Data Structures

Dilip Kumar Maripuri
Computer Applications

Data Structures

Session : AVL Trees

Dilip Kumar Maripuri
Computer Applications

Balanced Trees

- A **Balanced Search Tree** is a type of **Binary Search Tree (BST)** that ensures its height remains **logarithmic** with respect to the number of nodes.
- This guarantees **efficient searching, insertion, and deletion operations** with a time complexity of $\mathcal{O}(\log n)$, unlike unbalanced BSTs, which may degrade into linked lists with $\mathcal{O}(n)$ complexity.

- **Why Balanced Trees?**
  - **Efficiency in Searching**: If a BST is unbalanced, its height may reach $\mathcal{O}(n)$, making search operations inefficient. Balanced trees maintain a height of $\mathcal{O}(\log n)$.
  - **Performance Optimization**: Balanced trees are widely used in **databases, file systems, and memory indexing** because of their ability to keep operations fast.
  - **Preventing Skewness**: When data is inserted in **sorted order**, a BST can become skewed, resembling a linked list. Balancing techniques **prevent this from happening**.

- **Characteristics of Balanced Trees**
  - **Height Constraint**: The tree height is kept proportional to $\log(n)$.
  - **Balance Factor**: The absolute height difference between left and right subtrees is bounded.
  - **Self-Balancing Property**: The tree structure is automatically adjusted after insertions or deletions.

Types of Balanced Trees

- **AVL Trees**
  - Named after **Adelson-Velsky and Landis** (1962).
  - The **balance factor** is defined as:

$$\text{Balance Factor} = \text{Height of Left Subtree} - \text{Height of Right Subtree}$$

  - Balance factor must be $-1, 0, +1$ for every node.
  - If violated, **rotations (LL, RR, LR, RL)** restore balance.

Types of Balanced Trees

- **Red-Black Trees**
  - A BST where each node is assigned a **red** or **black** color.
  - Ensures balance using **coloring rules**:
    1. Root is always **black**.
    2. No two consecutive **red nodes**.
    3. Every path from root to a leaf has the **same number of black nodes**.
  - Used in **C++ STL (map, set, multimap, multiset)** and **Linux kernel**.

Types of Balanced Trees

- **Multiway Trees (2-3 trees, 2-3-4 trees, B-trees)**
  - These are types of balanced search trees that allow more than one element in a node, falling under the representation-change variety of the strategy.
  - **In a 2-3 tree**, every internal node may contain either one or two values and have either two or three children, respectively.
  - **In a 2-3-4 tree**, every internal node can have between two and four children and contain one to three elements.
  - **B-trees** generalize these concepts, allowing for more than two children per node and perfect balance across the tree.

Tree Rotations

- A critical operation in maintaining balance in certain types of self-balancing binary trees (such as AVL trees and red-black trees) is the tree rotation.
- A rotation is a local operation in a search tree that preserves the in-order ordering of the elements.
- If an insertion or deletion violates the tree's balance properties, rotations are used to restore balance.
  - **Left Rotation (LL)**
  - **Right Rotation (RR)**
  - **Left-Right Rotation (LR)**
  - **Right-Left Rotation (RL)**

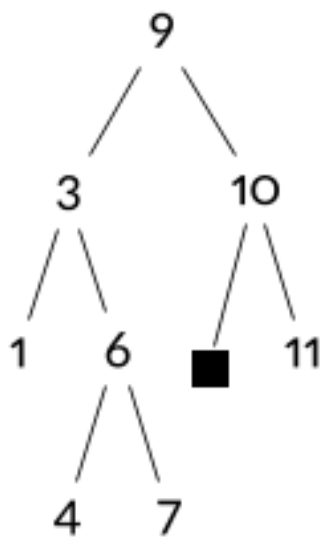
Types of Balanced Trees

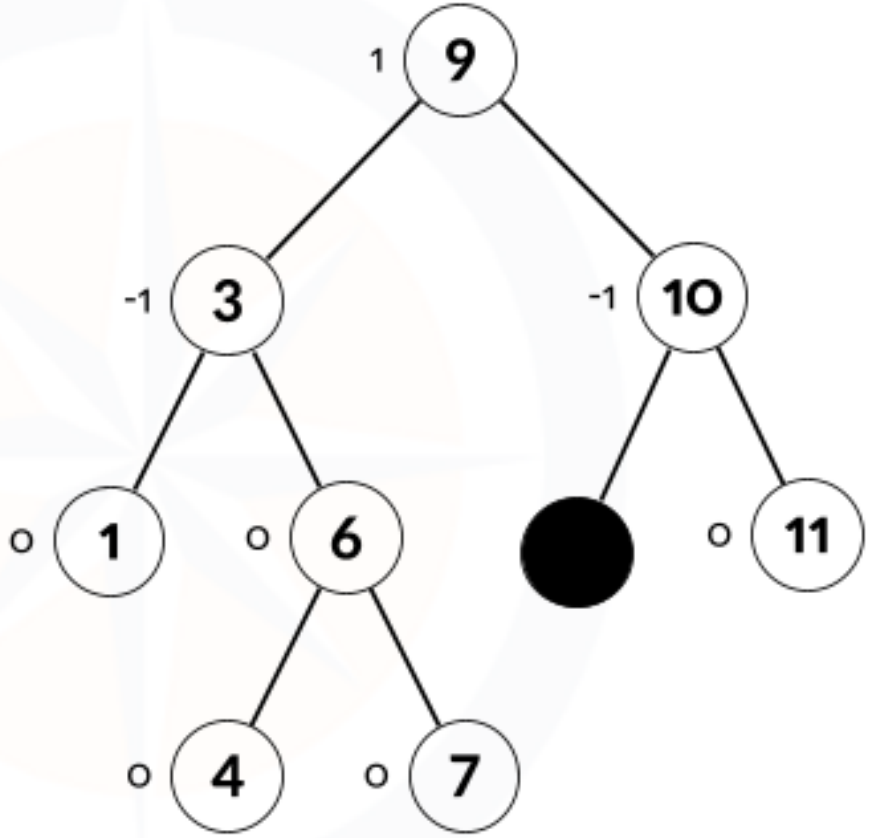
- Balanced trees are essential for ensuring **efficient** and **optimized** operations in various applications like databases, indexing, and memory management.
- By maintaining balance in these trees, the maximum number of comparisons for search operations is minimized, which is crucial for many applications, particularly those involving large datasets.

