PW SKILLS

Lesson Plan

Binary Trees - 2
Java

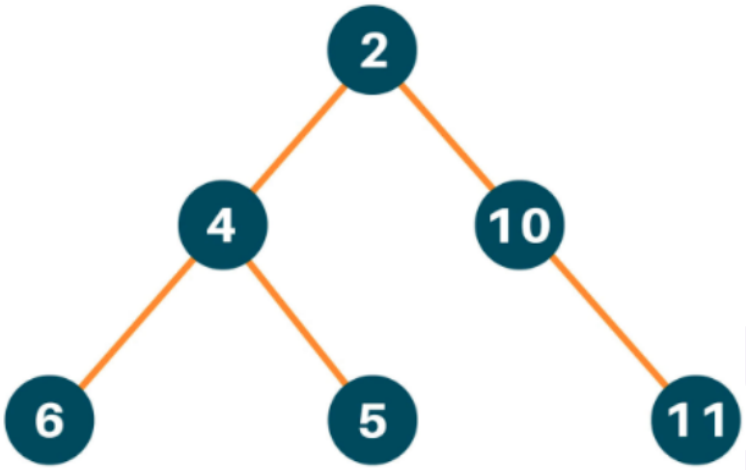
PW SKILLS

Today's Checklist

- Traversals - Preorder, Inorder, Postorder
- Print elements on nth level
- Level Order Traversal (BFS)
- Level Order Traversal (Right to Left)
- Level order traversal (Using Queue)
- Construct Tree from Level order traversal

Traversals

Let's consider this binary tree to understand various ways of traversing through it:



- **DFS**

DFS (Depth-first search) is a technique used for traversing trees or graphs. Here backtracking is used for traversal. We first traverse the deepest node and mark it visited if it isn't and then backtracks to its parent node if no sibling of that node exists

- **Preorder [LeetCode 144]**

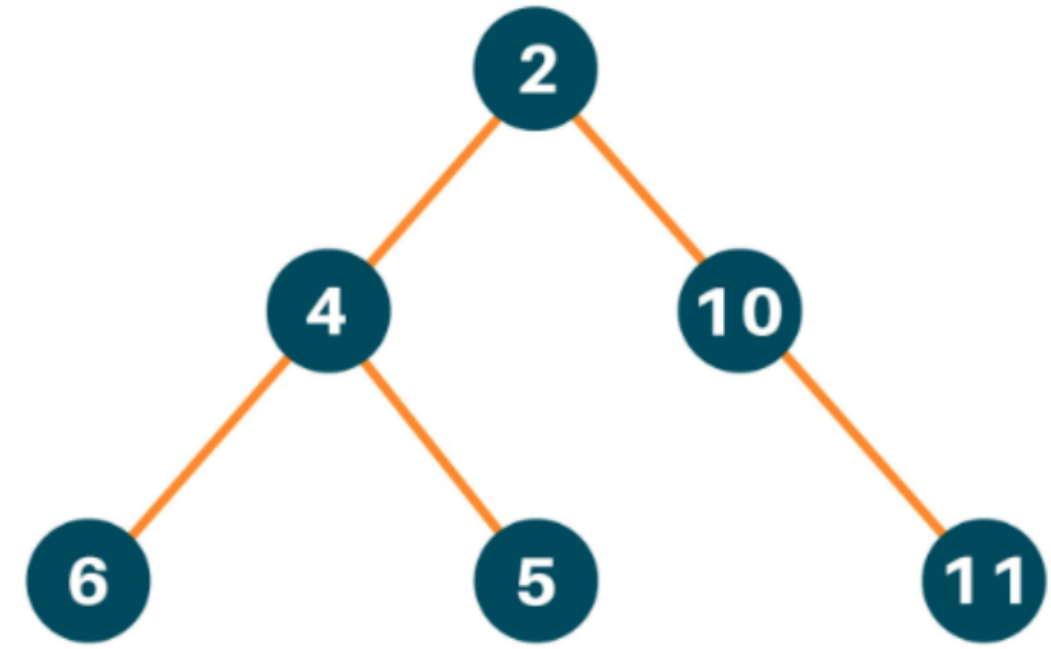
The order for Preorder traversal is
ROOT -> LEFT CHILD -> RIGHT CHILD
For the given tree, the preorder traversal would be: 2 4 6 5 10 11

```
class Node {
    int data;
    Node left;
    Node right;

    Node(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}
```

```
public class Main {
    static void preorder(Node root) {
        if (root == null) return;
        System.out.print(root.data + " ");
        preorder(root.left);
        preorder(root.right);
    }

    public static void main(String[] args) {
        Node root = new Node(2);
        root.left = new Node(4);
        root.right = new Node(10);
        root.left.left = new Node(6);
        root.left.right = new Node(5);
        root.right.right = new Node(11);
        preorder(root);
    }
}
```

