

## Day 7 and 8:

### 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.**

```
class TreeNode {  
    int val;  
    TreeNode left;  
    TreeNode right;  
  
    TreeNode(int val) {  
        this.val = val;  
    }  
}  
  
public class BalancedBinaryTree {  
  
    public boolean isBalanced(TreeNode root) {  
        return checkHeight(root) != -1;  
    }  
  
    private int checkHeight(TreeNode node) {  
        if (node == null) {  
            return 0;  
        }  
        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;
}

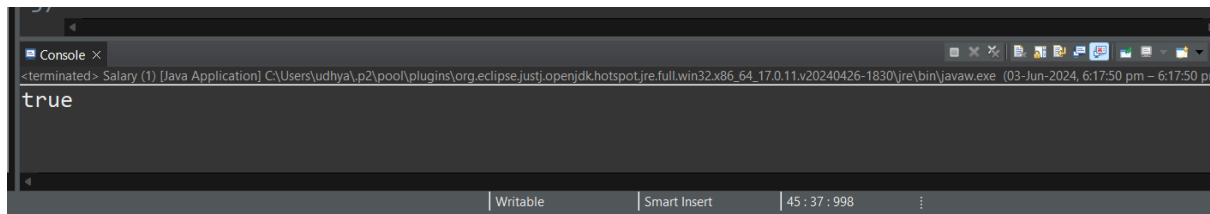
public static void main(String[] args) {
    BalancedBinaryTree solution = new BalancedBinaryTree();

    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.right = new TreeNode(6);

    System.out.println(solution.isBalanced(root));
}
}

```

## OUTPUT:



## Task 2: Trie for Prefix Checking

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

```
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();
```

```
    }
```

```
public void insert(String word) {  
    TrieNode current = root;  
    for (char ch : word.toCharArray()) {  
        current.children.putIfAbsent(ch, new TrieNode());  
        current = current.children.get(ch);  
    }  
    current.isEndOfWord = true;  
}
```

```
public boolean search(String word) {  
    TrieNode current = root;  
    for (char ch : word.toCharArray()) {  
        if (!current.children.containsKey(ch)) {  
            return false;  
        }  
        current = current.children.get(ch);  
    }  
    return current.isEndOfWord;  
}
```

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

```

        return true;
    }

    public static void main(String[] args) {

        Trie trie = new Trie();

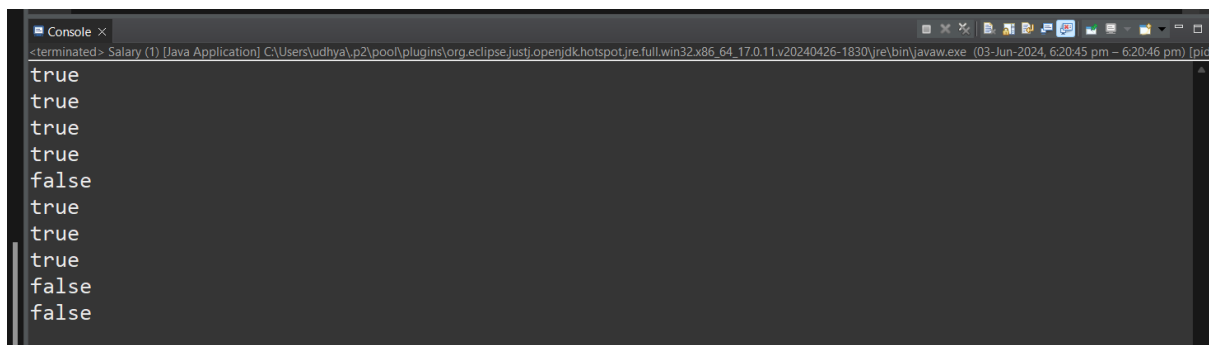
        trie.insert("apple");
        trie.insert("app");
        trie.insert("banana");
        trie.insert("bat");

        System.out.println(trie.search("apple"));    System.out.println(trie.search("app"));
        System.out.println(trie.search("banana"));    System.out.println(trie.search("bat"));
e
        System.out.println(trie.search("ball"));

        System.out.println(trie.startsWith("app"));
        System.out.println(trie.startsWith("ban"));    System.out.println(trie.startsWith("bat"));
        System.out.println(trie.startsWith("ball"));    System.out.println(trie.startsWith("zoo"));
    }
}

