BINARY TREE

Binary tree is a tree data structure (non-linear) in which each node can have atmost two children which are referred to as left child and right child.

Properties

Size: Total number of nodes in the tree

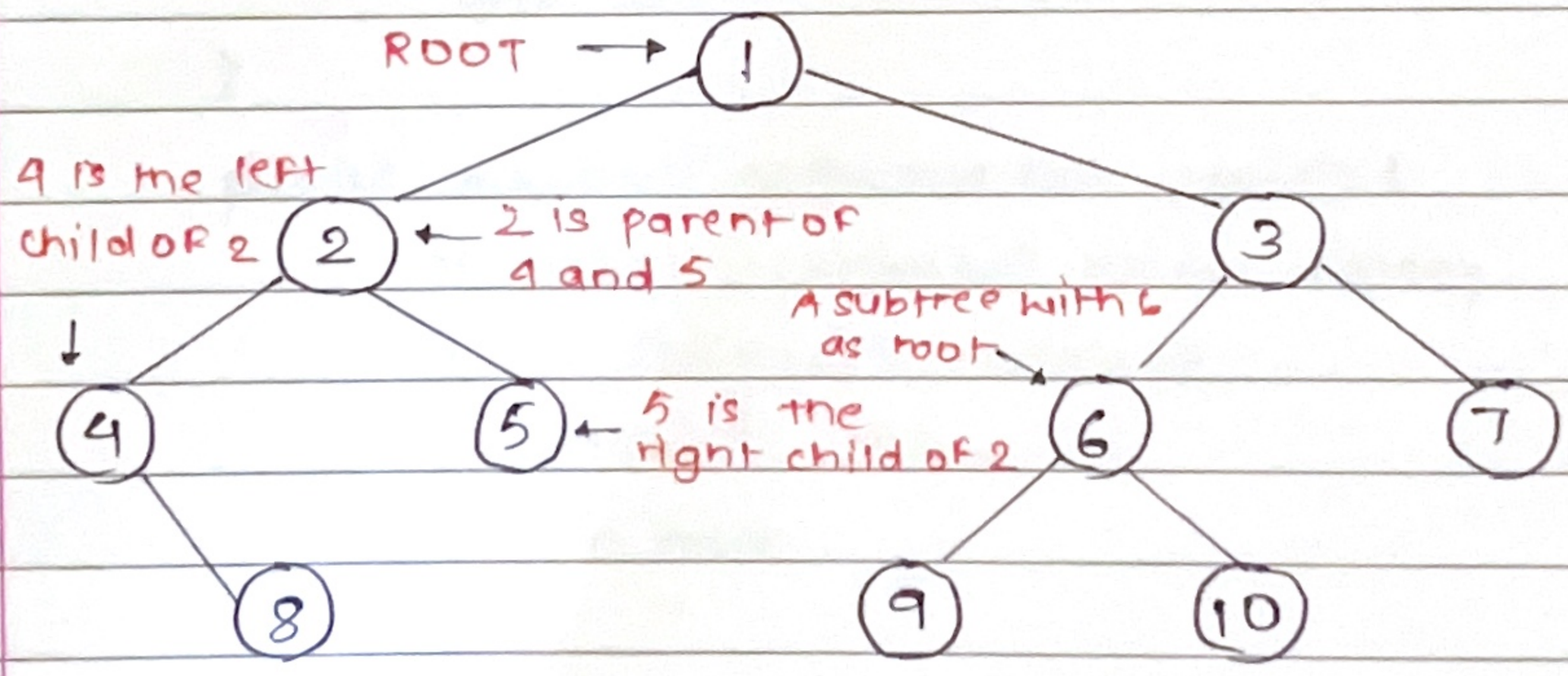
Root: The topmost node of the tree

Leaves: Nodes with no children, found at the bottom of the tree

Edges: connections between parent and child nodes.