Explanation: Preorder traversal is a method used for traversing the nodes of a binary tree. It refers to the process of visiting the root node, followed by the left subtree, and then the right subtree in a recursive manner. To perform a preorder traversal, we start at the root node of the tree and print the value of the node. Then we recursively traverse the left subtree, followed by the right subtree.

```
2 4 6 5 10 11

...Program finished with exit code 0
Press ENTER to exit console.
```



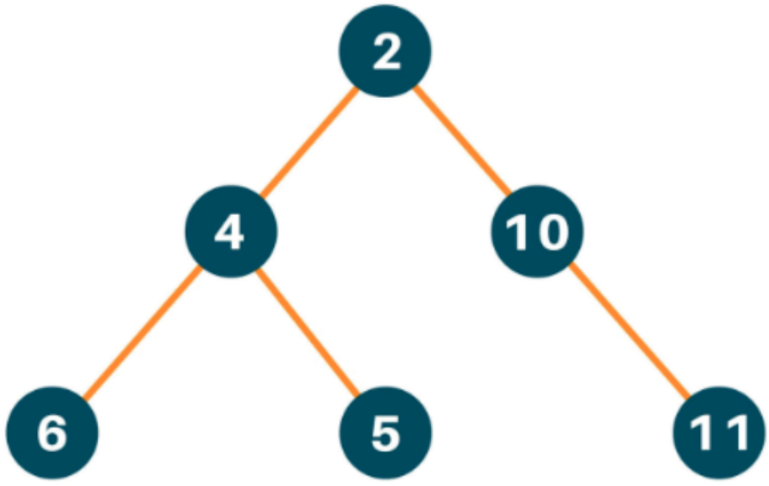
- **Inorder (Leetcode 94)**

The order for Preorder traversal is LEFT CHILD -> ROOT -> RIGHT CHILD.
For the given tree, the preorder traversal would be: 6 4 5 2 10 11

Code:

```
class Node {
    int data;
    Node left;
    Node right;

    Node(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}

public class Main {
    static void inorder(Node root) {
        if (root == null) return;
        inorder(root.left);
        System.out.print(root.data + " ");
        inorder(root.right);
    }

    public static void main(String[] args) {
        Node root = new Node(2);
        root.left = new Node(4);
        root.right = new Node(10);
        root.left.left = new Node(6);
        root.left.right = new Node(5);
        root.right.right = new Node(11);
        inorder(root);
    }
}
```

Explanation: Inorder traversal is a method used for traversing the nodes of a binary tree. It refers to the process of visiting the left subtree, followed by the root node, and then the right subtree in a recursive manner.
To perform an inorder traversal, we start at the root node of the tree and recursively traverse the left subtree until we reach a leaf node. Once we have reached a leaf node, we print the value of the node, and then recursively traverse the right subtree.




```
6 4 5 2 10 11

...Program finished with exit code 0
Press ENTER to exit console.
```

