Assignment Day-7&8

Core Java with DS and Algorithms

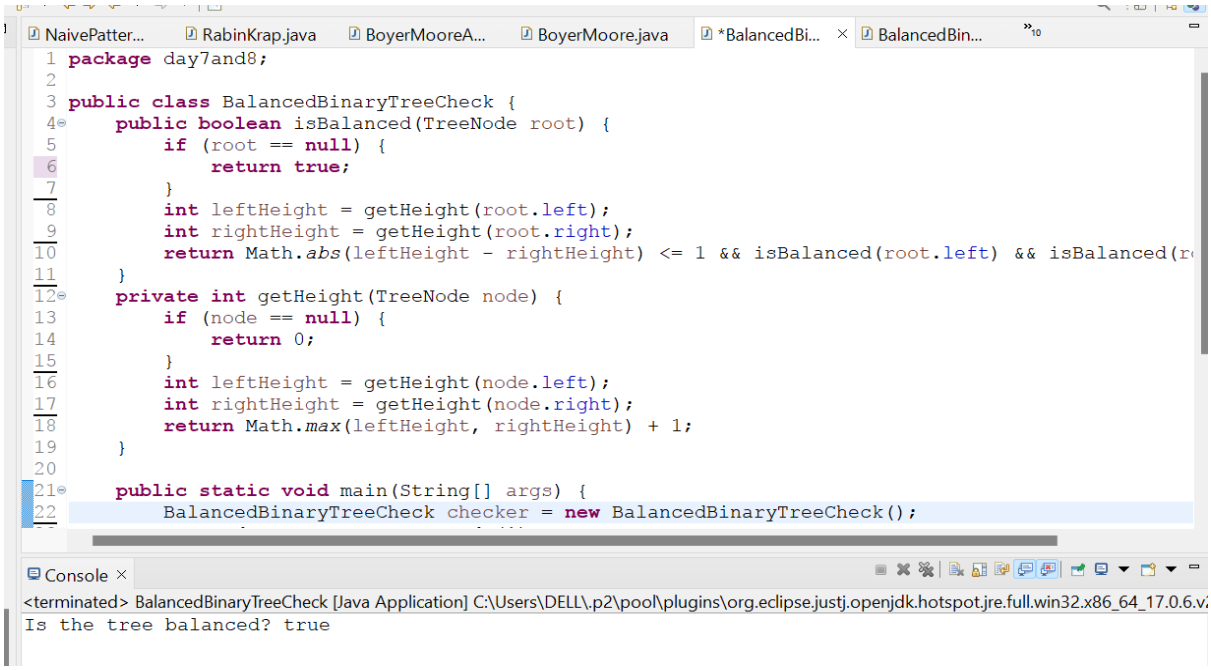
Name: Joshnitha Rangolu

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.

```
package day7and8;

public class BalancedBinaryTreeCheck {

public boolean isBalanced(TreeNode root) {

if (root == null) {

return true;

}

int leftHeight = getHeight(root.left);

int rightHeight = getHeight(root.right);

return Math.abs(leftHeight - rightHeight) <= 1 && isBalanced(root.left) &&
isBalanced(root.right);

}

private int getHeight(TreeNode node) {

if (node == null) {

return 0;

}

int leftHeight = getHeight(node.left);

int rightHeight = getHeight(node.right);

return Math.max(leftHeight, rightHeight) + 1;

}

public static void main(String[] args) {

BalancedBinaryTreeCheck checker = new BalancedBinaryTreeCheck();
```

```
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("Is the tree balanced? " + checker.isBalanced(root));

}

}
```



