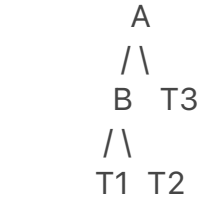
STARTING TREE for LL and LR

Original Tree Structure:

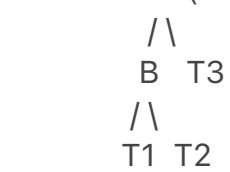


Before an insertion, height(A.left) and height(A.right) differ by 1. Thus, this tree is AVL-balanced initially. AVL trees maintain balance by ensuring the heights of two child subtrees of any node differ by no more than one.

———

Case 1: Left-Left Imbalance (Single Rotation)

Imbalance situation (after inserting into subtree T1):

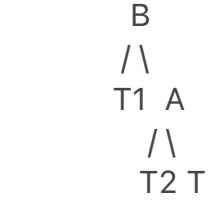


In this scenario, after insertion into subtree T1, node A becomes imbalanced, specifically due to a left-heavy subtree (B → T1). This is referred to as an **LL imbalance** (Left-Left).

Resolving LL Imbalance:
A **single right rotation** around the imbalanced node (A) is required to correct this imbalance:

- The left child B becomes the new root of this subtree.
- Node A becomes the right child of node B.
- Subtrees T2 and T3 are attached appropriately.

Result after rotation:



After rotation, node B now has the same height that A previously had before rotation. This restores AVL balance.

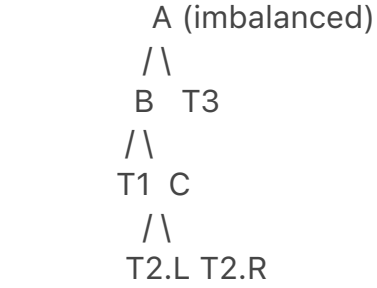
Summary of LL rotation (pseudo code):

```
fun rotateRight(oldRoot, parentOld):
  newRoot = oldRoot.left
  oldRoot.left = newRoot.right
  newRoot.right = oldRoot
  updateHeight(oldRoot)
  updateHeight(newRoot)
```

———

Case 2: Left-Right Imbalance (Double Rotation)

Original imbalance scenario (after insertion into subtree T2):

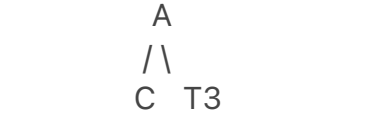


In this scenario, insertion into subtree T2 makes the node C the focal point of imbalance. Here we have a **Left-Right imbalance** (B → C), meaning the imbalance goes first left from the root node (A) and then right (C) from node (B).

A single rotation does not resolve this imbalance effectively because node C isn't directly beneath node B on one side; it is diagonal (B left child and then C right child). Hence, a **double rotation** is required.

Steps of LR double rotation:

Step 1: Rotate left around node B with its right child C:



```
    / \
   B  T2.R
  / \
 T1  T2.L
```

Step 2: Rotate right around node A with its new left child C:

```
      C
    /   \
   B     A
  / \   / \
 T1 T2.L T2.R T3
```

Now, the AVL property is restored.

Summary of LR double rotation (pseudo code):

```
fun doubleRotateLeftRight(oldRoot, parentOld):
  rotateLeft(oldRoot.left, parentOld)
  rotateRight(oldRoot, parentOld)
```

———

STARTING TREE for RR and RL

Original Tree Structure:

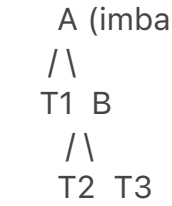
```
    A
   / \
  T1  B
     / \
    T2  T3
```

Again, initially, height(A.left) and height(A.right) differ by at most one. Tree is AVL-balanced at this stage.

———

Case 3: Right-Right Imbalance (Single Rotation)

Imbalance situation (after inserting into subtree T3):



In this case, insertion in T3 makes the right subtree too deep (Right-Right imbalance), creating imbalance at node A. This is similar to the LL case but mirrored on the right side.

Resolving RR imbalance:
Perform a **single left rotation** around node A:

- The right child (B) becomes the new subtree root.
- Node A moves to become left child of node B.
- Subtrees T1 and T2 are repositioned correctly.

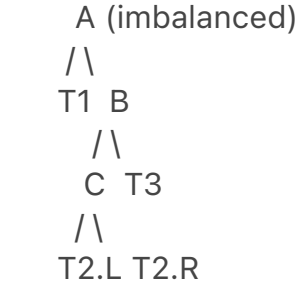
Result after rotation:

This restores AVL balance.

———

Case 4: Right-Left Imbalance (Double Rotation)

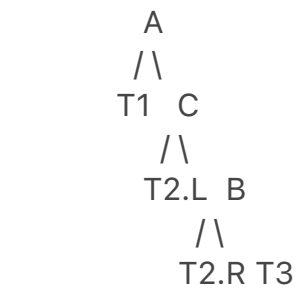
Original imbalance scenario (after inserting into subtree T2):



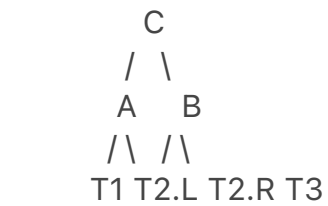
This is the **Right-Left imbalance**, similar to the Left-Right imbalance, but mirrored on the right side.

Steps of RL double rotation:

Step 1: Rotate right around node B with its left child C:



Step 2: Rotate left around node A with its new right child C:



AVL property is now restored after the double rotation.

Summary of RL double rotation (pseudo code):

```
fun doubleRotateRightLeft(oldRoot, parentOld):
  rotateRight(oldRoot.right, parentOld)
  rotateLeft(oldRoot, parentOld)
```

———

Summary & Intuition of AVL Rotations:

- **Single Rotation (LL or RR)**:
  - Used when insertion happens on the “outside” subtree (left-left or right-right).
  - Single rotations shift the imbalanced node down, moving the child node up to root the subtree.
- **Double Rotation (LR or RL)**:
  - Used when insertion happens on the “inside” subtree (left-right or right-left).
  - Requires two rotations:
    - First rotation moves the child subtree into a position suitable

for the second rotation.

- Second rotation restores AVL balance by shifting the median node up to subtree root position.

This detailed expansion helps solidify understanding of AVL tree rotations, clearly differentiating the situations when single and double rotations are needed and illustrating clearly how each rotation corrects the imbalance.