- **Postorder (Leetcode 145)**

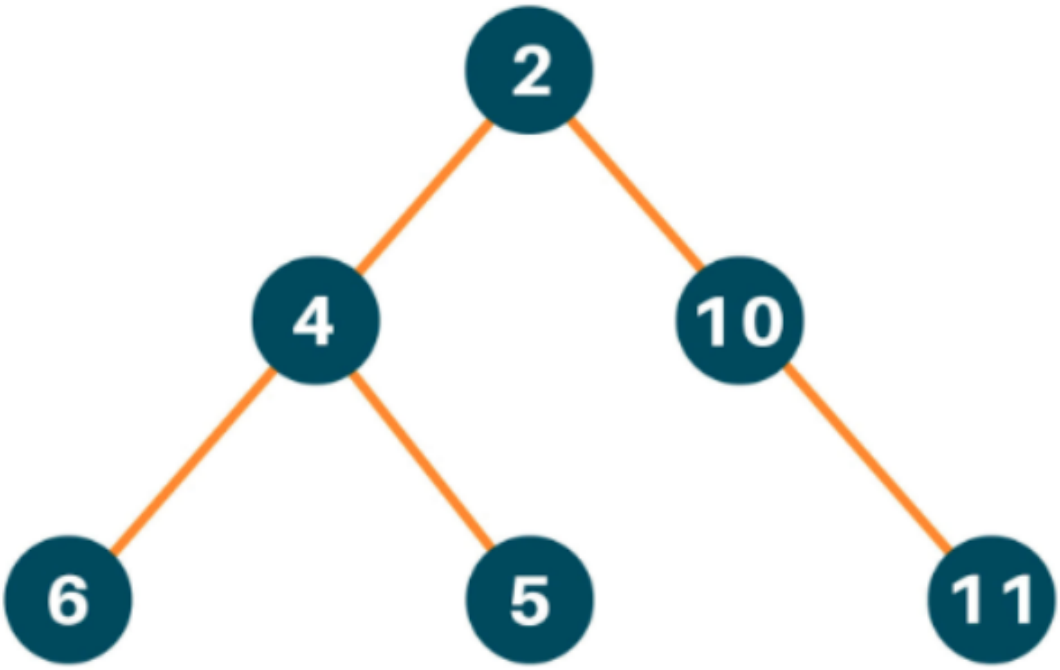
The order for Preorder traversal is LEFT CHILD -> RIGHT CHILD -> ROOT
For the given tree, the preorder traversal would be: 6 5 4 11 10 2

Code:

```
class Node {
    int data;
    Node left;
    Node right;

    Node(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}

public class Main {
    static void postorder(Node root) {
        if (root == null) return;
        postorder(root.left);
        postorder(root.right);
        System.out.print(root.data + " ");
    }
```

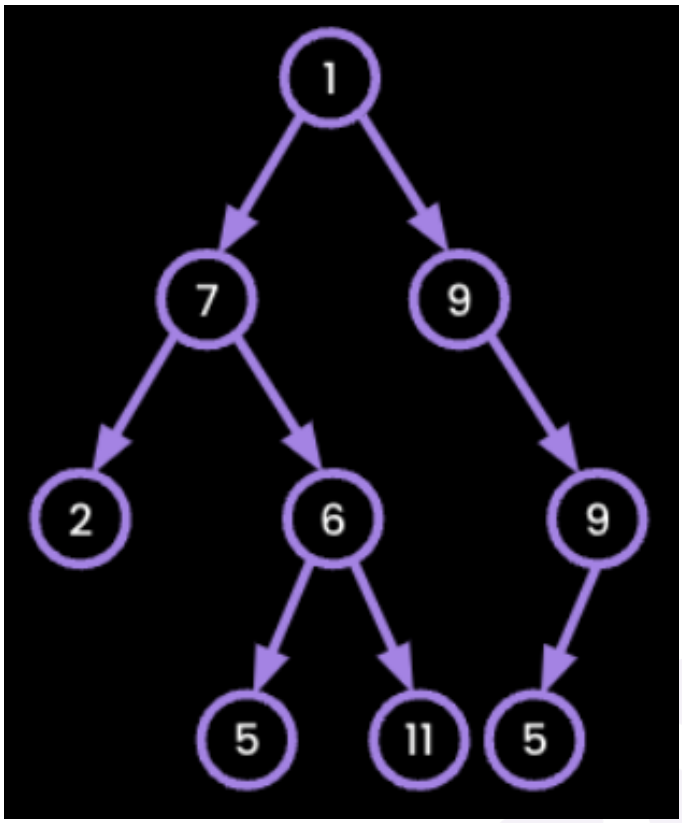
```
    public static void main(String[] args) {
        Node root = new Node(2);
        root.left = new Node(4);
        root.right = new Node(10);
        root.left.left = new Node(6);
        root.left.right = new Node(5);
        root.right.right = new Node(11);
        postorder(root);
    }
}
```