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.

```
package day7and8;

import java.util.HashMap;

public class TrieNode {

public boolean isWord;

public HashMap<Character, TrieNode> children;
```

```
public TrieNode() {
isWord = false;
children = new HashMap<>();
}
} public class Trie {
private TrieNode root;
public Trie() {
root = new TrieNode();
}
public void insert(String word) {
TrieNode current = root;
for (char ch : word.toCharArray()) {
if (!current.children.containsKey(ch)) {
current.children.put(ch, new TrieNode());
}
current = current.children.get(ch);
}
current.isWord = true;
}
public boolean isPrefix(String prefix) {
TrieNode current = root;
for (char ch : prefix.toCharArray()) {
if (!current.children.containsKey(ch)) {
return false;
}
current = current.children.get(ch);
}
return true;
```

```
}

}
```

Task 3: Implementing Heap Operations

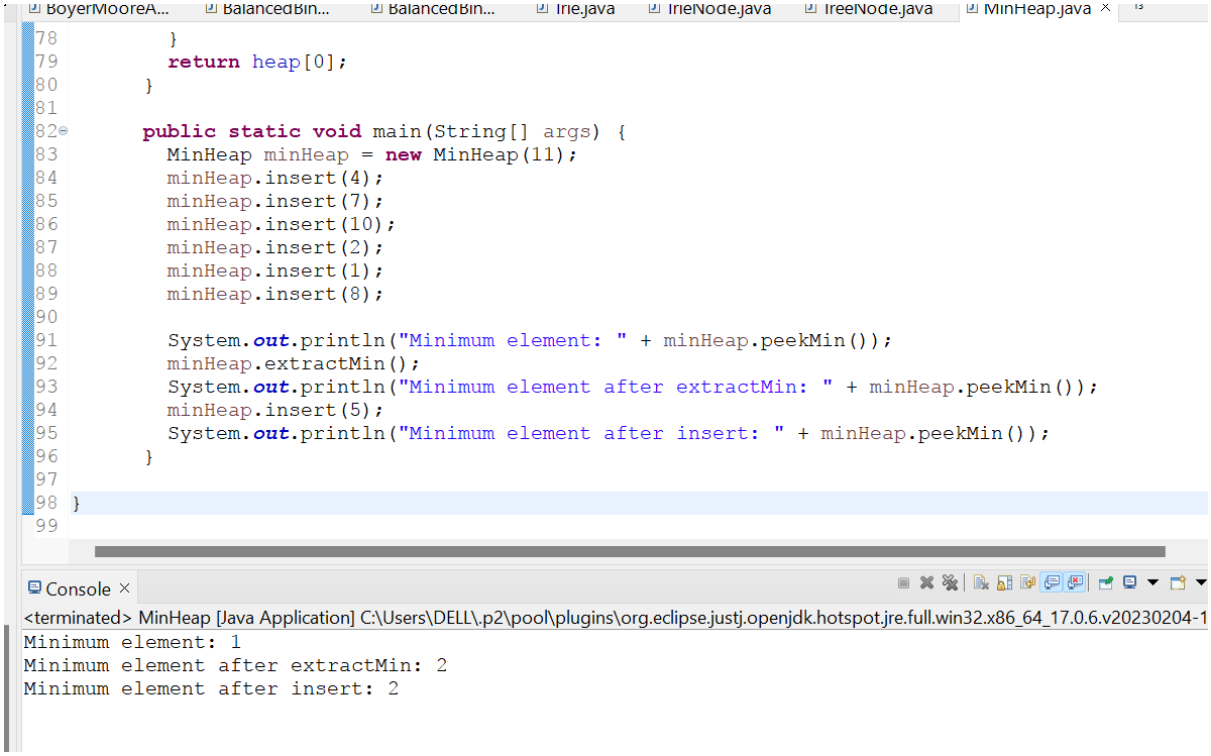
Code a min-heap in java with methods for insertion, deletion, and fetching the minimum element. Ensure that the heap property is maintained after each operation.

```
package day7and8;

public class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int val) {

this.val = val;

}

}

public class MinHeap {

private int capacity;

private int[] heap;

private int heapSize;

public MinHeap(int capacity) {

this.capacity = capacity;

heap = new int[capacity];

heapSize = 0;

}

private int parent(int i) {

return (i - 1) / 2;

}

private int left(int i) {

return 2 * i + 1;
```