AVL Trees

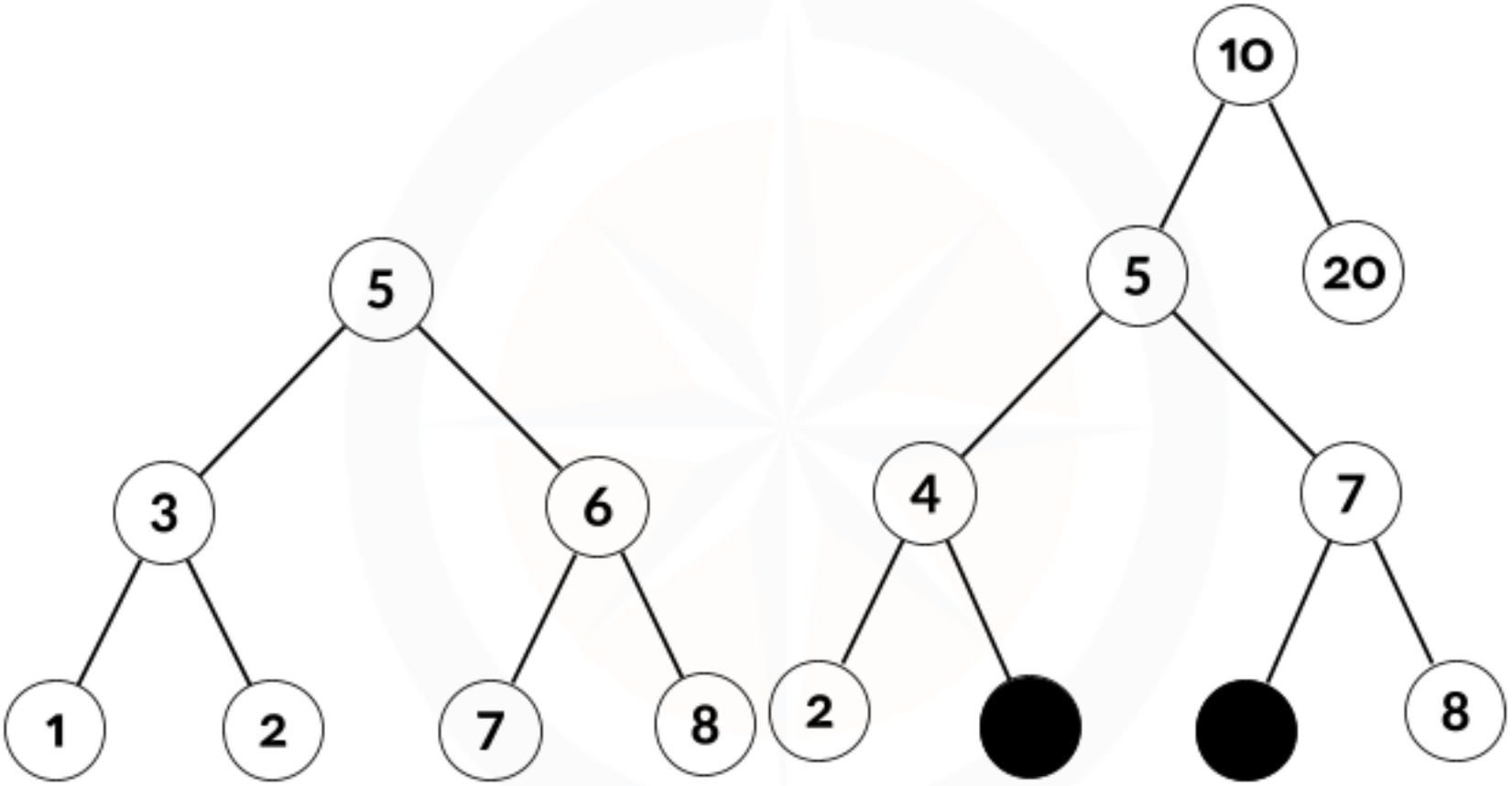
- An AVL tree is a binary search tree in which the balance factor of every node, which is defined as the difference between the heights of the node's left and right subtrees, is either 0 or +1 or −1.
- The height of the empty tree is defined as −1.
- The balance factor can also be computed as the difference between the numbers of levels rather than the height difference of the node's left and right subtrees.

