Name: Tejaswini Anil Kamble

Email : teju000kamble@gmail.com

Assignments : DAY 7 and 8

Task 1: Balanced Binary Tree Check

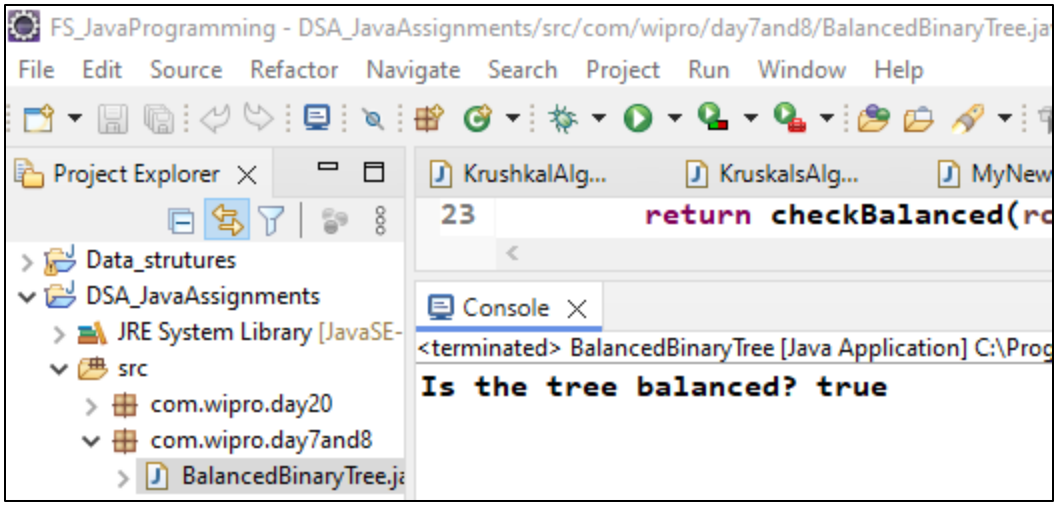
Write a function to check if a given binary tree is balanced. A balanced tree is one where the height of two subtrees of any node never differs by more than one.

Ans: Source Code

```
package com.wipro.day7and8;
class TreeNode
{
   int val;
   TreeNode left;
   TreeNode right;

   TreeNode(int val)
{
      this.val = val;
      this.left = null;
      this.right = null;
}
}
public class BalancedBinaryTree
{
   static class TreeInfo
{
      int height;
      boolean isBalanced;

      TreeInfo(int height, boolean isBalanced)
{
         this.height = height;
         this.isBalanced = isBalanced;
      }
```

```
    }
  public boolean isBalanced(TreeNode root)
   {
      return checkBalanced(root).isBalanced;
    }

    private TreeInfo checkBalanced(TreeNode node)
   {
        if (node == null) {
         return new TreeInfo(-1, true);
       }

       TreeInfo left = checkBalanced(node.left);
       TreeInfo right = checkBalanced(node.right);
       boolean isBalanced = left.isBalanced && right.isBalanced &&
        Math.abs(left.height - right.height) <= 1;
        int height = Math.max(left.height, right.height) + 1;

       return new TreeInfo(height, isBalanced);
    }

    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.right = new TreeNode(6);
       root.left.left.left = new TreeNode(7);

       BalancedBinaryTree tree = new BalancedBinaryTree();
       System.out.println("Is the tree balanced? " + tree.isBalanced(root));
    }
  }
```

Output:



Task 2: Trie for Prefix Checking

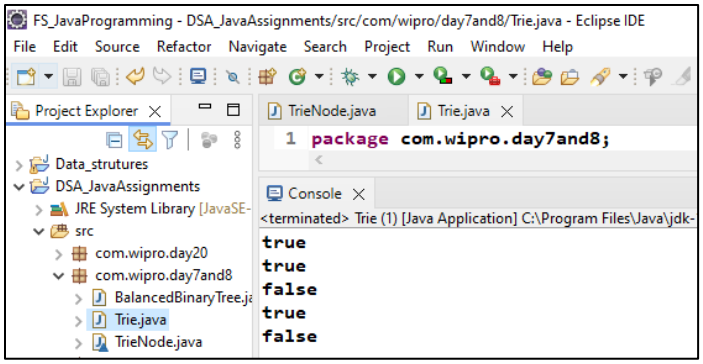
Implement a trie data structure in C# that supports insertion of strings and provides a method to check if a given string is a prefix of any word in the trie.

