AVL Trees: Balancing, Rotations, and Example

An **AVL tree** is a self-balancing binary search tree (BST) that maintains its balance by ensuring the height difference (balance factor) between the left and right subtrees of any node does not exceed one. This property guarantees efficient operations such as insertion, deletion, and lookup, each with a time complexity of **O(log n).**

Imbalance Cases and Rotations

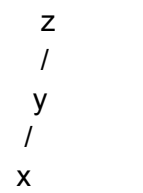
To preserve balance during insertions and deletions, AVL trees utilize **rotations** to rectify four primary imbalance scenarios:

1. Left-Left (LL) Case

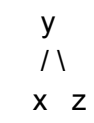
Occurs when a node becomes unbalanced due to an insertion in the left subtree of its left child. This imbalance is corrected by performing a **right rotation** on the unbalanced node.

Example:

Before Rotation:



After Right Rotation at 'z':

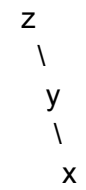


2. Right-Right (RR) Case

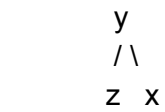
Occurs when a node becomes unbalanced due to an insertion in the right subtree of its right child. This imbalance is corrected by performing a **left rotation** on the unbalanced node.

Example:

Before Rotation:



After Left Rotation at 'z':

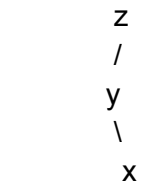


3. Left-Right (LR) Case

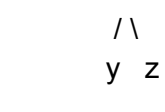
Occurs when a node becomes unbalanced due to an insertion in the right subtree of its left child. This imbalance is corrected by performing a **left rotation** on the left child, followed by a **right rotation** on the unbalanced node.

Example:

Before Rotations:



After Left Rotation at 'y' and Right Rotation at 'z':

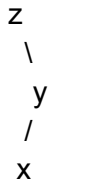


4. Right-Left (RL) Case

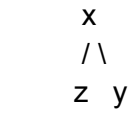
Occurs when a node becomes unbalanced due to an insertion in the left subtree of its right child. This imbalance is corrected by performing a **right rotation** on the right child, followed by a **left rotation** on the unbalanced node.

Example:

Before Rotations:



After Right Rotation at 'y' and Left Rotation at 'z':

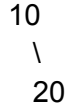


Insertion Example

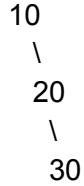
Consider inserting the following sequence of numbers into an empty AVL tree: **10, 20, 30, 40, 50, 25.**

1. **Insert 10**:

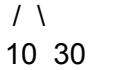
2. **Insert 20**:



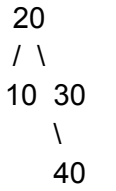
3. **Insert 30**:



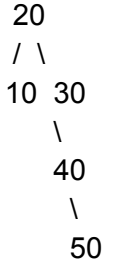
This forms an **RR case** at node 10. Perform a **left rotation** at 10:



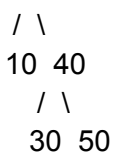
4. **Insert 40**:



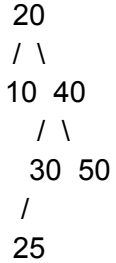
5. **Insert 50**:



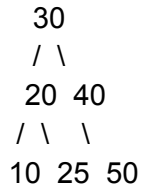
This forms an **RR case** at node 30. Perform a **left rotation** at 30:



6. **Insert 25**:



This forms an **LR case** at node 40. Perform a **right rotation** at 40, followed by a **left rotation** at 20:



The tree is now balanced, adhering to AVL properties.

For a visual explanation of AVL tree insertions and rotations, you might find this video helpful:
AVL Tree Insertion Example