Explanation: Postorder traversal is a common method used for traversing or visiting the nodes of a binary tree. It refers to the process of visiting the left subtree, followed by the right subtree, and then the root node in a recursive manner.
To perform a postorder traversal, we start at the root node of the tree and recursively traverse the left subtree, followed by the right subtree. Once we have traversed both subtrees, we print the value of the root node.




```
6 5 4 11 10 2

...Program finished with exit code 0
Press ENTER to exit console.
```

Print elements of nth level



Code:

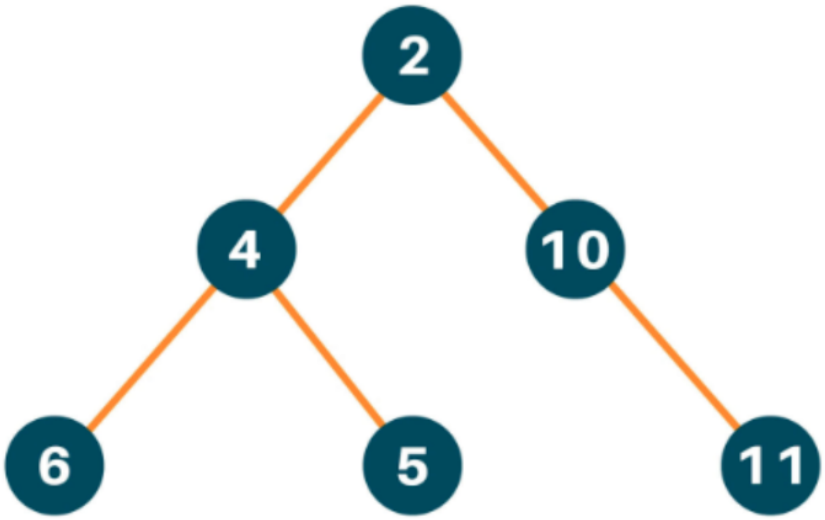
```
 import java.util.LinkedList;
import java.util.Queue;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class BinaryTree {

    static void printNthLevel(TreeNode root, int level) {
        if (root == null || level < 1) {
            System.out.println("Invalid level or empty tree!");
            return;
        }
```

```
        Queue<TreeNode> q = new LinkedList<>();
        q.add(root);
        int currentLevel = 1;

        while (!q.isEmpty() && currentLevel <= level) {
            int nodesAtCurrentLevel = q.size();

            if (currentLevel == level) {
                System.out.print("Elements at level " + level +
": ");
                for (int i = 0; i < nodesAtCurrentLevel; ++i) {
                    TreeNode node = q.poll();
                    System.out.print(node.val + " ");
                }
                System.out.println();
                return;
            }

            for (int i = 0; i < nodesAtCurrentLevel; ++i) {
                TreeNode node = q.poll();

                if (node.left != null) {
                    q.add(node.left);
                }
                if (node.right != null) {
                    q.add(node.right);
                }
            }

            ++currentLevel;
        }

        System.out.println("Level " + level + " not found in the
tree!");
    }

    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);
        root.right.left = new TreeNode(6);
        root.right.right = new TreeNode(7);

        int levelToPrint = 3;
        printNthLevel(root, levelToPrint);
    }
}
```

Explanation: printNthLevel function uses a queue for level-order traversal.
It iterates through levels, stopping at the given level 'n'.
When it reaches the desired level, it prints the elements found.
Otherwise, it notifies if the level is invalid or not found in the tree.