Ans :

```
package com.wipro.day7and8;
public class Trie
 {
   private final TrieNode root;

   public Trie()
   {
      root = new TrieNode();
   }

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

   public boolean isPrefix(String prefix) {
```

```
        TrieNode current = root;
        for (char c : prefix.toCharArray()) {
            if (!current.children.containsKey(c)) {
                return false;
            }
            current = current.children.get(c);
        }
        return true;
    }

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

        System.out.println(trie.isPrefix("app"));
        System.out.println(trie.isPrefix("ban"));
        System.out.println(trie.isPrefix("bat"));
        System.out.println(trie.isPrefix("appl"));
        System.out.println(trie.isPrefix("apx"));
    }
}
package com.wipro.day7and8;

import java.util.HashMap;
import java.util.Map;

class TrieNode
{
    Map<Character, TrieNode> children;
    boolean isEndOfWord;

    public TrieNode() {
        children = new HashMap<>();
        isEndOfWord = false;
    }
}
```

Output:



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.

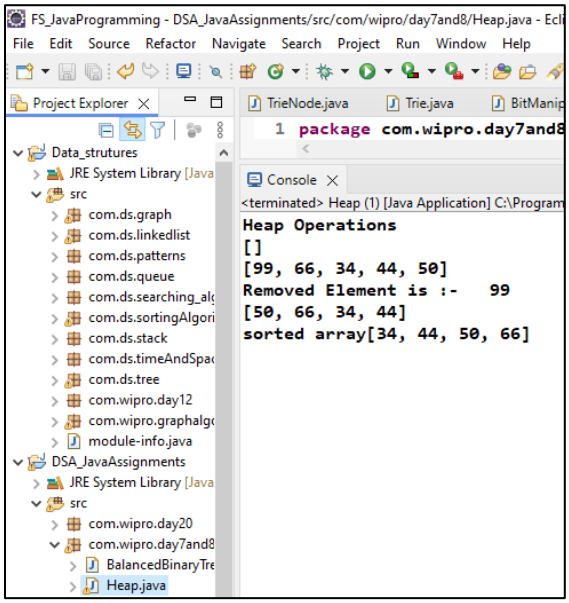
Ans : Source Code

```
package com.wipro.day7and8;

import java.util.ArrayList;

import java.util.Collection;
import java.util.Collections;
import java.util.List;

public class Heap {
	private List<Integer>heap;
	public Heap()
	{
		this.heap=new ArrayList<>();
	}
	public List<Integer> getheap()
	{
		return new ArrayList<Integer>(heap);
	}
	public int lefrchild(int index)
	{
		return (index*2)+2;

	}
	public int rightchild(int index)
	{
```

```
            return (index*2)+2;

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

        }
        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);
            }
        }
        private void swap(int index1, int index2) {
            int temp=heap.get(index1);
            heap.set(index1, heap.get(index2));
            heap.set(index2, temp);}
        public Integer remove()
        {   if(heap.size()==0)
            {
                return null;

            }
            if(heap.size()==1)
            {
                return heap.remove(0);

            }

            int maxvalue=heap.get(0);
            heap.set(0, heap.remove(heap.size()-1));
            sinkDown(0);
            return maxvalue;}
```

```
	private void sinkDown(int index) {
		int maxindex=index;
		int leftindex=lefrchild(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 List<Integer> heapSort() {
//		}

		Collections.sort(heap);
		return heap;
	}

	public static void main(String[] args) {

		Heap h=new Heap();
		System.out.println("Heap Operations");


		System.out.println(h.getheap());

	 h.insert(99);
	 h.insert(66);;
	 h.insert(34);
	 h.insert(44);
```

```
        h.insert(50);

        System.out.println(h.getheap());

        System.out.println("Removed Element is :-   "+h.remove());

        System.out.println(h.getheap());
System.out.println( "sorted array"+h.heapSort());
        }

}
```



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.

