Day 7 and 8

1] 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.

Solution:-

```
☑ BalancedBinaryTree.java ×
 1 package com.assignment;
 3 class TreeNode {
            int val;
 4
 5
            TreeNode left;
            TreeNode right;
 6
 7
 89
            TreeNode(int x) {
 9
                val = x;
            }
10
11
        }
12
        public class BalancedBinaryTree{
13
14
15⊕
            public boolean isBalanced(TreeNode root) {
16
                return checkBalance(root) != -1;
17
18
            private int checkBalance(TreeNode node) {
199
20
                if (node == null) {
21
                     return 0;
                }
22
23
                int leftHeight = checkBalance(node.left);
24
```

```
■ BalancedBinaryTree.java ×
22
                }
23
                int leftHeight = checkBalance(node.left);
24
25
                if (leftHeight == -1) {
                    return -1;
26
                }
27
28
29
                int rightHeight = checkBalance(node.right);
                if (rightHeight == -1) {
30
31
                    return -1;
                }
32
33
                if (Math.abs(leftHeight - rightHeight) > 1) {
34
35
                    return -1;
                }
36
37
                return Math.max(leftHeight, rightHeight) + 1;
38
            }
39
40
            public static void main(String[] args) {
419
42
43
                TreeNode root = new TreeNode(1);
44
                root.left = new TreeNode(2);
                root.right = new TreeNode(3);
45
```

```
☑ BalancedBinaryTree.java ×
                     recurn -1,
32
                }
33
                if (Math.abs(leftHeight - rightHeight) > 1) {
34
35
                     return -1;
                }
36
37
                return Math.max(leftHeight, rightHeight) + 1;
38
            }
39
40
            public static void main(String[] args) {
419
42
43
                TreeNode root = new TreeNode(1);
44
                root.left = new TreeNode(2);
45
                root.right = new TreeNode(3);
                root.left.left = new TreeNode(4);
46
                root.left.right = new TreeNode(5);
47
48
                BalancedBinaryTree sol = new BalancedBinaryTree();
49
                System.out.println(sol.isBalanced(root));
50
            }
51
52
53
        }
```

```
■ Console ×

<terminated> BalancedBinaryTree [Java Application] C:\Program Files

true
```

2] 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.

Solution:-

```
☑ Tries,java X
  1 package com.wipro.nonlinear;
  3 import java.util.HashMap;
  4 import java.util.Map;
  5
  6 public class Tries {
        private final TrieNode root;
  7
  8
       class TrieNode {
  90
             Map<Character, TrieNode> children;
 10
             boolean endOfWord;
 11
 12
 13
             public TrieNode() {
                 children = new HashMap<>();
 14
                 endOfWord = false;
 15
             }
 16
        }
 17
 18
        public Tries() {
 19
             root = new TrieNode();
 20
        }
 21
```

```
☑ Tries.java ×
 199
         public Tries() {
 20
             root = new TrieNode();
 21
 22
 239
         public void insert(String word) {
 24
             TrieNode current = root;
             for (int i = 0; i < word.length(); i++) {
 25
 26
                  char ch = word.charAt(i);
 27
                  TrieNode node = current.children.get(ch);
 28
                  if (node == null) {
 29
                      node = new TrieNode();
 30
                      current.children.put(ch, node);
 31
 32
                 current = node;
 33
             }
 34
             current.endOfWord = true;
 35
         }
 36
 37⊖
         private void collectWords(TrieNode current, StringBuilder prefix) {
 38
             if (current.endOfWord) {
                  System.out.println(prefix.toString());
 39
☑ Tries.java ×
        private void collectWords(TrieNode current, StringBuilder prefix) {
 379
             if (current.endOfWord) {
 38
 39
                 System.out.println(prefix.toString());
 40
        for(Map.Entry<Character, TrieNode> entry : current.children.entrySet()) {
 41
 42
                 prefix.append(entry.getKey());
                 collectWords(entry.getValue(), prefix);
 43
 44
                 prefix.deleteCharAt(prefix.length() - 1);
 45
             }
 46
        }
 47
 489
        public void printAllWords() {
 49
             collectWords(root, new StringBuilder());
 50
        }
 51
        public boolean search(String word) {
 52⊖
 53
             TrieNode current = root;
 54
             for (char c : word.toCharArray()) {
 55
                 TrieNode node = current.children.get(c);
 56
                 if (node == null) {
 57
                     return false;
 FO
```

```
☑ Tries.java X
 55
                TrieNode node = current.children.get(c);
                if (node == null) {
 56
                    return false;
 57
 58
                }
 59
                current = node;
 60
            }
 61
            return current.endOfWord;
        }
 62
 63
 649
        public static void main(String[] args) {
 65
            Tries trie = new Tries();
 66
            trie.insert("Hot");
 67
            trie.insert("Hope");
 68
 69
            System.out.println("Inserted words are:");
 70
 71
            trie.printAllWords();
 72
 73
            System.out.println();
 74
            System.out.println("Search results:");
 75

☑ Tries.java ×
              return current.endOfWord;
 61
         }
 62
 63
         public static void main(String[] args) {
 64⊕
              Tries trie = new Tries();
 65
 66
 67
              trie.insert("Hot");
              trie.insert("Hope");
 68
 69
 70
              System.out.println("Inserted words are:");
 71
              trie.printAllWords();
 72
              System.out.println();
 73
 74
              System.out.println("Search results:");
 75
              System.out.println("house: " + trie.search("house"));
 76
              System.out.println("Hot: " + trie.search("Hot"));
 77
 78
         }
 79 }
```



house: false Hot: true

3] 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.