- **Levelorder**

Level order traversal is a method used for traversing the nodes of a binary tree by exploring the nodes level by level. We visit all the nodes of the same level before moving on to the next level.
To perform a level order traversal we take a queue and push the root node in it. In each iteration we pop one node from the queue and push its left and right children in the queue. This way we are able to traverse the tree level wise.



Code:

```
import java.util.LinkedList;
import java.util.Queue;

class Node {
    public int data;
    public Node left;
    public Node right;

    public Node(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}

public class BinaryTree {

    public static void levelOrder(Node root) {
        Queue<Node> q = new LinkedList<>();
        q.offer(root);
```

```
        while (!q.isEmpty()) {
            int size = q.size();
            while (size > 0) {
                Node temp = q.poll();
                System.out.print(temp.data + " ");

                if (temp.left != null) {
                    q.offer(temp.left);
                }
                if (temp.right != null) {
                    q.offer(temp.right);
                }
                size--;
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        Node root = new Node(2);
        root.left = new Node(4);
        root.right = new Node(10);
        root.left.left = new Node(6);
        root.left.right = new Node(5);
        root.right.right = new Node(11);

        levelOrder(root);
    }
}
```

Explanation: The inner loop of the function levelorder() basically runs for all the elements of a level, while the outer loop makes sure that we visit and push all the tree nodes in the queue.

```
2
4 10
6 5 11


...Program finished with exit code 0
Press ENTER to exit console.
```

Level order traversal (Right to Left)

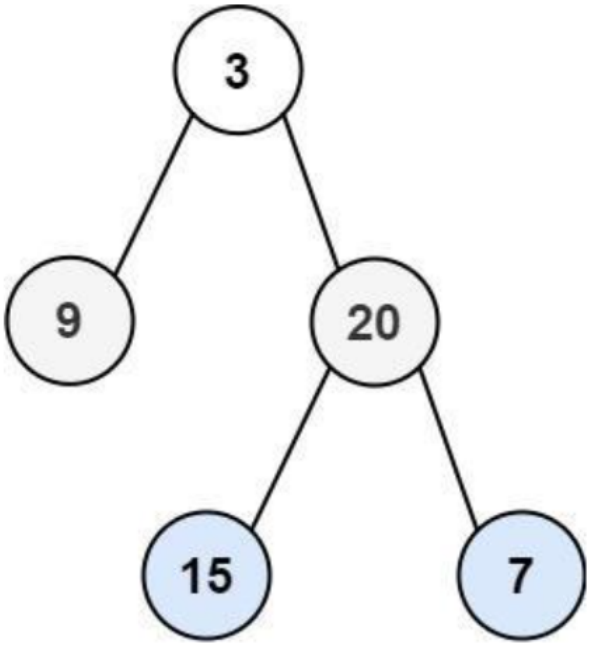
Code:

```
public static void levelOrderRightToLeft(TreeNode root) {
        if (root == null) {
            System.out.println("Empty tree!");
            return;
        }

        Queue<TreeNode> queue = new LinkedList<>();
        Stack<Integer> levelValues = new Stack<>();

        queue.offer(root);

        while (!queue.isEmpty()) {
            TreeNode node = queue.poll();

            levelValues.push(node.val);

            if (node.right != null) {
                queue.offer(node.right);
            }
            if (node.left != null) {
                queue.offer(node.left);
            }
        }

        System.out.print("Level order traversal (Right to Left):
");
        while (!levelValues.isEmpty()) {
            System.out.print(levelValues.pop() + " ");
        }
        System.out.println();
    }
```

Explanation:
levelOrderRightToLeft function traverses the tree level by level from right to left using a queue and a stack.
It starts at the root and iterates through each level using a queue (maintaining a right-to-left order).
It stores the values at each level in a stack.
After traversal, it prints the values from the stack, which now holds the level order traversal in the desired right-to-left fashion.

Q. Binary Tree Level Order Traversal [LeetCode 102]

Given the root of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).