Computing Balance Factors





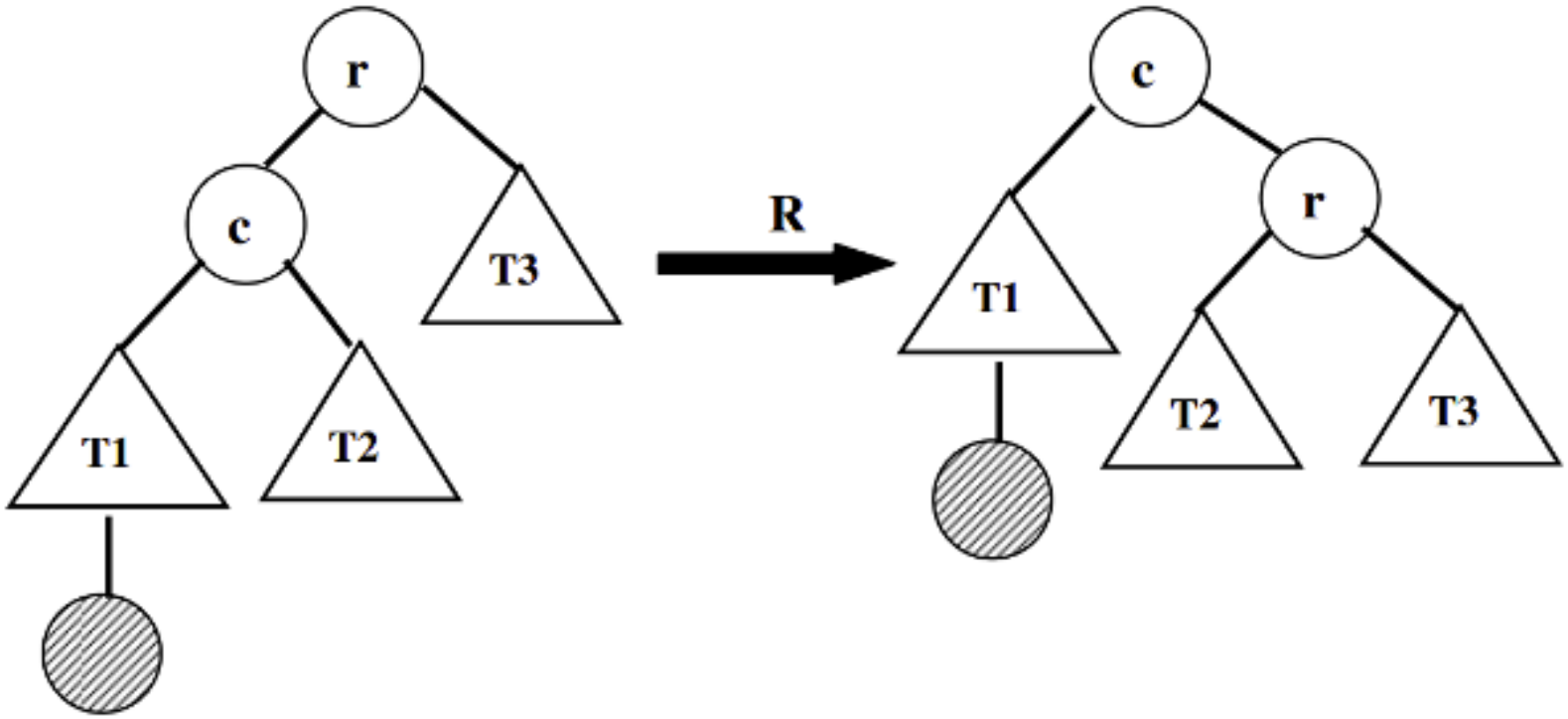
Computing Balance Factors - Exercise



AVL Trees - Rotations

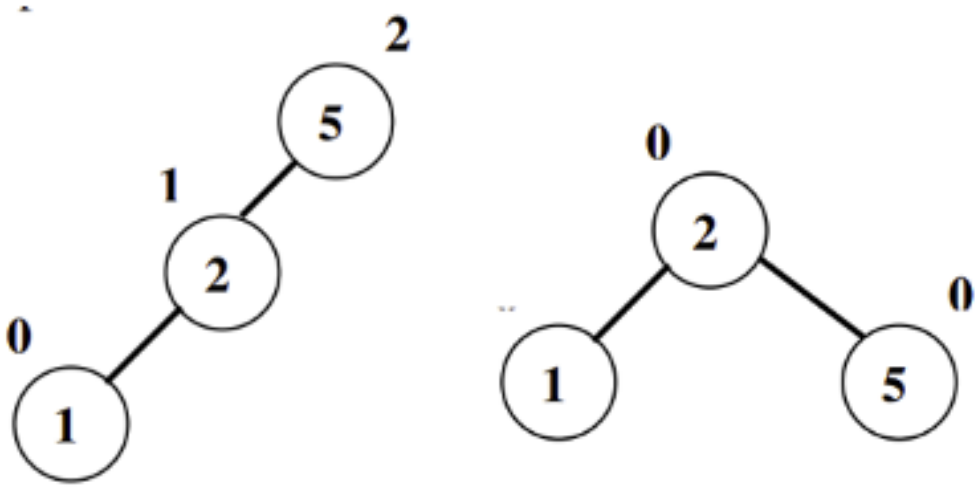
- If an insertion of a new node makes an AVL tree unbalanced, we transform the tree by a rotation.
- A rotation in an AVL tree is a local transformation of its sub-tree rooted at a node whose balance has become either +2 or -2.
- If there are several such nodes, we rotate the tree rooted at the unbalanced node that is the closest to the newly inserted leaf

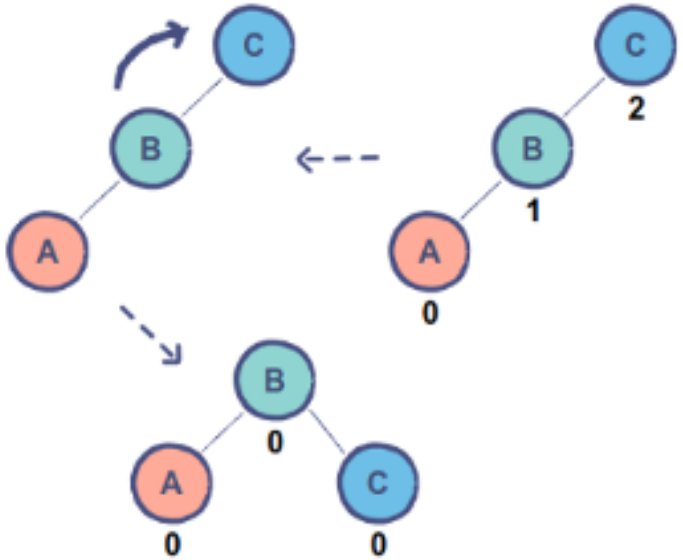
Single Right Rotation



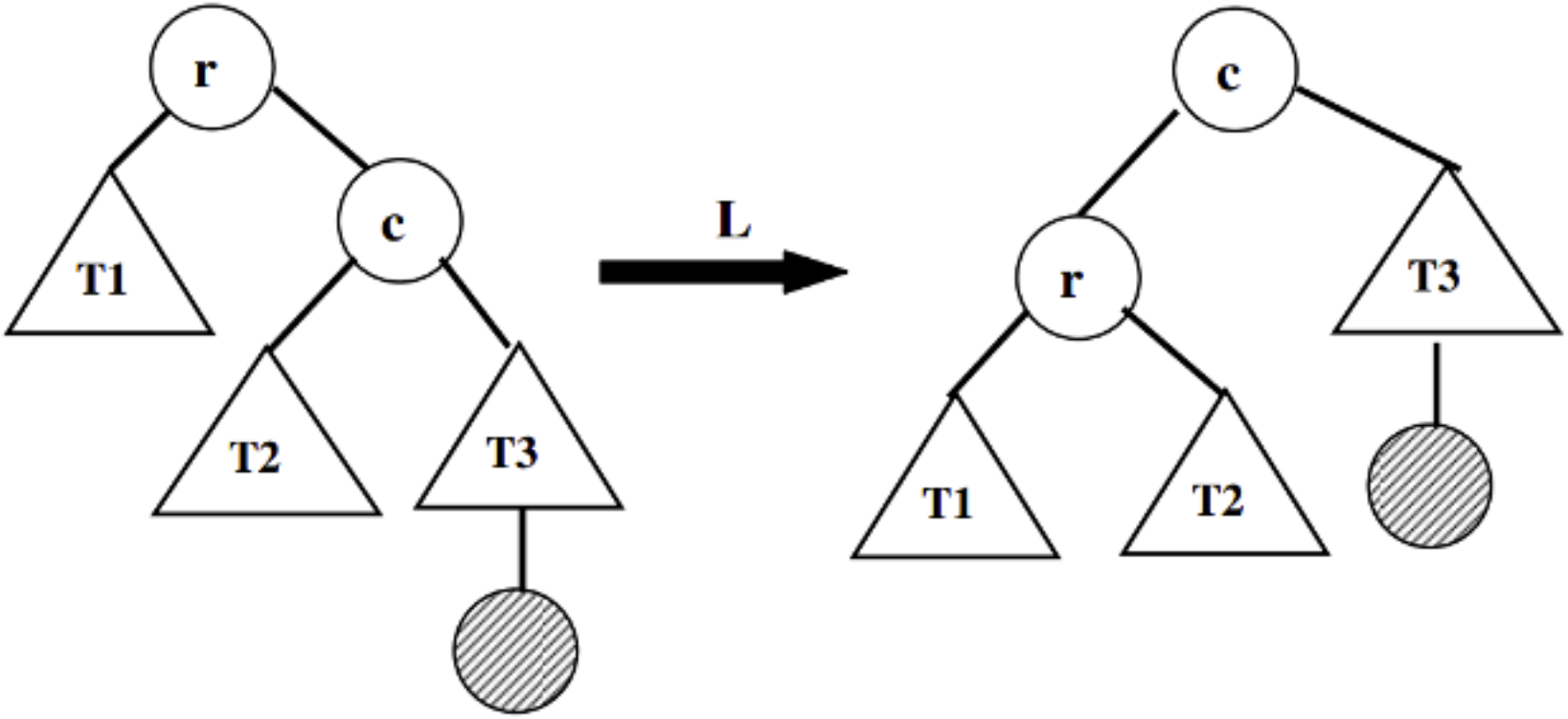
- Here rotation is performed after a new key is inserted into the left sub-tree of the left child of a tree whose root had the balance of +1 before the insertion

