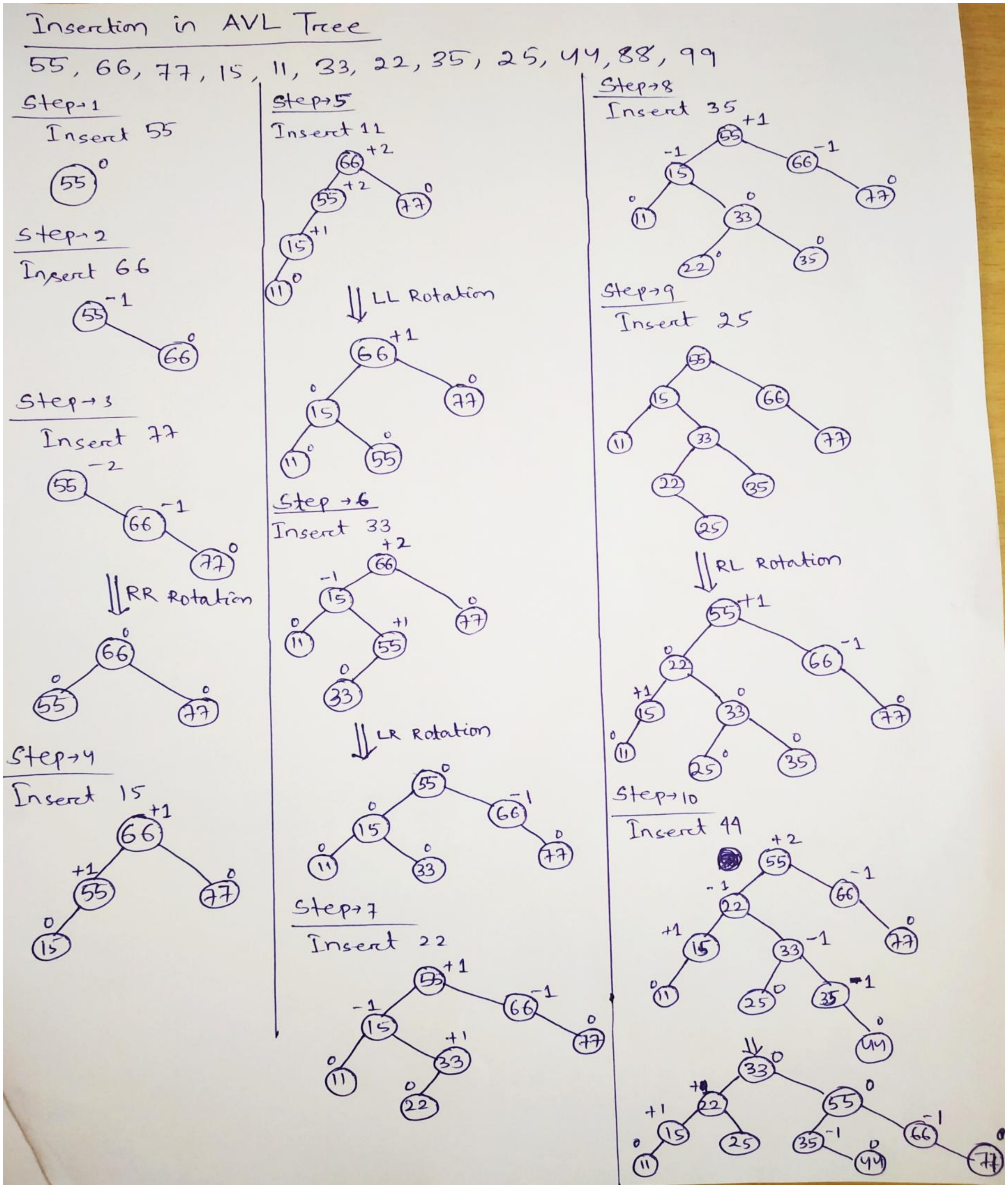


Figure Structure of an AVL tree

Insert node in AVL tree



Insertion in AVL Tree

55, 66, 77, 15, 11, 33, 22, 35, 25, 44, 88, 99

Step→1
Insert 55

Step→2
Insert 66

Step→3
Insert 77

⇓ RR Rotation

Step→4
Insert 15

Step→5
Insert 11

⇓ LL Rotation

Step→6
Insert 33

⇓ LR Rotation

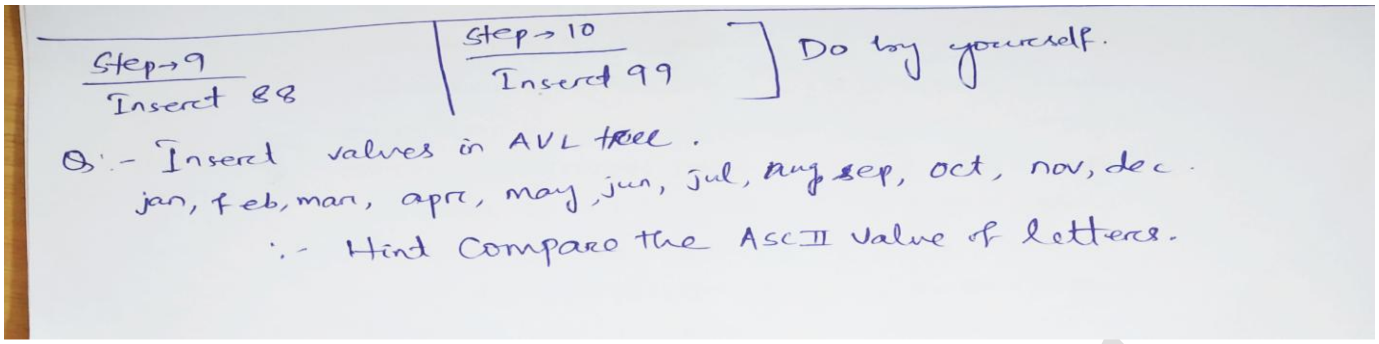
Step→7
Insert 22

Step→8
Insert 35

Step→9
Insert 25

⇓ RL Rotation

Step→10
Insert 44



Step→9
Insert 88

Step→10
Insert 99

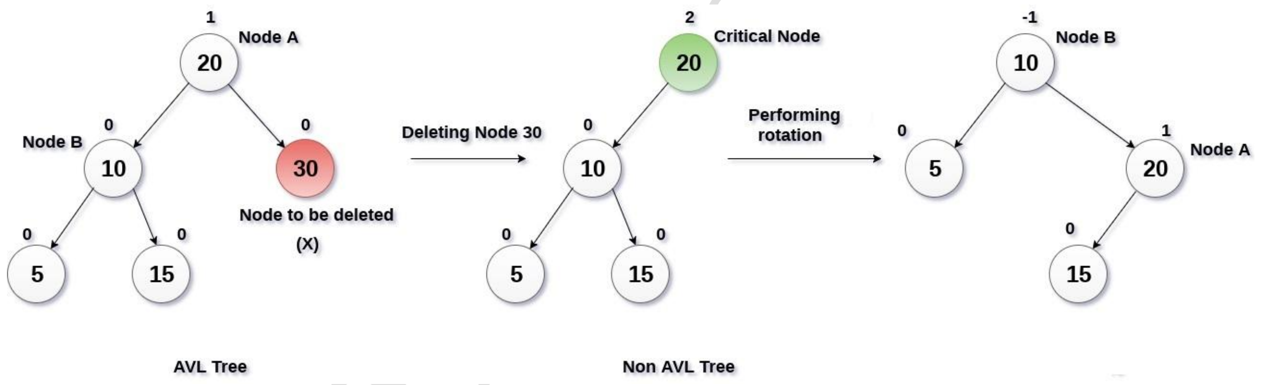
] Do by yourself.

Q:- Insert values in AVL tree.
jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec.