```
}
private int right(int i) {
return 2 * i + 2;
}
private void swap(int i, int j) {
int temp = heap[i];
heap[i] = heap[j];
heap[j] = temp;
}
private void heapify(int i) {
int smallest = i;
int l = left(i);
int r = right(i);
if (l < heapSize && heap[l] < heap[smallest]) {
smallest = l;
}
if (r < heapSize && heap[r] < heap[smallest]) {
smallest = r;
}
if (smallest != i) {
swap(i, smallest);
heapify(smallest);
}
}
public void insert(int key) {
if (heapSize == capacity) {
System.out.println("Heap overflow");
return;
```

```
}
heap[heapSize] = key;
heapSize++;
int i = heapSize - 1;
while (i > 0 && heap[parent(i)] > heap[i]) {
swap(i, parent(i));
i = parent(i);
}
}
public int extractMin() {
if (heapSize == 0) {
System.out.println("Heap underflow");
return -1;
}
int min = heap[0];
if (heapSize == 1) {
heapSize--;
return min;
}
heap[0] = heap[heapSize - 1];
heapSize--;
heapify(0);
return min;
}
public int peekMin() {
if (heapSize == 0) {
System.out.println("Heap is empty");
return -1;
```

```
}

return heap[0];

}

public static void main(String[] args) {

MinHeap minHeap = new MinHeap(11);

minHeap.insert(4);

minHeap.insert(7);

minHeap.insert(10);

minHeap.insert(2);

minHeap.insert(1);

minHeap.insert(8);

System.out.println("Minimum element: " + minHeap.peekMin());

minHeap.extractMin();

System.out.println("Minimum element after extractMin: " +
minHeap.peekMin());

minHeap.insert(5);

System.out.println("Minimum element after insert: " + minHeap.peekMin());

}

}
```