Single Right Rotation



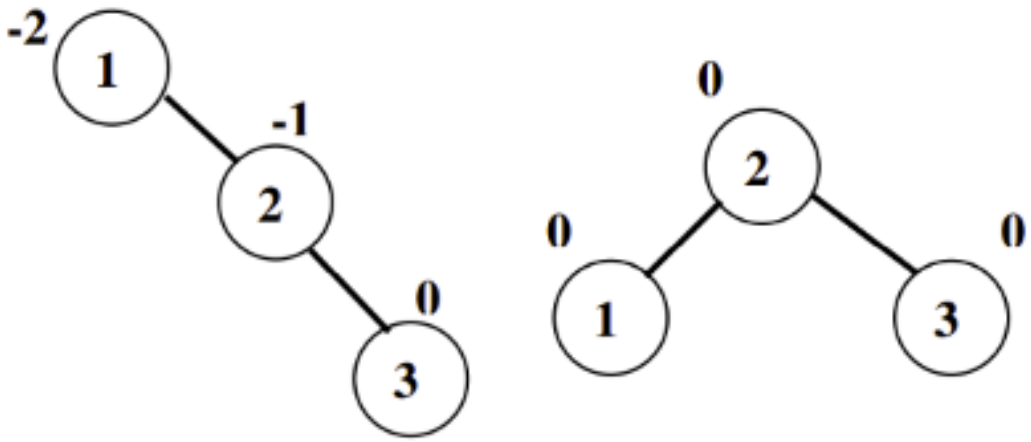


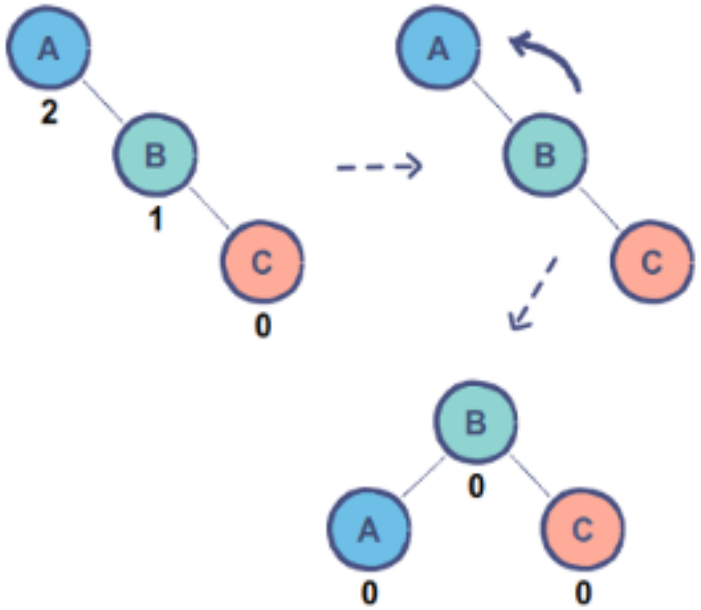
Single Left Rotation



- Here rotation is performed after a new key is inserted into the right sub-tree of the right child of a tree whose root had the balance of -1 before the insertion

