ASSIGNMENT:DAY_07-08

Task 1: Balanced Binary Tree Check

Write a function to check if a given binary tree is balanced. A balanced tree is one where the height of two subtrees of any node never differs by more than one.

ANS:

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

public class BSTcheck {

    public boolean isBalanced(TreeNode root) {
        return checkHeight(root) != -1;
    }

    private int checkHeight(TreeNode node) {
        if (node == null) return 0;
```

```java
        int leftHeight = checkHeight(node.left);
        if (leftHeight == -1) return -1;

        int rightHeight = checkHeight(node.right);
        if (rightHeight == -1) return -1;
        if (Math.abs(leftHeight - rightHeight) > 1) return -1;
        return Math.max(leftHeight, rightHeight) + 1;
    }
    public static void main(String[] args) {
        BSTcheck treeChecker = new BSTcheck();

        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.left=new TreeNode(7);
        root.left.left.left = new TreeNode(8);

        System.out.println("Tree is Balanced ?" +( treeChecker.isBalanced(root)==true ? "Yes":"No"));
    }
}
//code-by-RUBY
```

Task 2: Trie for Prefix Checking

Implement a trie data structure in java that supports insertion of strings and provides a method to check if a given string is a prefix of any word in the trie.

ANS:

```
import java.util.HashMap;

import java.util.Map;


class TrieNode {

    Map<Character, TrieNode> children;

    boolean isEndOfWord;


    public TrieNode() {

        children = new HashMap<>();

        isEndOfWord = false;

    }

}


public class Trie {

    private TrieNode root;


    public Trie() {

        root = new TrieNode();

    }
```

```java
public void insert(String word) {
    TrieNode current = root;
    for (char ch : word.toCharArray()) {
        current = current.children.computeIfAbsent(ch, c -> new TrieNode());
    }
    current.isEndOfWord = true;
}

public boolean startsWith(String prefix) {
    TrieNode current = root;
    for (char ch : prefix.toCharArray()) {
        current = current.children.get(ch);
        if (current == null) {
            return false;
        }
    }
    return true;
}

public static void main(String[] args) {
    Trie trie = new Trie();
    trie.insert("apple");
    trie.insert("app");
    trie.insert("application");
```

```java
        System.out.println(trie.startsWith("app"));

        System.out.println(trie.startsWith("appl"));

        System.out.println(trie.startsWith("banana"));

    }

}


//code-by-RUBY
```

Task 3: Implementing Heap Operations

Code a min-heap in Java with methods for insertion, deletion, and fetching the minimum element. Ensure that the heap property is maintained after each operation.

ANS:

```java
import java.util.ArrayList;

import java.util.List;


public class MinHeap {

    private List<Integer> heap;


    public MinHeap() {

        heap = new ArrayList<>();

    }


    public void insert(int value) {

        heap.add(value);

        heapifyUp(heap.size() - 1);
```

```java
}

public int deleteMin() {
   if (heap.size() == 0) {
      throw new IllegalStateException("Heap is empty");
   }
   if (heap.size() == 1) {
      return heap.remove(0);
   }

   int minValue = heap.get(0);
   heap.set(0, heap.remove(heap.size() - 1));
   heapifyDown(0);
   return minValue;
}

public int getMin() {
   if (heap.size() == 0) {
      throw new IllegalStateException("Heap is empty");
   }
   return heap.get(0);
}

private void heapifyUp(int index) {
   int parentIndex = (index - 1) / 2;
```

```
    if (index > 0 && heap.get(index) < heap.get(parentIndex)) {

        swap(index, parentIndex);
      heapifyUp(parentIndex);
    }
}

private void heapifyDown(int index) {
    int leftChild = 2 * index + 1;
    int rightChild = 2 * index + 2;
    int smallest = index;

    if (leftChild < heap.size() && heap.get(leftChild) < heap.get(smallest)) {
        smallest = leftChild;
    }
    if (rightChild < heap.size() && heap.get(rightChild) < heap.get(smallest)) {
        smallest = rightChild;
    }
    if (smallest != index) {
        swap(index, smallest);
        heapifyDown(smallest);
    }
}

private void swap(int index1, int index2) {
```

```
        int temp = heap.get(index1);
        heap.set(index1, heap.get(index2));
        heap.set(index2, temp);
    }

    public static void main(String[] args) {
        MinHeap minHeap = new MinHeap();
        minHeap.insert(3);
        minHeap.insert(1);
        minHeap.insert(6);
        minHeap.insert(5);
        minHeap.insert(2);
        minHeap.insert(4);
        System.out.println(minHeap.heap);
        System.out.println("Min value: " + minHeap.getMin());
        System.out.println("Removed min value: " + minHeap.deleteMin());
        System.out.println("New min value: " + minHeap.getMin());

    }
}
//code-by-RUBY
```

Task 4: Graph Edge Addition Validation

Given a directed graph, write a function that adds an edge between two nodes and then checks if the graph still has no cycles. If a cycle is created, the edge should not be added.

ANS:

```
import java.util.*;


public class DirectedGraph_EdgeValidation {

    private Map<Integer, List<Integer>> adjList;


    public DirectedGraph_EdgeValidation() {

        this.adjList = new HashMap<>();