Task 4: Graph Edge Addition Validation

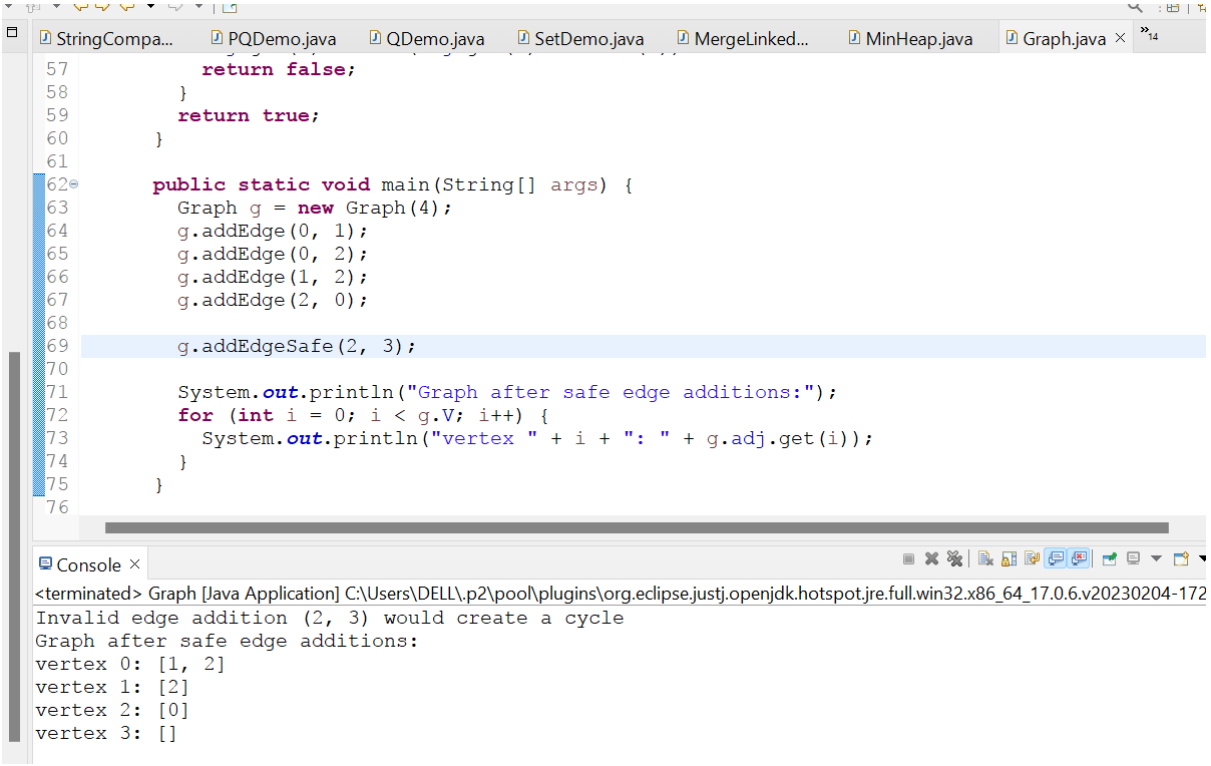
Given a directed graph, write a function that adds an edge between two nodes and then checks if the graph still has no cycles. If a cycle is created, the edge should not be added.

```
package day7and8;


import java.util.ArrayList;

import java.util.List;


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);
}

// Utility function for recursive DFS
private boolean isCyclicUtil(int v, boolean[] visited, boolean[] recStack) {
  if (visited[v]) return false; // Already visited

  visited[v] = true;
  recStack[v] = true;

  for (int neighbor : adj.get(v)) {
    if (!visited[neighbor] && isCyclicUtil(neighbor, visited, recStack)) {
      return true;
    } else if (recStack[neighbor]) {
      return true;
    }
  }

  recStack[v] = false;
  return false;
}

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

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

public boolean addEdgeSafe(int u, int v) {
  adj.get(u).add(v);
  if (isCyclic()) {
    System.out.println("Invalid edge addition (" + u + ", " + v + ") would create a cycle");
    adj.get(u).remove(adj.get(u).indexOf(v));
    return false;
  }
  return true;
}

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.addEdgeSafe(2, 3);

  System.out.println("Graph after safe edge additions:");
  for (int i = 0; i < g.V; i++) {
    System.out.println("vertex " + i + ": " + g.adj.get(i));
  }
```

```
        }

}
```



