AVL Tree Algorithm Overview

An AVL tree is a self-balancing binary search tree where the height difference between the left and right subtrees of any node is at most one. This balance ensures that the tree remains approximately balanced, leading to $O(\log n)$ time complexity for search, insertion, and deletion operations.

Key Concepts

- **Height**: The height of a node in an AVL tree is the length of the longest path from the node to a leaf. This height is used to determine the balance of the node.
- **Balance Factor**: For any node in the AVL tree, the balance factor is calculated as the height of the left subtree minus the height of the right subtree. It helps in determining if the node is balanced, left-heavy, or right-heavy.
- **Rotations**: To maintain balance in the AVL tree after insertions and deletions, rotations are performed. There are four types of rotations:
  - **Right Rotation (Single Rotation)**: Applied when a left-heavy subtree needs balancing.
  - **Left Rotation (Single Rotation)**: Applied when a right-heavy subtree needs balancing.
  - **Left-Right Rotation (Double Rotation)**: Applied when a left subtree has a right-heavy child.
  - **Right-Left Rotation (Double Rotation)**: Applied when a right subtree has a left-heavy child.

Operations

- **Insertion**:
  - Insert the new node following the standard binary search tree (BST) insertion rules.
  - Update the height of each ancestor node.
  - Calculate the balance factor of each node.
  - Perform rotations if any node becomes unbalanced (balance factor is greater than 1 or less than -1).
- **Rotations**:
  - **Right Rotation**: Used when a left-heavy subtree (balance factor > 1) is unbalanced.
  - **Left Rotation**: Used when a right-heavy subtree (balance factor < -1) is unbalanced.
  - **Left-Right Rotation**: First perform a left rotation on the left child, then a right rotation on the current node.
  - **Right-Left Rotation**: First perform a right rotation on the right child, then a left rotation on the current node.