Level: The level of a node is the number of edges on the path from the root to the node. The root is at level 0, its children at level 1 and so on

Height: The length of the longest path from the root to a leaf. In a balanced BST, the height is approximately $\log_2(n)$.

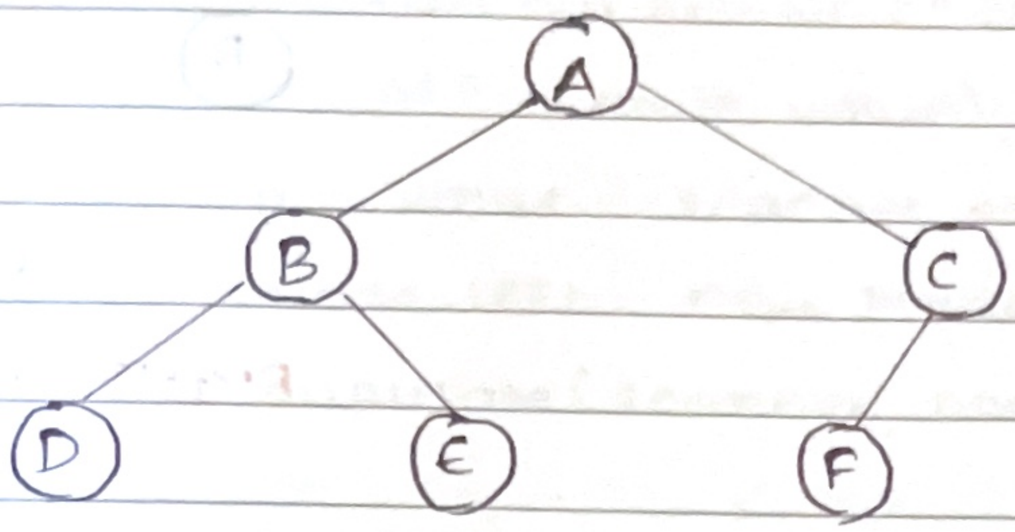


8, 5, 9, 10, 7 are leaf nodes

Types of Binary Tree

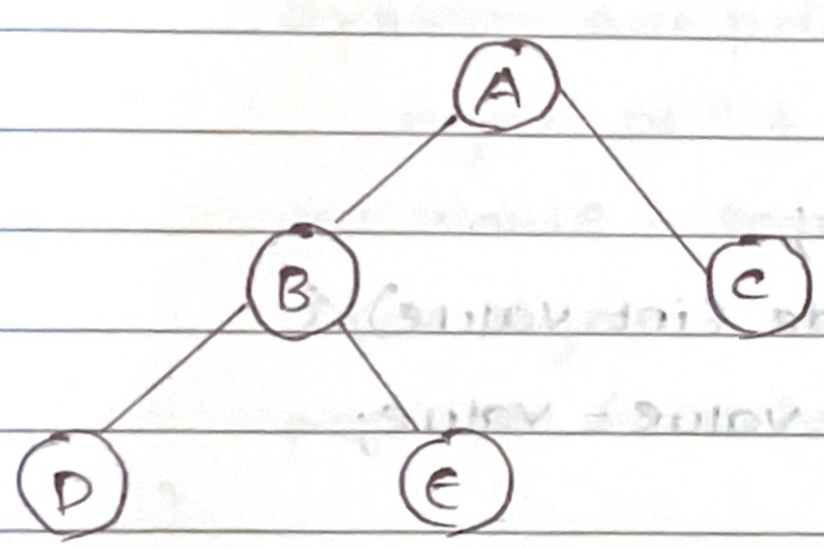
1. Complete Binary Tree

A complete binary tree is special type of tree where all levels of tree are filled completely except the lowest level nodes which are filled from as left as possible