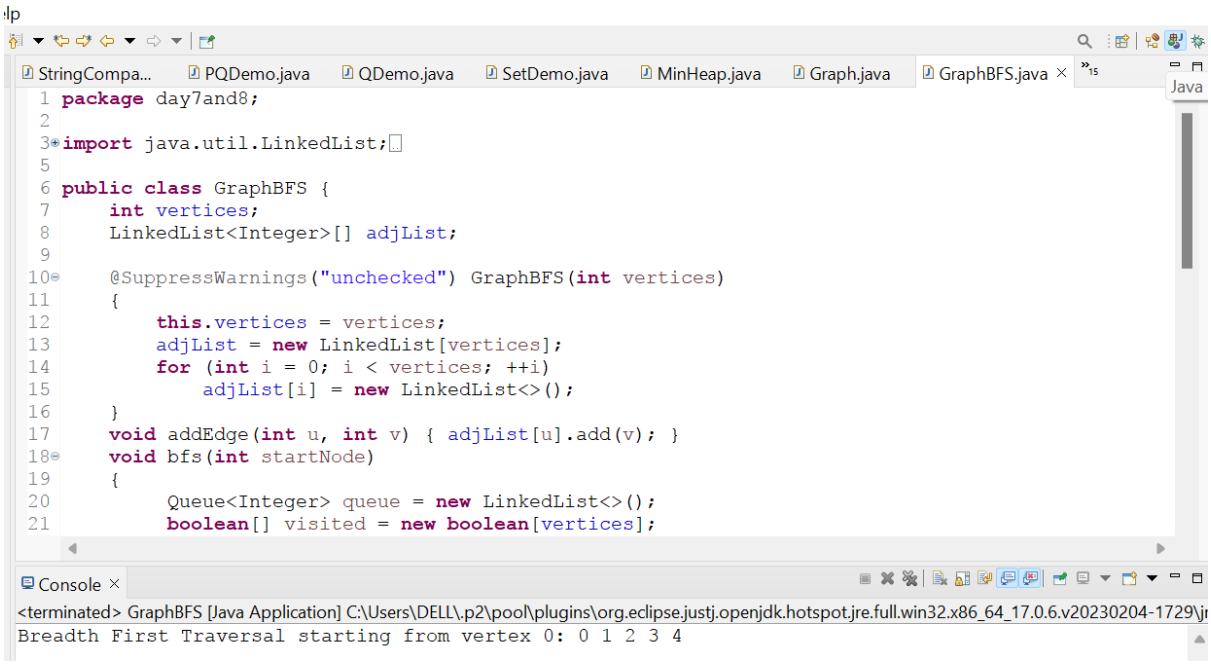
Task 5: Breadth-First Search (BFS) Implementation

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 day7and8;


import java.util.LinkedList;

import java.util.Queue;


public class GraphBFS {

        int vertices;

   LinkedList<Integer>[] adjList;
```

```
@SuppressWarnings("unchecked") GraphBFS(int vertices)
{
   this.vertices = vertices;
   adjList = new LinkedList[vertices];
   for (int i = 0; i < vertices; ++i)
      adjList[i] = new LinkedList<>();
}
void addEdge(int u, int v) { adjList[u].add(v); }
void bfs(int startNode)
{
        Queue<Integer> queue = new LinkedList<>();
    boolean[] visited = new boolean[vertices];
    visited[startNode] = true;
    queue.add(startNode);
    while (!queue.isEmpty()) {
       int currentNode = queue.poll();
       System.out.print(currentNode + " ");
       for (int neighbor : adjList[currentNode]) {
          if (!visited[neighbor]) {
             visited[neighbor] = true;
             queue.add(neighbor);
          }
       }
    }
 }

public static void main(String[] args)
{
   int vertices = 5;
   GraphBFS graph = new GraphBFS(vertices);
   graph.addEdge(0, 1);
```

```
        graph.addEdge(0, 2);
        graph.addEdge(1, 3);
        graph.addEdge(1, 4);
        graph.addEdge(2, 4);
        System.out.print(
            "Breadth First Traversal starting from vertex 0: ");
        graph.bfs(0);
    }


}
```



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.

```
package day7and8;


import java.util.Iterator;

import java.util.LinkedList;
```

```
public class GraphDFS {

        private int V;
          private LinkedList<Integer> adj[];


          @SuppressWarnings("unchecked") GraphDFS(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[])
{
    GraphDFS g = new GraphDFS(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);
}
}
```

StringCompa... PQDemo.java QDemo.java MinHeap.java Graph.java GraphBFS.java GraphDFS.java ×

```
1  package day7and8;
2
3  import java.util.Iterator;
5
6  public class GraphDFS {
7
8       private int V;
9          private LinkedList<Integer> adj[];
10
11         @SuppressWarnings("unchecked") GraphDFS(int v)
12         {
13             V = v;
14             adj = new LinkedList[v];
15             for (int i = 0; i < v; ++i)
16                 adj[i] = new LinkedList();
17         }
18         void addEdge(int v, int w)
19         {
20             adj[v].add(w);
21         }
```

Console ×

<terminated> GraphDFS [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v

```
Depth First Traversal (starting from vertex 2)
2 0 1 3
```