    }


    public void addNode(int node) {

        adjList.putIfAbsent(node, new ArrayList<>());

    }


    public boolean addEdge(int from, int to) {

        if (!adjList.containsKey(from) || !adjList.containsKey(to)) {

            throw new IllegalArgumentException("Node does not exist");

        }



        adjList.get(from).add(to);
```

```java
    if (hasCycle()) {

        adjList.get(from).remove((Integer) to);
        return false;
    }

    return true;
}

private boolean hasCycle() {
    Set<Integer> visited = new HashSet<>();
    Set<Integer> recStack = new HashSet<>();

    for (int node : adjList.keySet()) {
        if (hasCycleUtil(node, visited, recStack)) {
            return true;
        }
    }

    return false;
}

private boolean hasCycleUtil(int node, Set<Integer> visited, Set<Integer> recStack) {
```

```
        if (recStack.contains(node)) {
            return true;
        }
        if (visited.contains(node)) {
            return false;
        }

        visited.add(node);
        recStack.add(node);

        for (int neighbor : adjList.get(node)) {
            if (hasCycleUtil(neighbor, visited, recStack)) {
                return true;
            }
        }

        recStack.remove(node);
        return false;
    }

    public static void main(String[] args) {
        DirectedGraph_EdgeValidation graph = new DirectedGraph_EdgeValidation();
        graph.addNode(1);
        graph.addNode(2);
        graph.addNode(3);
```

```
        graph.addNode(4);


        System.out.println("Add only if no cycle is formed");
        System.out.println("ADDED : "+ (graph.addEdge(1, 2)==true? "YES":"NO"));
        System.out.println("ADDED : "+ (graph.addEdge(2, 3)==true? "YES":"NO"));
        System.out.println("ADDED : "+ (graph.addEdge(3, 4)==true? "YES":"NO"));
        System.out.println("ADDED : "+ (graph.addEdge(4, 1)==true? "YES":"NO"));
    }
}
//code-by-RUBY
```

Task 5: Breadth-First Search (BFS) Implementation

For a given undirected graph, implement BFS to traverse the graph starting from a given node and print each node in the order it is visited.

ANS:

```
import java.util.*;


public class BFS_Graph {
    private Map<Integer, List<Integer>> adjList;


    public BFS_Graph() {
        this.adjList = new HashMap<>();
    }


    public void addNode(int node) {
```

```
    adjList.putIfAbsent(node, new ArrayList<>());
}

public void addEdge(int node1, int node2) {
    adjList.putIfAbsent(node1, new ArrayList<>());
    adjList.putIfAbsent(node2, new ArrayList<>());
    adjList.get(node1).add(node2);
    adjList.get(node2).add(node1);
}

public void bfs(int startNode) {
    Set<Integer> visited = new HashSet<>();
    Queue<Integer> queue = new LinkedList<>();

    visited.add(startNode);
    queue.add(startNode);

    while (!queue.isEmpty()) {
        int node = queue.poll();
        System.out.print(node + " ");

        for (int neighbor : adjList.get(node)) {
            if (!visited.contains(neighbor)) {
                visited.add(neighbor);
                queue.add(neighbor);
```

```
                }
            }
        }
    }

    public static void main(String[] args) {
        BFS_Graph graph = new BFS_Graph();
        graph.addNode(1);
        graph.addNode(2);
        graph.addNode(3);
        graph.addNode(4);

        graph.addEdge(1, 2);
        graph.addEdge(1, 3);
        graph.addEdge(2, 4);
        graph.addEdge(3, 4);

        System.out.print("BFS starting from node 4: ");
        graph.bfs(4);
    }
}
#code-by-RUBY
```

Task 6: Depth-First Search (DFS) Recursive

Write a recursive DFS function for a given undirected graph. The function should visit every node and print it out.

ANS:

```
public class DFS_Graph {

  private Map<Integer, List<Integer>> adjList;


  public DFS_Graph() {

    this.adjList = new HashMap<>();

  }


  public void addNode(int node) {

    adjList.putIfAbsent(node, new ArrayList<>());

  }


  public void addEdge(int node1, int node2) {

    adjList.putIfAbsent(node1, new ArrayList<>());

    adjList.putIfAbsent(node2, new ArrayList<>());

    adjList.get(node1).add(node2);

    adjList.get(node2).add(node1);

  }


  public void dfs(int startNode) {

    Set<Integer> visited = new HashSet<>();
```

```
    dfsRecursive(startNode, visited);
}

private void dfsRecursive(int node, Set<Integer> visited) {
    visited.add(node);
    System.out.print(node + " ");

    for (int neighbor : adjList.get(node)) {
        if (!visited.contains(neighbor)) {
            dfsRecursive(neighbor, visited);
        }
    }
}

public static void main(String[] args) {
    DFS_Graph graph = new DFS_Graph();
    graph.addNode(1);
    graph.addNode(2);
    graph.addNode(3);
    graph.addNode(4);

    graph.addEdge(1, 2);
    graph.addEdge(1, 3);
    graph.addEdge(2, 4);
    graph.addEdge(3, 4);
```

```java
        System.out.print("DFS starting from node 4: ");
        graph.dfs(4);
    }
}
//code-by-RUBY
```