. AVL Tree - By Mani Abedii

An **AVL Tree** is a type of self-balancing binary search tree. It is named after its inventors, **Adelson-Velsky and Landis**, who introduced it in 1962.

In an AVL tree, the balance of the tree is maintained by ensuring that the **height difference** (or **balance factor**) between the left and right subtrees of every node is at most 1.

The height of a binary tree determines the time complexity of its operations. A completely balanced tree has a height of O(log n), where n is the number of nodes. If the height difference between subtrees (the balance factor) is allowed to grow beyond 1, the tree can become **skewed**, resembling a linked list in the worst case. This would degrade the time complexity of operations to O(n).

Maintaining a strict balance factor means that search, insertion, and deletion always take O(log n) time because the height of the tree never exceeds log(n) for n nodes.

Characteristics of an AVL Tree:

- **Binary Search Tree Property:**
  All the nodes in the left subtree < Parent
  All the nodes in the right subtree > Parent
- **Balance Factor:**
  For each node in the tree:
  Balance Factor = Height of Left Subtree – Height of Right Subtree
  Balance factor must be **1, -1 or 0** for the tree to remain balanced.
- **Rotations:**
  To maintain balance, the tree performs **rotations** when a node becomes unbalanced.

  1. **Right Rotation:**
     By performing a right rotation on a node:
     - The left child of the node will become the new root of the subtree.
     - The original root will become the right child of the new root.
     - Existing right subtree of the node -that has become the new root- (if exists), will become the left subtree of the previous root (which is now the right child of the root).

  2. **Left Rotation:**
     By performing a left rotation on a node:
     - The right child of the node will become the new root of the subtree.
     - The original root will become the left child of the new root.
     - Existing left subtree of the node -that has become the new root- (if exists), will become the right subtree of the previous root (which is now the left child of the root).

- **Imbalances**

  There are 4 types of imbalances in an AVL Tree:

  1. **Left-Left Case (LL Case):**
     This occurs when a node becomes unbalanced due to an insertion in the left subtree of the left child. Or a deletion in the right subtree of the left child.

     - **Fix:** Perform a single **right rotation**.

  2. **Right-Right Case (RR Case):**
     This occurs when a node becomes unbalanced due to an insertion in the right subtree of the right child. Or a deletion in the left subtree of the right child.

     - **Fix:** Perform a single **left rotation**.

  3. **Left-Right Rotation (LR):**
     This occurs when a node becomes unbalanced due to an insertion in the right subtree of the left child. Or a deletion in the left subtree of the left child.

     - **Fix:** Perform a left rotation on the left child, then a right rotation on the node.

  4. **Right-Left Rotation (RL):**
     This occurs when a node becomes unbalanced due to an insertion in the left subtree of the right child. Or a deletion in the right subtree of the right child.

     - **Fix:** Perform a right rotation on the right child, then a left rotation on the node.