LAB SESSION 8: AVL (Adelson - Velsky & Landis) TREES

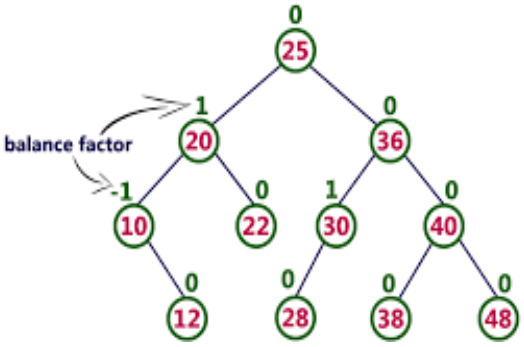
AIM: To implement AVL trees and perform the listed operations on such trees.

PROBLEM DEFINITION:

Develop C program to create an AVL Tree through a series of insertions from a predefined array of elements. Provide the following options for the said operations to be performed on the tree:

1. Insertion of a new element
2. Deletion of an existing element
3. Searching for a given element
4. Finding the width of the tree
5. Finding the max value and min value nodes
6. Listing the elements of the AVL in descending order

THEORY: AVL Tree was invented by GM Adelson - Velsky and EM Landis in 1962. The tree is named AVL in honor of its inventors.



AVL Tree can be defined as a height balanced binary search tree in which each node is associated with a balance factor which is calculated by subtracting the height of its right subtree from that of its left sub-tree.

Tree is said to be balanced if the balance factor of each node is in between -1 to 1, otherwise, the tree will be unbalanced and need to be balanced.

Balance Factor (k) = height (left(k)) - height (right(k))

If the balance factor of any node is 1, it means that the left sub-tree is one level higher than the right subtree.

Operations on AVL tree

Due to the fact that, AVL tree is also a binary search tree therefore, all the operations are performed in the same way as they are performed in a binary search tree. Searching and traversing do not lead to the violation in property of AVL tree. However, insertion and deletion are the operations which can violate this property and therefore, they need to be revisited.

1	Insertion	Insertion in AVL tree is performed in the same way as it is performed in a binary search tree. However, it may lead to violation in the AVL tree property and therefore the tree may need balancing. The tree can be balanced by applying rotations.
2	Deletion	Deletion can also be performed in the same way as it is performed in a binary search tree. Deletion may also disturb the balance of the tree therefore, various types of rotations are used to rebalance the tree.

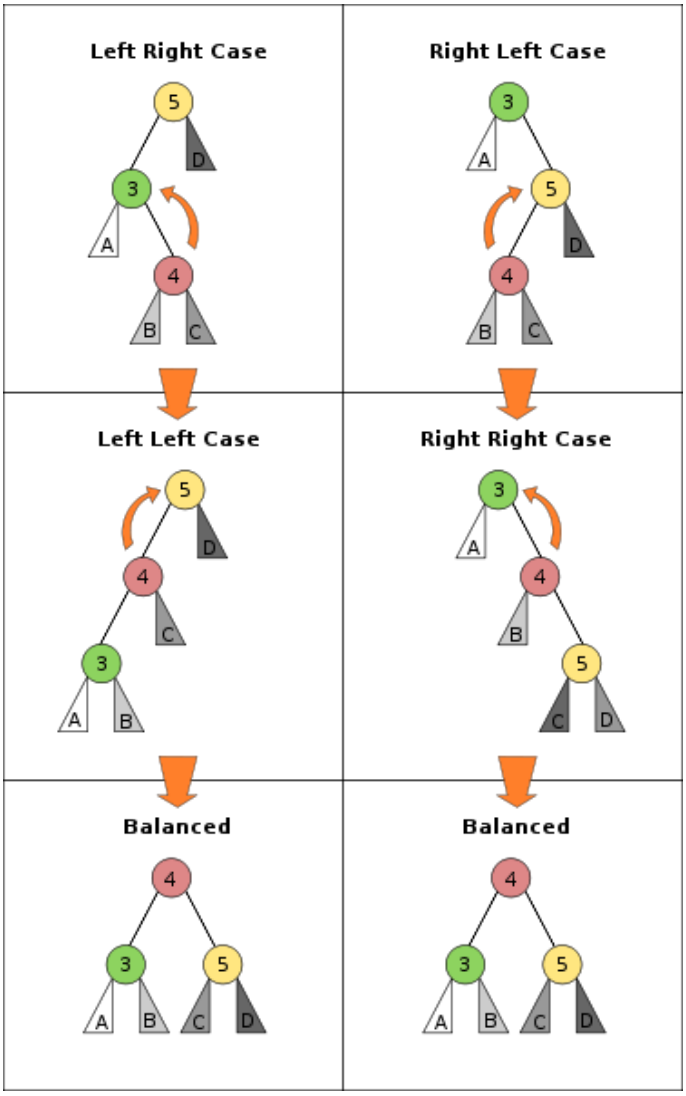
Why AVL Tree?

AVL tree controls the height of the binary search tree by not letting it to be skewed. The time taken for all operations in a binary search tree of height h is **O(h)**. However, it can be extended to **O(n)** if the BST becomes skewed (i.e. worst case). By limiting this height to log

n, AVL tree imposes an upper bound on each operation to be **O(log n)** where n is the number of nodes.

AVL Rotations

We perform rotation in the AVL tree only in case the Balance Factor is other than **-1, 0, and 1**. There are basically four types of rotations which are as follows:



1. L L rotation: Inserted node is in the left subtree of left subtree of A
2. R R rotation : Inserted node is in the right subtree of right subtree of A
3. L R rotation : Inserted node is in the right subtree of left subtree of A
4. R L rotation : Inserted node is in the left subtree of right subtree of A

Where node A is the node whose balance Factor is other than -1, 0, 1.

ALGORITHM AND DEMONSTRATION:

1. **Insertion in an AVL Tree**
2. **Deletion from an AVL Tree**