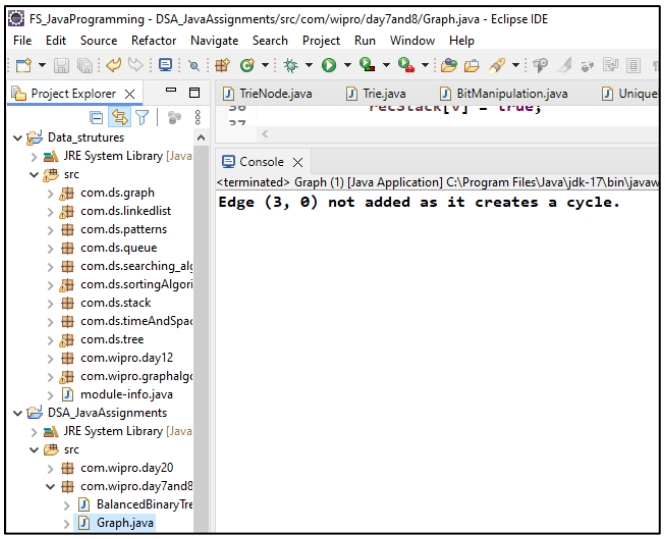
Ans: Source Code

```
package com.wipro.day7and8;
import java.util.*;

public class Graph
{
   private int V;
   private List<List<Integer>> adj;

   public Graph(int V) {
```

```
    this.V = V;
    adj = new ArrayList<>(V);
    for (int i = 0; i < V; i++) {
        adj.add(new ArrayList<>());
    }
}

public void addEdge(int u, int v) {
    adj.get(u).add(v);
}

public boolean isCyclic() {
    boolean[] visited = new boolean[V];
    boolean[] recStack = new boolean[V];

    for (int i = 0; i < V; i++) {
        if (isCyclicUtil(i, visited, recStack)) {
            return true;
        }
    }
    return false;
}

private boolean isCyclicUtil(int v, boolean[] visited, boolean[] recStack) {
    if (!visited[v]) {
        visited[v] = true;
        recStack[v] = true;

        List<Integer> neighbors = adj.get(v);
        for (Integer neighbor : neighbors) {
            if (!visited[neighbor] && isCyclicUtil(neighbor, visited, recStack)) {
                return true;
            } else if (recStack[neighbor]) {
                return true;
            }
        }
    }
    recStack[v] = false;
    return false;
}

public boolean addEdgeAndCheckCycle(int u, int v) {
    addEdge(u, v);
    if (isCyclic()) {
        adj.get(u).remove((Integer) v);
        return false;
```

```
        }
        return true;
    }

    public static void main(String[] args) {
        int V = 4;
        Graph graph = new Graph(V);
        graph.addEdge(0, 1);
        graph.addEdge(1, 2);
        graph.addEdge(2, 3);

        int u = 3;
        int v = 0;

        if (graph.addEdgeAndCheckCycle(u, v)) {
            System.out.println("Edge (" + u + ", " + v + ") added successfully without creating a cycle.");
        } else {
            System.out.println("Edge (" + u + ", " + v + ") not added as it creates a cycle.");
        }
    }
}
```

Output:



Task 5: Breadth-First Search (BFS) Implementation

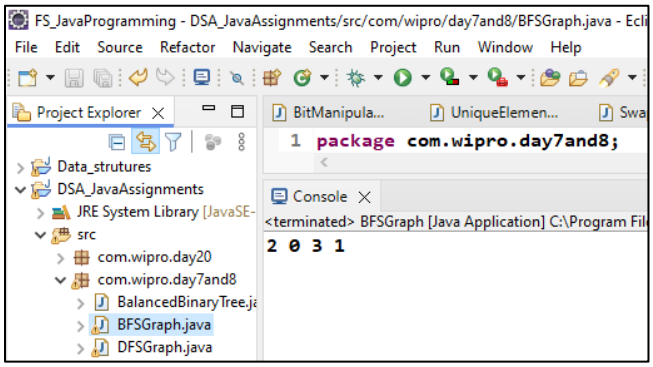
For a given undirected graph, implement BFS to traverse the graph starting from a given node and print each node in the order it is visited.

```
package com.wipro.day7and8;
```

```
import java.util.*;

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

   @SuppressWarnings("unchecked")
   public BFSGraph(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<Integer>();

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

      while (queue.size() != 0) {
         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[]) {
      BFSGraph g = new BFSGraph(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);

        g.BFS(2);
    }
}
```

Output:



Task 6: Depth-First Search (DFS) Recursive

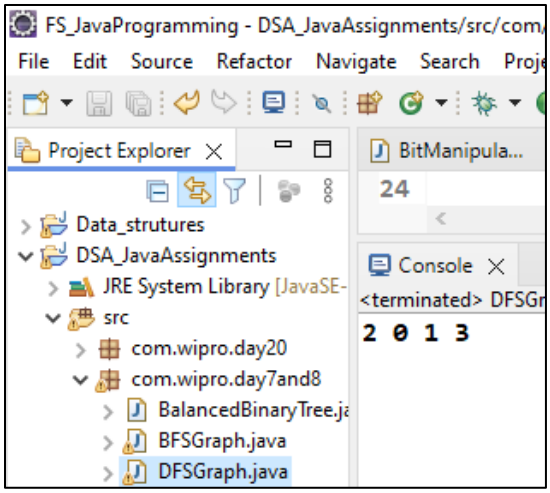
Write a recursive DFS function for a given undirected graph. The function should visit every node and print it out.

Ans:Source code

```
package com.wipro.day7and8;

import java.util.*;

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

    @SuppressWarnings("unchecked")
    public DFSGraph(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[]) {
        DFSGraph g = new DFSGraph(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);

        g.DFS(2);
    }
}
```

Output:


FS_JavaProgramming - DSA_JavaAssignments/src/com
File Edit Source Refactor Navigate Search Proje
Project Explorer
Data_strutures
DSA_JavaAssignments
JRE System Library [JavaSE-
src
com.wipro.day20
com.wipro.day7and8
BalancedBinaryTree.ja
BFSGraph.java
DFSGraph.java
BitManipula...
24
Console
<terminated> DFSGr
2 0 1 3