AVL Trees: Properties, Imbalance Cases, and Rotations

Introduction

An AVL Tree is a type of self-balancing binary search tree named after its inventors Adelson-Velsky and Landis. In an AVL tree, the difference in heights between the left and right subtrees (called the balance factor) is at most 1 for all nodes. This balance ensures O(log n) time complexity for search, insertion, and deletion operations.

Balance Factor

The balance factor of a node is defined as:
balance_factor = height(left_subtree) - height(right_subtree)

A node is:
- Balanced if its balance factor is -1, 0, or 1
- Unbalanced if the balance factor is less than -1 or greater than 1

Imbalance Cases

There are four types of imbalances in an AVL tree:

1. Left-Left (LL) Case:
- Insertion in the left subtree of the left child
- Solution: Single right rotation

2. Right-Right (RR) Case:
- Insertion in the right subtree of the right child
- Solution: Single left rotation

3. Left-Right (LR) Case:
- Insertion in the right subtree of the left child
- Solution: Left rotation on left child, then right rotation on current node

4. Right-Left (RL) Case:
- Insertion in the left subtree of the right child
- Solution: Right rotation on right child, then left rotation on current node

Rotations

1. Right Rotation:
- Used to fix LL imbalance
- Steps:
  a. Let y be the unbalanced node
  b. Let x be y's left child
  c. Perform rotation so that x becomes the new root of the subtree

2. Left Rotation:
- Used to fix RR imbalance

- Similar to right rotation, but in the opposite direction

3. Left-Right Rotation:
- Used to fix LR imbalance
- First perform a left rotation on the left child, then a right rotation on the unbalanced node

4. Right-Left Rotation:
- Used to fix RL imbalance
- First perform a right rotation on the right child, then a left rotation on the unbalanced node

Properties of AVL Trees

- Always balanced to maintain O(log n) height
- Insertion and deletion require rebalancing through rotations
- Provides faster lookups than unbalanced BSTs
- Height of an AVL tree with n nodes is O(log n)
- Suitable for applications where search operations are more frequent than insertions/deletions