Single Left Rotation

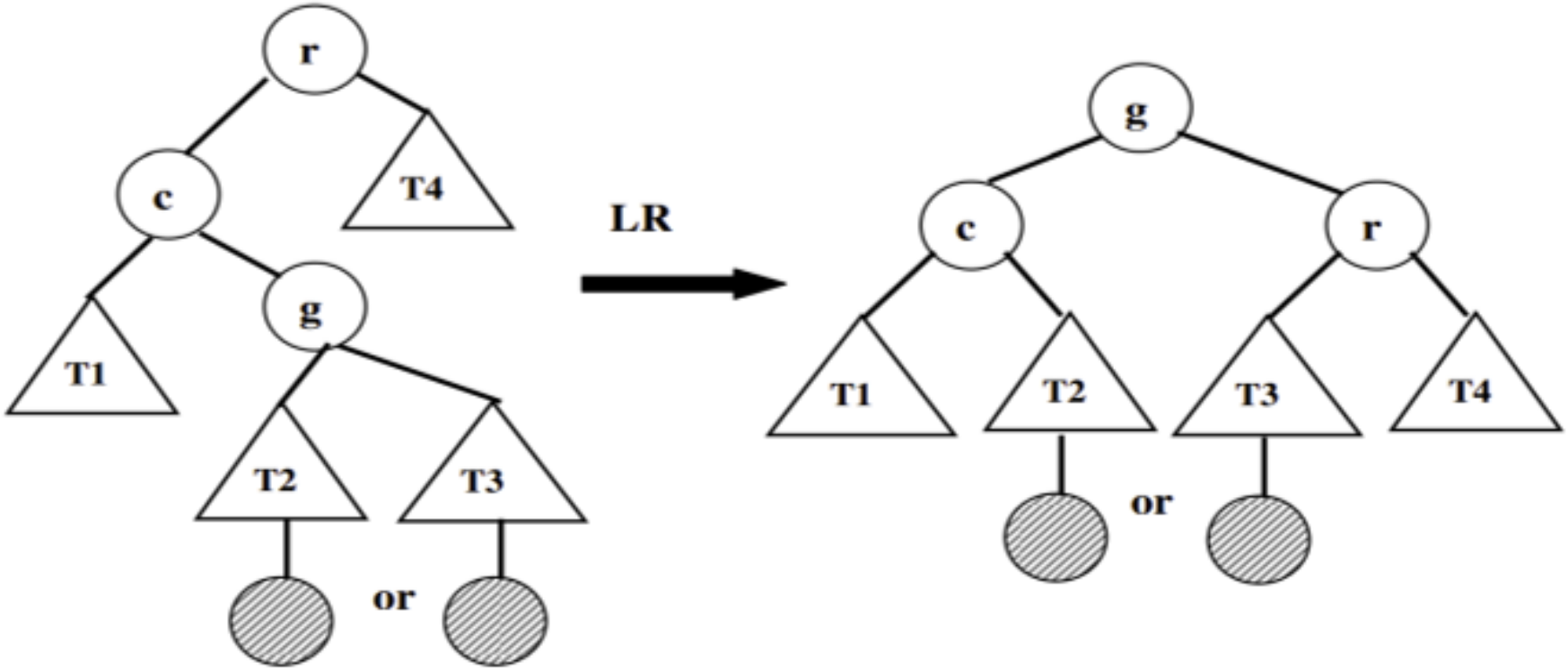




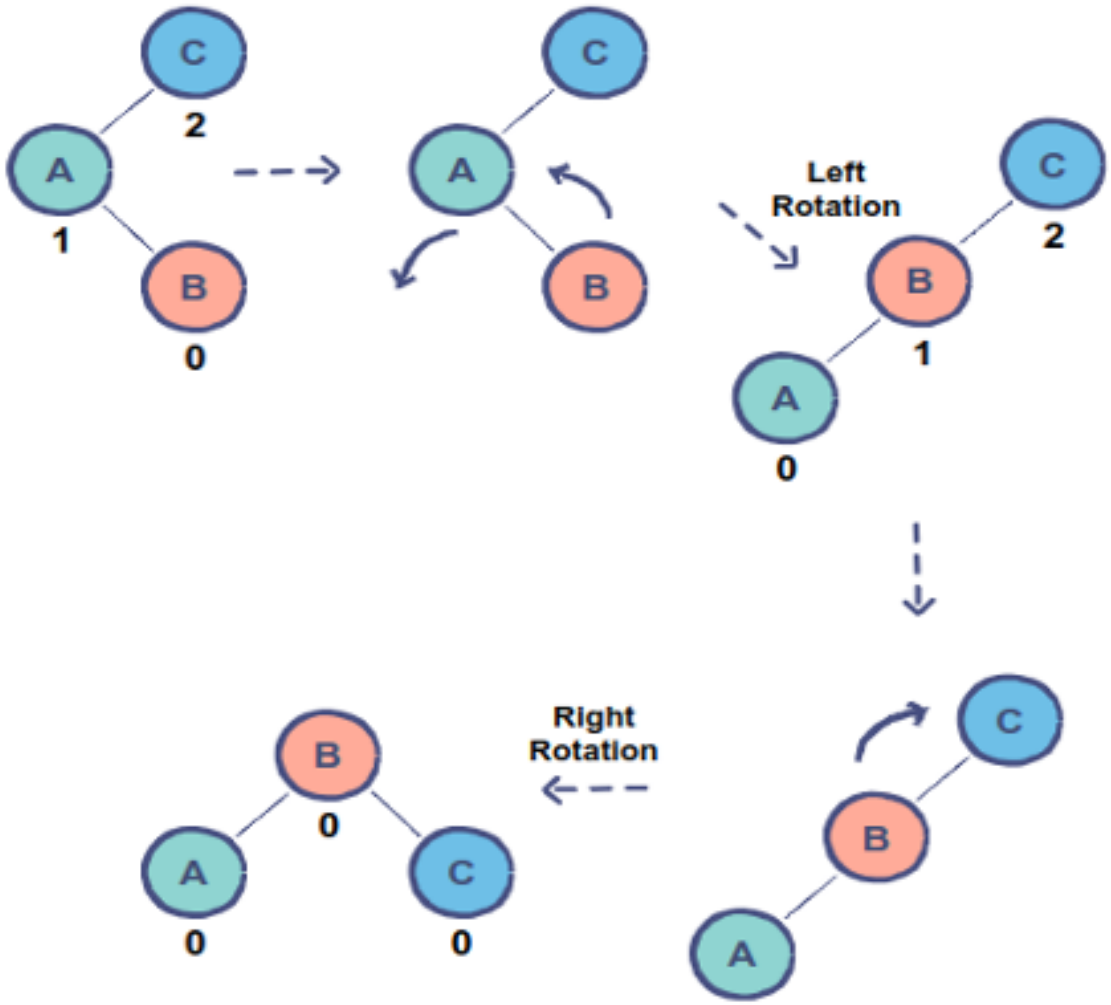
Double – Left Right Rotation

- **Combination of two rotations**
  - perform left rotation of the left sub-tree of root r
  - perform right rotation of the new tree rooted at r
- It is performed after a new key is inserted into the **right sub-tree of the left child of a tree whose root had the balance of +1** before the insertion