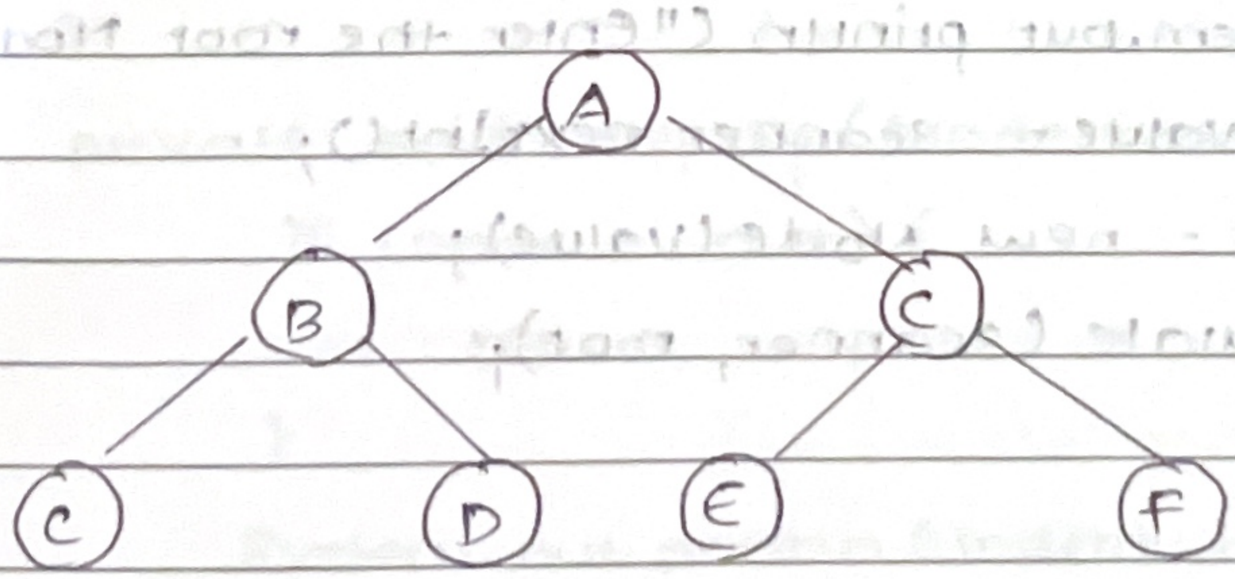
2. full Binary Tree / strict Binary Tree

A full binary tree is a binary tree with either zero or two child nodes for each node.