Input: root = [3,9,20,null,null,15,7]
Output: [[3],[9,20],[15,7]]

Code:

```
public class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        if (root == null) return res;

        Queue<TreeNode> q = new LinkedList<>();
        q.offer(root);

        while (!q.isEmpty()) {
            int n = q.size();
            List<Integer> level = new ArrayList<>();

            for (int i = 0; i < n; i++) {
                TreeNode p = q.poll();
                level.add(p.val);

                if (p.left != null) q.offer(p.left);
                if (p.right != null) q.offer(p.right);
            }

            res.add(level);
        }

        return res;
    }

}
```

Explanation: The idea is to use a count variable which stores number of nodes at each level
Time: O(n) to traverse all the nodes

In the given code:

Time Complexity: O(n) - 'n' being the number of nodes in the tree. The code traverses each node exactly once using a level-order traversal approach.

Space Complexity: O(n) - It uses a queue to perform the level-order traversal, potentially storing all nodes in the queue (worst-case scenario). The `res` vector holds the resulting levels, potentially containing 'n' elements if all nodes are present in the tree.

Level order traversal (Using Queue)

Code:

```
public class BinaryTreeTraversal {
    public List<List<Integer>> levelOrder(TreeNode root) {
        if (root == null) return new ArrayList<>(); // Return
empty list if root is null

        List<List<Integer>> result = new ArrayList<>();
        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);

        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            List<Integer> currentLevel = new ArrayList<>();

            for (int i = 0; i < levelSize; ++i) {
                TreeNode node = queue.poll();
                currentLevel.add(node.val);

                if (node.left != null) queue.add(node.left);
                if (node.right != null) queue.add(node.right);
            }

            result.add(currentLevel);
        }

        return result;
    }

  }
```

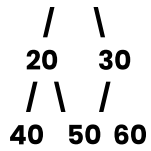
Explanation:
The levelOrder function performs level order traversal using a queue.
It initializes an empty result vector of vectors to store levels.
It starts by pushing the root into the queue.
While the queue is not empty, it processes nodes at each level, adding their values to the result vector.

It populates the result vector with levels as vectors of integers.
Finally, the main function demonstrates the use of levelOrder and prints the level order traversal of the tree.

Construct Tree from Level order traversal

Given an array of elements, the task is to insert these elements in level order and construct a tree.

Input : arr[] = {10, 20, 30, 40, 50, 60}
Output : 10



Code:

```
public class ConstructTreeFromLevelOrder {
    public TreeNode constructTree(List<Integer> levelOrder) {
        if (levelOrder.isEmpty()) return null;

        TreeNode root = new TreeNode(levelOrder.get(0));
        Queue<TreeNode> nodes = new LinkedList<>();
        nodes.add(root);

        for (int i = 1; i < levelOrder.size(); i += 2) {
            TreeNode parent = nodes.poll();

            if (levelOrder.get(i) != -1) {
                parent.left = new TreeNode(levelOrder.get(i));
                nodes.add(parent.left);
            }

            if (i + 1 < levelOrder.size() && levelOrder.get(i +
1) != -1) {
                parent.right = new TreeNode(levelOrder.get(i +
1));
                nodes.add(parent.right);
            }
        }

        return root;
    }

    // Example usage:
    public static void main(String[] args) {
        ConstructTreeFromLevelOrder constructor = new
ConstructTreeFromLevelOrder();
        List<Integer> levelOrder = List.of(3, 9, 20, -1, -1, 15,
7); // Modify the level order here
```

```
        TreeNode root = constructor.constructTree(levelOrder);
        System.out.println("Constructed tree from level order");
        // Perform operations with the tree constructed
    }
}
```

Explanation:
The constructTree function takes a vector levelOrder representing the level order traversal of a binary tree.
It creates a binary tree from this level order traversal.
The code initializes a queue and starts constructing the tree by popping elements from the level order vector.
It creates nodes for each element in the level order and links them appropriately based on their positions.
The -1 in the levelOrder vector represents a null node.
The inorderTraversal function demonstrates a basic inorder traversal to verify the constructed tree.

PW SKILLS

THANK YOU !

PW SKILLS