∴- Hint Compare the ASCII value of letters.

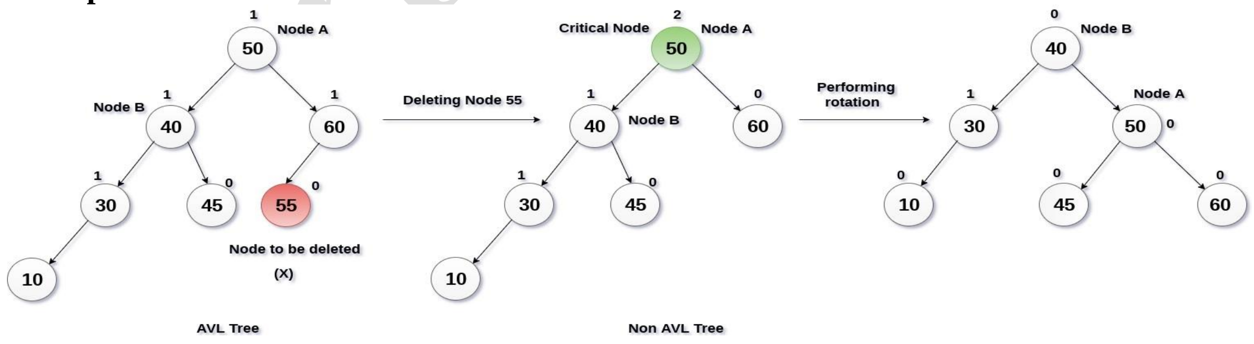
Deletion of node in AVL tree

Deleting a node from an AVL tree is similar to that in a binary search tree. Deletion may disturb the balance factor of an AVL tree and therefore the tree needs to be rebalanced in order to maintain the AVLness. For this purpose, we need to perform rotations. The two types of rotations are L rotation and R rotation. Here, we have discussed R rotations. L rotations through examples.

Example-1



Example-2



Learn the fundamentals for programming, which is just you need as a first step for your career change.

With Regards
Your Sam Sir