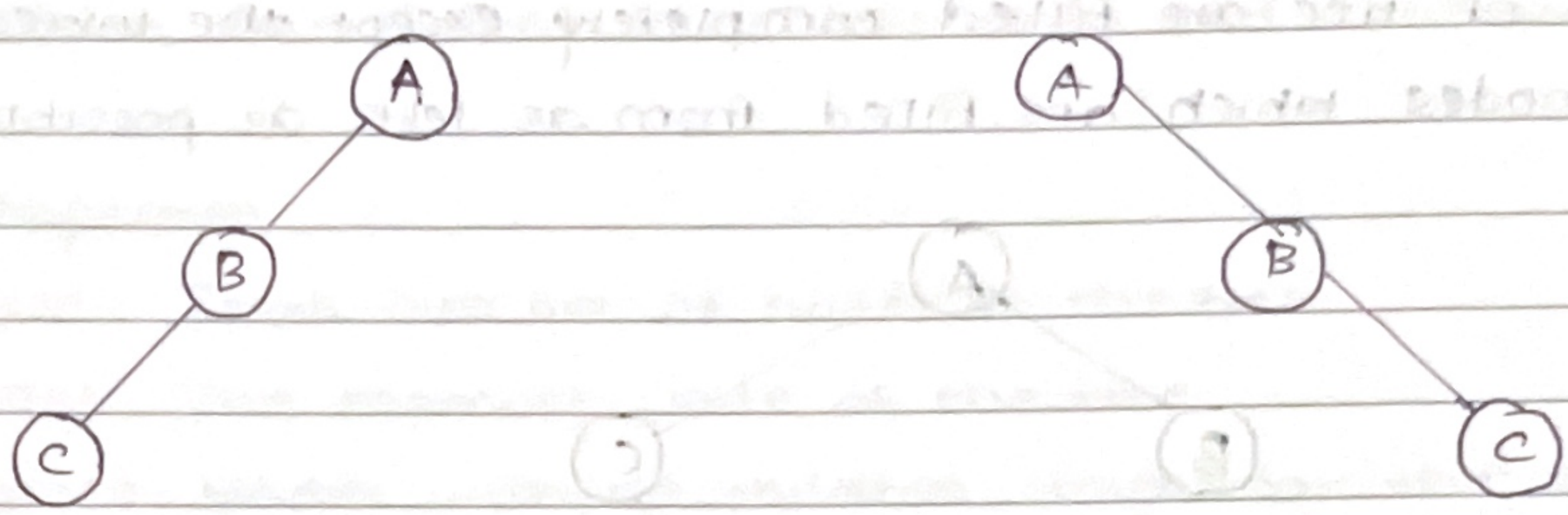
3. Perfect Binary Tree

A perfect binary tree is a special kind of binary tree in which all leaf nodes are at same depth, and all non-leaf nodes have two children.



4. Skewed Binary Tree.

A skewed Binary Tree is a type of binary tree in which all the nodes have only either one child or no child.



LEFT SKEWED RIGHT SKEWED

Implementation - Binary Tree

```
class BinaryTree {
    public BinaryTree() { }
    private static class Node {
        int value;
        Node left;
        Node right;
        public Node (int value) {
            this.value = value;
        }
    }
    private Node root;
    //insert elements
    public void populate (Scanner scanner) {
        System.out.println ("Enter the root Node : ");
        int value = scanner.nextInt();
        root = new Node (value);
        populate (scanner, root);
    }
```

```
private void populate(Scanner scanner, Node node){
    System.out.println("Do you want to enter left of " +
        node.value);
    boolean left = scanner.nextBoolean();
    if(left) {
        system.out.println("Enter the value of the left
            of" + node.value);
        int value = scanner.nextInt();
        node.left = new Node(value);
        populate(scanner, node.left);
    }
    System.out.println("Do you want to enter right of " +
        node.value);
    Boolean right = scanner.nextBoolean();
    if(right) {
        System.out.println("Enter the value of the
            right of" + node.value);
        int value = scanner.nextInt();
        node.right = new Node(value);
        populate(scanner, node.right);
    }
}

//display
public void display {
    display(this.root, "");
}
private void display(Node node, String indent){
    if(node == null) {
        return;
    }
    System.out.println(indent + node.value);
```

```
        display(node.left, indent + "\t");
        display(node.right, indent + "\t");
    }

    public void prettyDisplay(){
        prettyDisplay(root, 0);
    }

    private void prettyDisplay(Node node, int level){
        if(node == null){
            return;
        }
        prettyDisplay(node.right, level+1);
        if(level != 0){
            for(int i=0; i<level-1; i++){
                System.out.println("|\t\t");
            }
            System.out.println("|------>" + node.value);
        }else{
            System.out.println(node.value);
        }
        prettyDisplay(node.left, level+1);
    }
    // for copying/maths  N → L → R
    public void preOrder(){
        preOrder(root);
    }

    private void preOrder(Node node){
        if(node == null){
            return;
        }
        System.out.println(node.value + " ");
        preOrder(node.left);
        preOrder(node.right);
    }
```

```
public void inOrder() {
        inOrder(root);
} // for getting elements in sorted order from BST L-N-R
private void inOrder(Node node) {
        if (node == null) {
                return;
        }
        inOrder(node.left);
        System.out.println(node.value + " ");
        inOrder(node.right);
}
// for deletion  L-R-N
public void postOrder() {
        postOrder(root);
}
private void postOrder(Node node) {
        if (node == null) {
                return;
        }
        postOrder(node.left);
        postOrder(node.right);
        System.out.println(node.value + " ");
}
}
```