```

## OUTPUT:



```

true
true
true
true
false
true
true
true
false
false

```

### Task 3: Implementing Heap Operations

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

```
package com.wipro.nonlinear;

import java.util.ArrayList;
import java.util.List;

public class Heap {

    private List<Integer> heap;

    public Heap() {
        this.heap = new ArrayList<>();
    }

    public List<Integer> getHeap() {
        return new ArrayList<>(heap);
    }

    public int leftChild(int index) {
        return (index * 2) + 1;
    }

    public int rightChild(int index) {
        return (index * 2) + 2;
    }
}
```

```
public int parent(int index) {  
    return (index - 1) / 2;  
}
```

```
public void swap(int index1,int index2) {  
    int temp = heap.get(index1);  
    heap.set(index1, heap.get(index2));  
    heap.set(index2, temp);  
}
```

```
public void insert(int value) {  
    heap.add(value);  
    int current = heap.size()-1;  
    while(current > 0 && heap.get(current) > heap.get(parent(current))) {  
        swap(current,parent(current));  
        current = parent(current);  
    }  
}
```

```
public Integer remove() {  
    if(heap.size() == 0) {  
        return null;  
    }  
    if(heap.size() == 1) {  
        return heap.remove(0);  
    }  
    int maxVal = heap.get(0);
```

```
        heap.set(0, heap.remove(heap.size() - 1));  
        sinkDown(0);  
        return maxVal;e;  
    }  
}
```

```
private void sinkDown(int index) {  
    int maxIndex = index;  
    int leftIndex = leftChild(index);  
    int rightIndex = rightChild(index);  
  
    if(leftIndex < heap.size() && heap.get(leftIndex) > heap.get(maxIndex)) {  
        maxIndex = leftIndex;  
    }  
    if(rightIndex < heap.size() && heap.get(rightIndex) > heap.get(maxIndex)) {  
        maxIndex = rightIndex;  
    }  
  
    if(maxIndex != index) {  
        swap(index, maxIndex);  
        index = maxIndex;  
    }  
}
```

```
public static void main(String[] args) {  
    Heap myHeap = new Heap();  
    System.out.println(myHeap.getHeap());  
  
    myHeap.insert(99);  
}
```



```

        myHeap.insert(72);

        myHeap.insert(61);

        myHeap.insert(58);

        myHeap.insert(60);

        myHeap.insert(100);

        System.out.println(myHeap.getHeap());

        System.out.println(myHeap.remove());

        System.out.println(myHeap.getHeap());

    }

}

```

## OUTPUT:

```

<terminated> Heap [Java Application] C:\Users\udhya\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin\javaw.exe (03-Jun-2024, 6:22:17 pm - 6:22:18 pm) [p
[]
[100, 72, 99, 58, 60, 61]
100
[99, 72, 61, 58, 60]

```

## 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.**

```
package com.wipro.graph;
```

```
import java.util.ArrayList;
```

```
import java.util.HashMap;
```

```

public class Graph {

    private HashMap<String, ArrayList<String>> adjList = new HashMap<String,
ArrayList<String>>();

    public static void main(String[] args) {

        Graph myGraph = new Graph();
        myGraph.addVertex("A");
        myGraph.addVertex("B");
        myGraph.addVertex("C");
        myGraph.printGraph();

        myGraph.addEdge("A","B");
        myGraph.printGraph();
        myGraph.addEdge("A","C");
        myGraph.printGraph();
        myGraph.removeEdge("A","C");
        myGraph.printGraph();

        myGraph.removeVertex("C");
        myGraph.printGraph();

    }

    private boolean addEdge(String vertex1,String vertex2) {

        if(adjList.get(vertex1) != null && adjList.get(vertex2)!= null) {

            adjList.get(vertex1).add(vertex2);
            adjList.get(vertex2).add(vertex1);
            return true;
        }
    }
}

```

```

    }
    return false;
}

private boolean removeEdge(String vertex1,String vertex2) {
    if(adjList.get(vertex1) != null && adjList.get(vertex2)!= null) {
        adjList.get(vertex1).remove(vertex2);
        adjList.get(vertex2).remove(vertex1);
        return true;
    }
    return false;
}

private void printGraph() {
    System.out.println(adjList);
}

private boolean addVertex(String vertex) {
    if (adjList.get(vertex) == null) {
        adjList.put(vertex, new ArrayList<String>());
        return true;
    }
    return false;
}

```

```

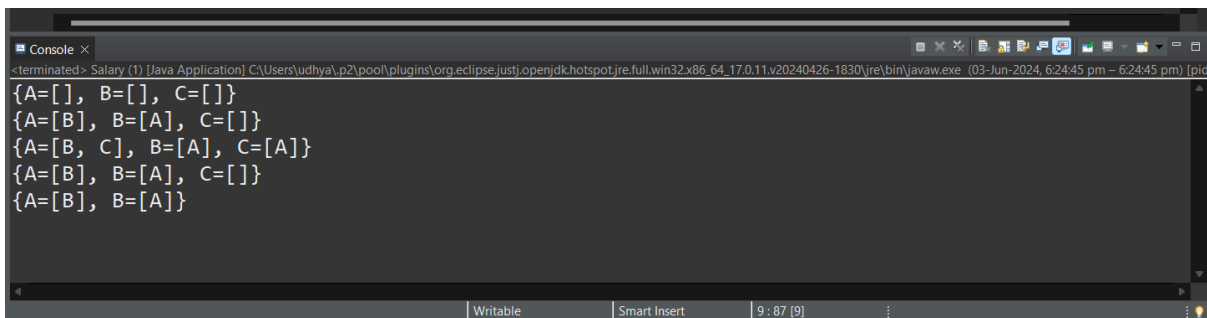
private boolean removeVertex(String vertex) {
    if (adjList.containsKey(vertex)) {

        for (ArrayList<String> list : adjList.values()) {
            list.remove(vertex);
        }

        adjList.remove(vertex);
        return true;
    }
    return false;
}

