Worksheet- 20

257. Binary Tree Paths

```
class Pair {
   TreeNode node;
   List<Integer> list;
   public Pair(TreeNode node, List<Integer> list){
      this.node = node;
      this.list = list;
   }
}

class Solution {
   // Helper method to recursively find all paths from a given node to leaf nodes
   private void getAllPaths(TreeNode node, List<Integer> path, List<List<Integer>> allPaths) {
      // If the current node is a leaf node, add the path to all Paths
      if (node.left == null && node.right == null) {
         path.add(node.val);
         allPaths.add(new ArrayList<>(path));
         path.remove(path.size() - 1);
         return;
      }
      // Add the current node to the path
      path.add(node.val);
      // Recursively traverse the left and right subtrees
      if (node.left != null) {
         getAllPaths(node.left, path, allPaths);
      }
      if (node.right != null) {
```

```
        getAllPaths(node.right, path, allPaths);
    }
    // Remove the current node from the path as we backtrack
    path.remove(path.size() - 1);
}

// Helper method to perform a level-order traversal and find all paths iteratively
private void lvlOrder(TreeNode node, List<List<Integer>> allPaths) {
    Queue<Pair> q = new LinkedList<>();
    q.add(new Pair(node, new ArrayList<>()));
    while (!q.isEmpty()) {
        Pair ele = q.peek();
        q.remove();
        node = ele.node;
        // If the current node is a leaf node, add the path to allPaths
        if (node.left == null && node.right == null) {
            ele.list.add(node.val);
            allPaths.add(new ArrayList<>(ele.list));
        }
        // Add the current node to the path
        ele.list.add(node.val);
        // Enqueue the left and right children of the current node along with the updated path
        if (node.left != null) {
            q.add(new Pair(node.left, new ArrayList<>(ele.list)));
        }
        if (node.right != null) {
            q.add(new Pair(node.right, new ArrayList<>(ele.list)));
        }
    }
}
```

```
    public List<String> binaryTreePaths(TreeNode root) {
        List<List<Integer>> allPaths = new ArrayList<>();
        List<String> ans = new ArrayList<>();

        // Perform level-order traversal to find all paths
        lvlOrder(root, allPaths);

        // Convert the integer paths to string paths
        for (List<Integer> arr : allPaths) {
            StringBuffer str = new StringBuffer();
            for (int i = 0; i < arr.size() - 1; i++) {
                str.append(String.valueOf(arr.get(i)) + "->");
            }
            str.append(String.valueOf(arr.get(arr.size() - 1)));
            ans.add(str.toString());
        }

        return ans;
    }
}

public class Main {
    public static void main(String[] args) {
        // Create a binary tree
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.right = new TreeNode(5);

        // Create an instance of the Solution class
        Solution solution = new Solution();
```

```
        // Call the binaryTreePaths method to find all paths from root to leaf nodes
        List<String> paths = solution.binaryTreePaths(root);

        // Print the paths
        for (String path : paths) {
            System.out.println(path);
        }
    }
}
```

258. Add Digits

```
class Solution {
    public int addDigits(int num) {
        int temp = 0;
        // Calculate the digital root of the given number
        temp = numnum(num);
        // Continue calculating the digital root until it becomes a single-digit number
        while (temp > 9) {
            temp = numnum(temp);
        }
        return temp;
    }

    public int numnum(int num) {
        int temp = 0;
        // Extract the digits of the number and add them together
        while (num > 9) {
            temp += num % 10;
            num /= 10;
        }
        return temp + num;
    }
}

public class Main {
    public static void main(String[] args) {
        Solution solution = new Solution();
        int num = 12345;
        // Calculate the digital root of the given number
        int result = solution.addDigits(num);
        System.out.println("Digital root of " + num + ": " + result);
```

```
    }
}
```