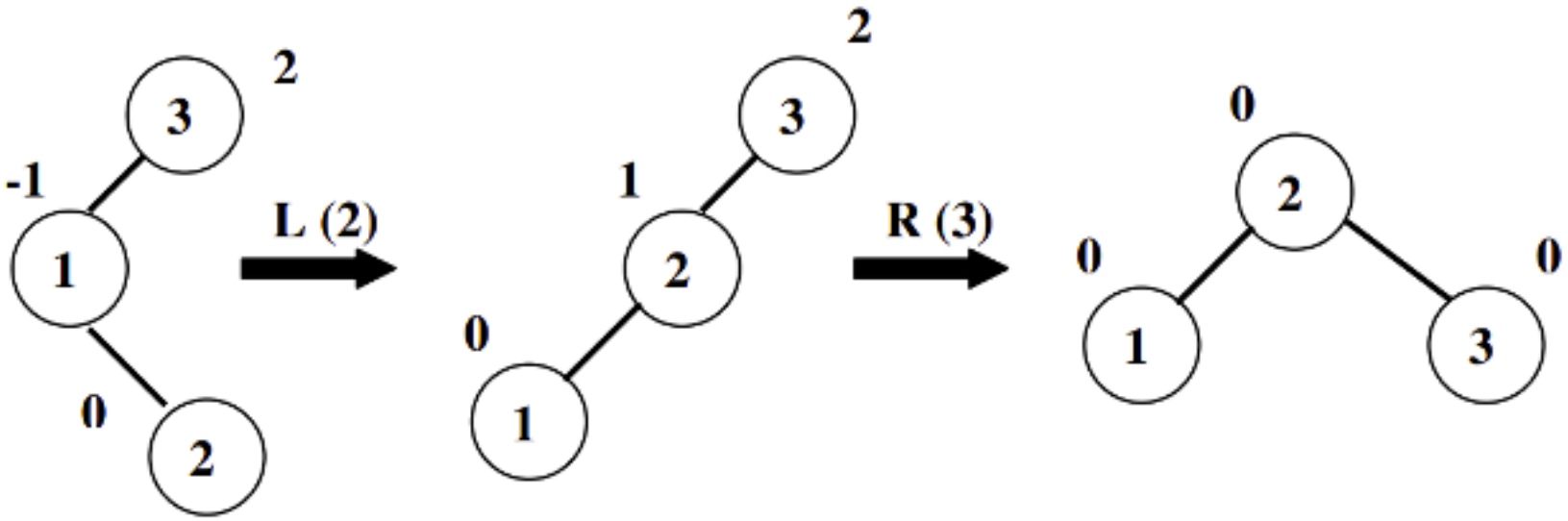
Double – Left Right Rotation



Double – Left Right Rotation



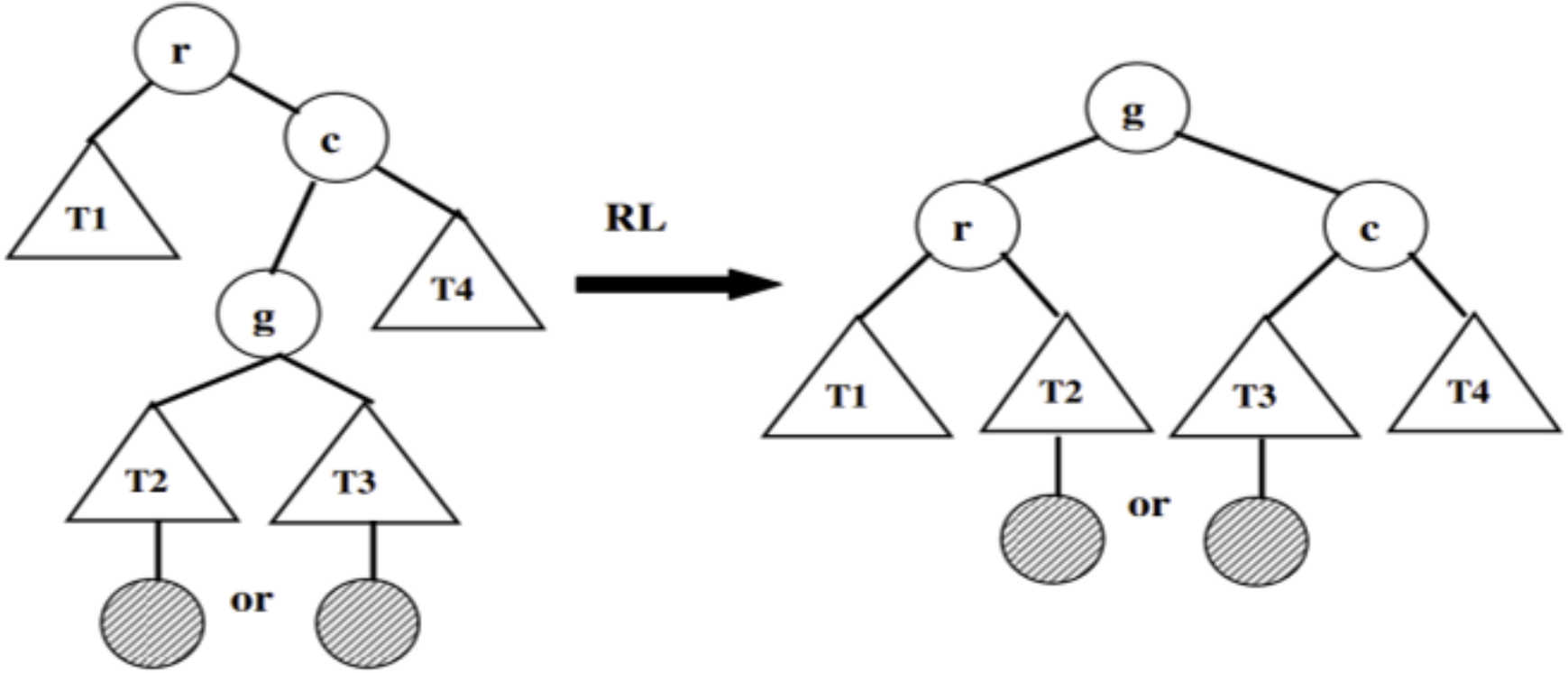
Double – Left Right Rotation



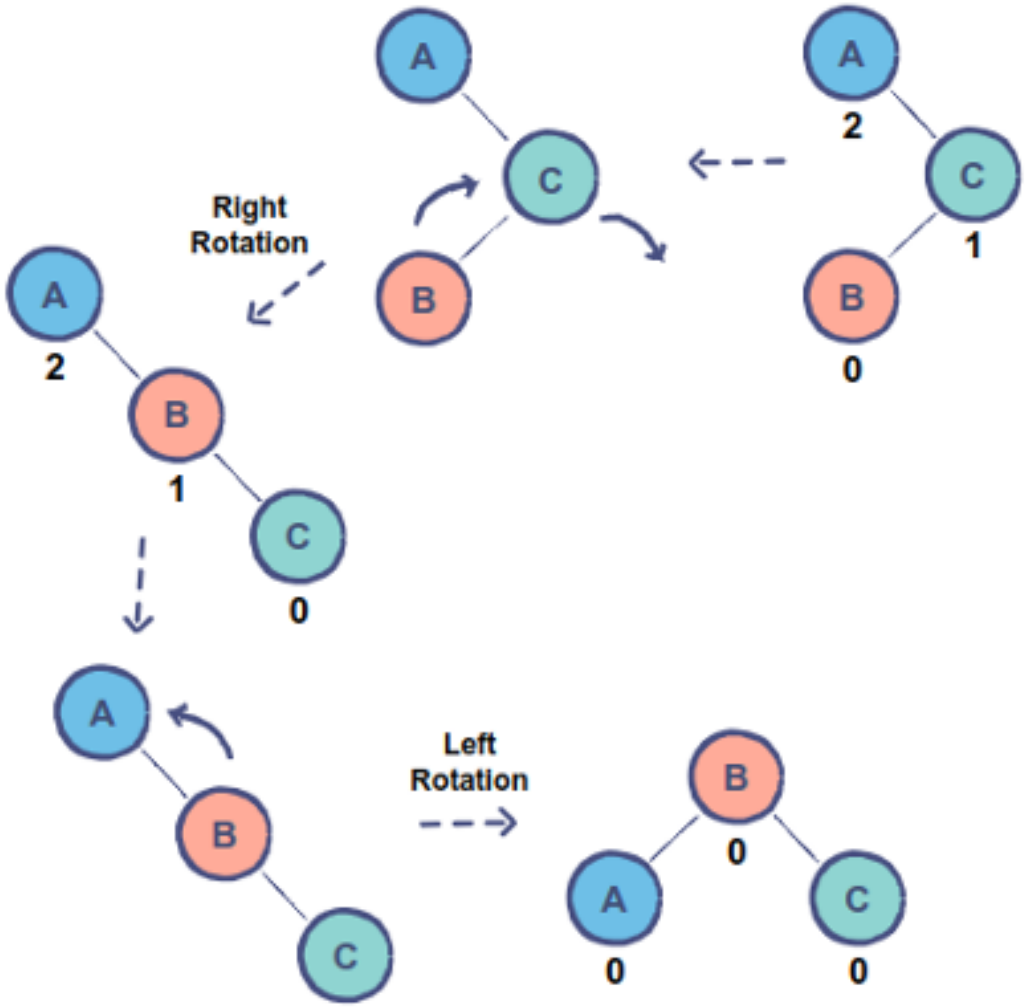
Double – Right Left Rotation

- Combination of two rotations
  - perform right rotation of the right sub-tree of root r
  - perform left rotation of the new tree rooted at r
- It is performed after a new key is inserted into the left sub-tree of the right child of a tree whose root had the balance of -1 before the insertion