```

## OUTPUT:



```

<terminated> Salary (1) [Java Application] C:\Users\udhya\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin\javaw.exe (03-Jun-2024, 6:24:45 pm - 6:24:45 pm) [pid
{A=[], B=[], C=[]}
{A=[B], B=[A], C=[]}
{A=[B, C], B=[A], C=[A]}
{A=[B], B=[A], C=[]}
{A=[B], B=[A]}

```

## 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.**

```
import java.util.*;
```

```

public class Graph {
    private int V;
    private LinkedList<Integer> adj[];

```

```

Graph(int v) {
    V = v;
    adj = new LinkedList[v];
    for (int i = 0; i < v; ++i)
        adj[i] = new LinkedList();
}

void addEdge(int v, int w) {
    adj[v].add(w);
}

void BFS(int s) {
    boolean visited[] = new boolean[V];

    LinkedList<Integer> queue = new LinkedList<>();

    visited[s] = true;
    queue.add(s);

    while (!queue.isEmpty()) {

        s = queue.poll();
        System.out.print(s + " ");

        Iterator<Integer> i = adj[s].listIterator();
    }
}

```

```
        while (i.hasNext()) {  
            int n = i.next();  
            if (!visited[n]) {  
                visited[n] = true;  
                queue.add(n);  
            }  
        }  
    }  
}
```

```
public static void main(String args[]) {  
    Graph g = new Graph(4);  
  
    g.addEdge(0, 1);  
    g.addEdge(0, 2);  
    g.addEdge(1, 2);  
    g.addEdge(2, 0);  
    g.addEdge(2, 3);  
    g.addEdge(3, 3);  
  
    System.out.println("Breadth First Traversal (starting from vertex 2):");  
    g.BFS(2);  
}
```

**OUTPUT:**

```
Console x
<terminated> Salary (1) [Java Application] C:\Users\udhya\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin\javaw.exe (03-Jun-2024, 6:28:49 pm - 6:28:50 pm) [pic
Breadth First Traversal (starting from vertex 2):
2 0 3 1
```

## 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.**

```
import java.util.*;
```

```
public class Graph {
    private int V;
    private LinkedList<Integer> adj[];

    Graph(int v) {
        V = v;
        adj = new LinkedList[V];
        for (int i = 0; i < v; ++i)
            adj[i] = new LinkedList();
    }

    void addEdge(int v, int w) {
        adj[v].add(w);
    }

    void DFSUtil(int v, boolean visited[]) {
        visited[v] = true;
        System.out.print(v + " ");
        Iterator<Integer> i = adj[v].listIterator();
        while (i.hasNext()) {
```

```

        int n = i.next();
        if (!visited[n])
            DFSUtil(n, visited);
    }
}

```

```

void DFS(int v) {

```

```

    boolean visited[] = new boolean[V];

```

```

    DFSUtil(v, visited);
}

```

```

public static void main(String args[]) {

```

```

    Graph g = new Graph(4);

```

```

    g.addEdge(0, 1);

```

```

    g.addEdge(0, 2);

```

```

    g.addEdge(1, 2);

```

```

    g.addEdge(2, 0);

```

```

    g.addEdge(2, 3);

```

```

    g.addEdge(3, 3);

```

```

    System.out.println("Depth First Traversal (starting from vertex 2):");

```

```

    g.DFS(2);

```

```

}

```

```

}

```



## OUTPUT:



```
Console x
<terminated> Salary (1) [Java Application] C:\Users\udhya\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin\javaw.exe (03-Jun-2024, 6:31:04 pm) [pl
Depth First Traversal (starting from vertex 2):
2 0 1 3
```

The screenshot shows an Eclipse IDE console window. The title bar indicates the application is "Salary (1) [Java Application]" and the path is "C:\Users\udhya\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86\_64\_17.0.11.v20240426-1830\jre\bin\javaw.exe". The console output displays the text "Depth First Traversal (starting from vertex 2):" followed by the sequence of vertices "2 0 1 3". The status bar at the bottom shows "Writable", "Smart Insert", and "13 : 11 : 277".