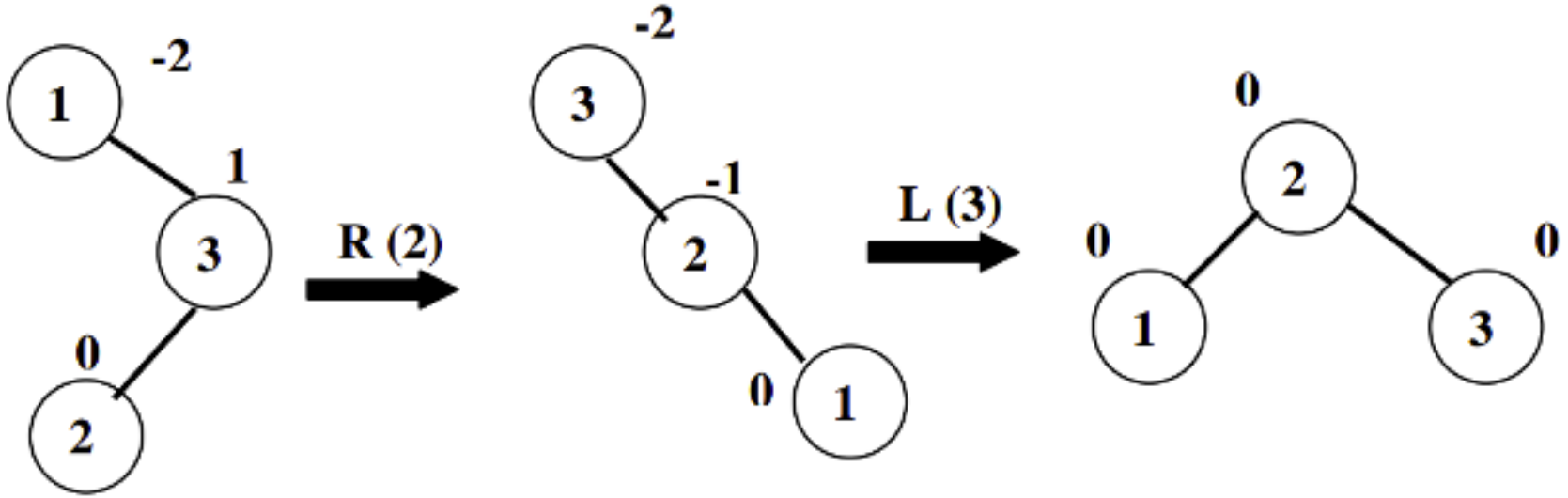
Double – Right Left Rotation

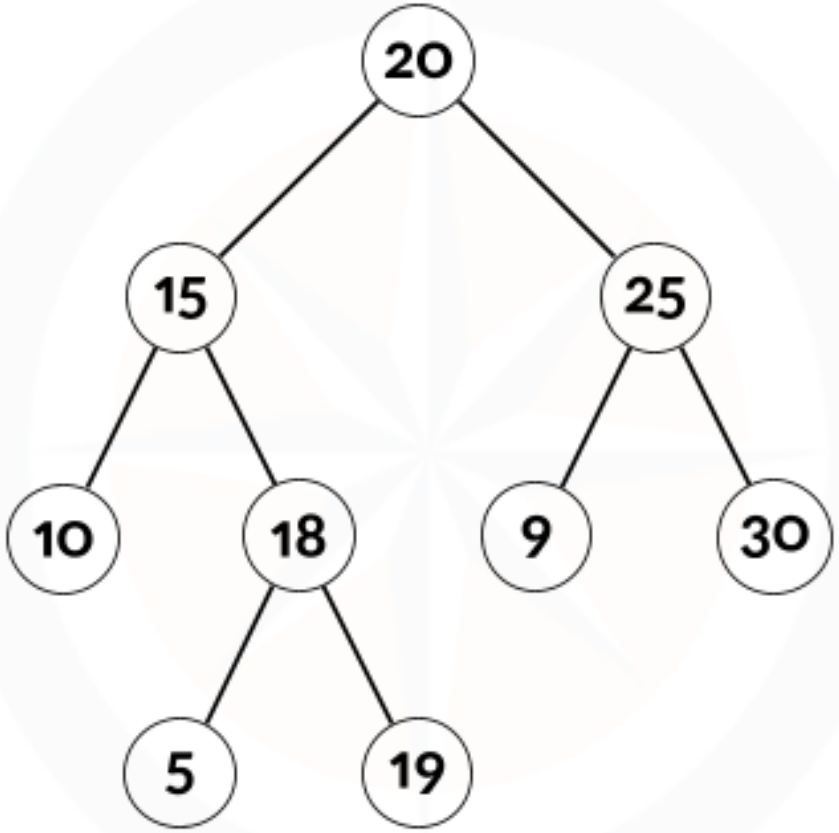


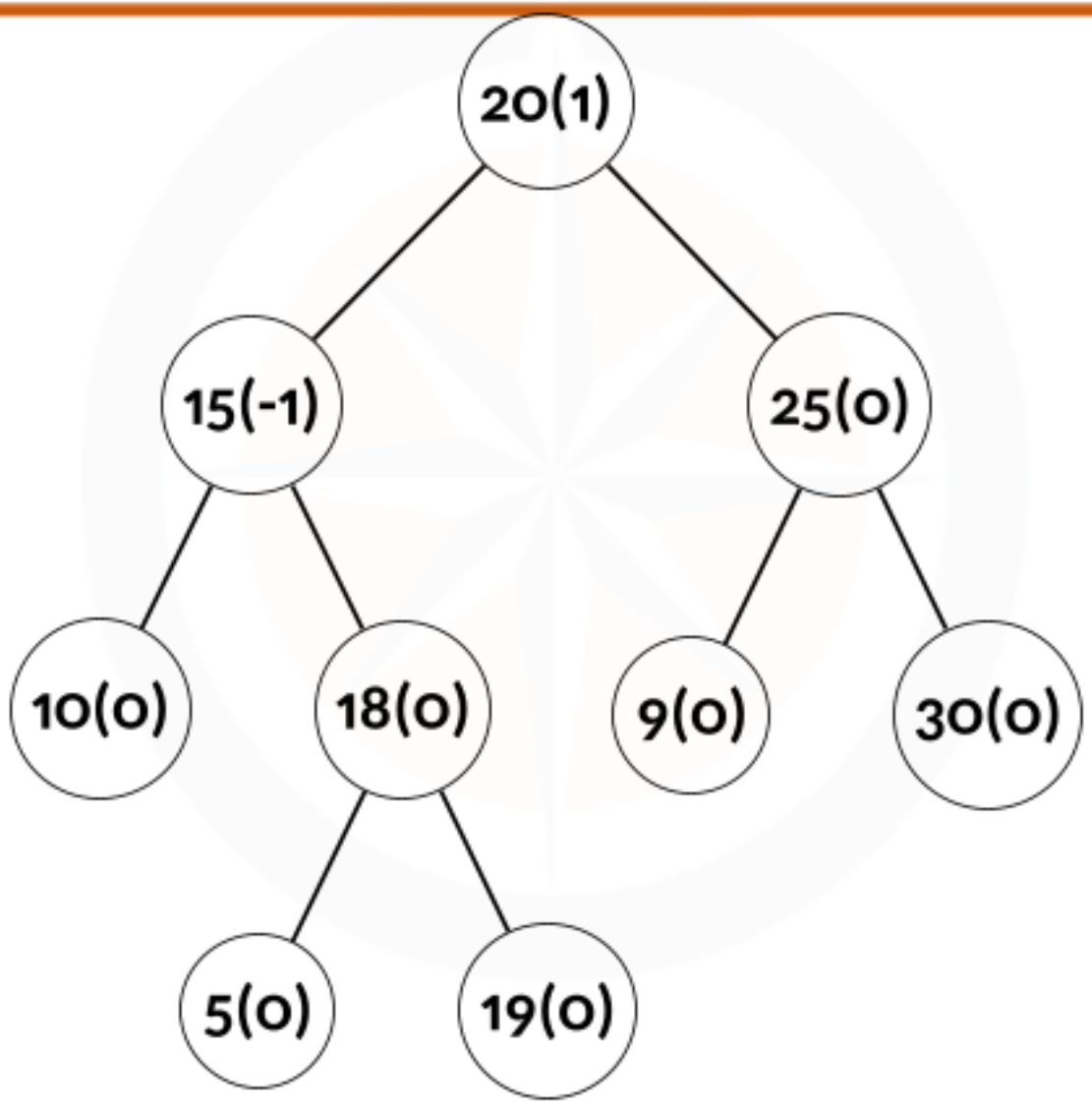
Double – Right Left Rotation



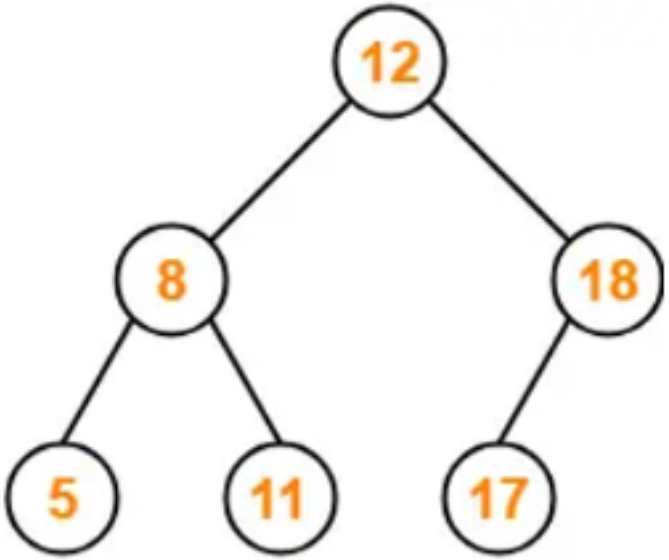
Double – Right Left Rotation

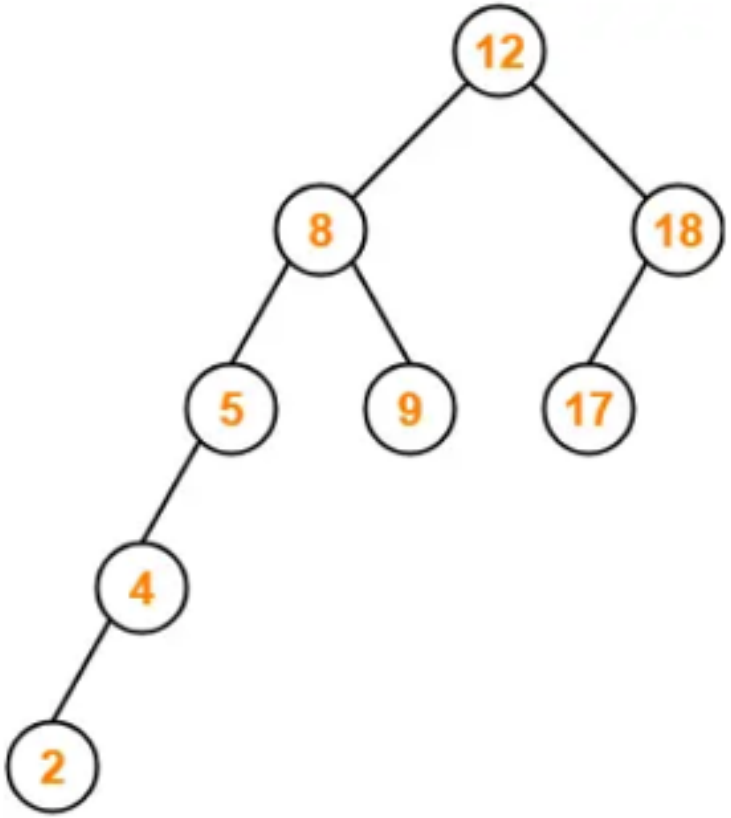



20
15
25
10
18
9
30
5
19


20(1)
15(-1)
25(0)
10(0)
18(0)
9(0)
30(0)
5(0)
19(0)

Illustrations





Class Exercises

- Construct a AVL tree step by step for the following
  1. 20,18,12,40,43,33,29,60
  2. 20,40,13,25,12,14,16,1,3
  3. 15,8,10,20,23,25,29

Thank You

Dilip Kumar Maripuri
Associate Professor
Department of Computer Applications
dilip.maripuri@pes.edu
8073212026