Solution:-Code -

```
☑ MinHeap.java ×
  1 package com.assignment;
3 import java.util.ArrayList;
  4
  5
        public class MinHeap {
  6
             private ArrayList<Integer> heap;
  7
  89
             public MinHeap() {
  9
                 heap = new ArrayList<>();
 10
 11
 129
             public void insert(int value) {
 13
                 heap.add(value);
 14
                 heapifyUp(heap.size() - 1);
 15
             }
 16
17⊕
             public int deleteMin() {
                 if (heap.isEmpty()) {
 18
 19
                      throw new IllegalStateException("Heap is empty");
 20

☑ MinHeap.java ×
16
179
           public int deleteMin() {
18
                if (heap.isEmpty()) {
                    throw new IllegalStateException("Heap is empty");
19
20
               int min = heap.get(0);
21
                int last = heap.remove(heap.size() - 1);
22
23
               if (!heap.isEmpty()) {
24
                    heap.set(0, last);
25
                   heapifyDown(0);
26
27
               return min;
28
           }
29
309
           public int getMin() {
31
               if (heap.isEmpty()) {
                    throw new IllegalStateException("Heap is empty");
32
33
34
               return heap.get(0);
35
           }
36
37⊕
           private void heapifyUp(int index) {
                int parentIndex = (index - 1) / 2;
38
               while (index > 0 && heap.get(index) < heap.get(parentIndex)) {</pre>
39
```

```
☑ MinHeap.java ×
34
                return heap.get(0);
35
            }
36
37⊜
            private void heapifyUp(int index) {
                int parentIndex = (index - 1) / 2;
38
                while (index > 0 && heap.get(index) < heap.get(parentIndex)) {</pre>
39
40
                    swap(index, parentIndex);
41
                    index = parentIndex;
42
                    parentIndex = (index - 1) / 2;
43
                }
44
            }
45
46⊜
            private void heapifyDown(int index) {
47
                int smallest = index;
                int leftChildIndex = 2 * index + 1;
48
                int rightChildIndex = 2 * index + 2;
49
50
             if (leftChildIndex < heap.size() && heap.get(leftChildIndex) < heap.get(smallest)) {</pre>
51
                     smallest = leftChildIndex;
52
53
                if (rightChildIndex < heap.size() && heap.get(rightChildIndex) < heap.get(smallest)) {
54
55
                    smallest = rightChildIndex;
56
                }
57

☑ MinHeap.java ×
52
                 }
53
54
                 if (rightChildIndex < heap.size() && heap.get(rightChildIndex) < heap.get(smallest)) {</pre>
55
                     smallest = rightChildIndex;
 56
                 }
57
58
                 if (smallest != index) {
 59
                     swap(index, smallest);
                     heapifyDown(smallest);
 60
 61
                 }
            }
62
 63
 649
            private void swap(int index1, int index2) {
 65
                 int temp = heap.get(index1);
                heap.set(index1, heap.get(index2));
heap.set(index2, temp);
 66
 67
 68
 69
 709
            public static void main(String[] args) {
 71
                 MinHeap minHeap = new MinHeap();
 72
 73
                 minHeap.insert(3);
 74
                minHeap.insert(1);
75
                minHeap.insert(6);
```

```
☑ MinHeap.java ×
            privace votu swap(int inuexi, int inuexi) (
 65
                int temp = heap.get(index1);
66
                heap.set(index1, heap.get(index2));
 67
                heap.set(index2, temp);
 68
 709
            public static void main(String[] args) {
                MinHeap minHeap = new MinHeap();
 71
 72
 73
                minHeap.insert(3);
 74
                minHeap.insert(1);
 75
                minHeap.insert(6);
 76
                minHeap.insert(5);
 77
                minHeap.insert(2);
 78
                minHeap.insert(4);
 79
                System.out.println("Min value: " + minHeap.getMin());
 80
                System.out.println("Deleted min value: " + minHeap.deleteMin());
 81
 82
                System.out.println("New min value: " + minHeap.getMin());
 83
            }
 84
        }
 85
 86
87
```

```
© Console X
<terminated> MinHeap [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\jav
Min value: 1
Deleted min value: 1
New min value: 2
```

4] 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.

Solution:-

```
☑ GraphEdgeAdd.java ×
  1 package com.assignment;
  2 import java.util.*;
  3 public class GraphEdgeAdd {
  4
               private Map<Integer, List<Integer>> adjList;
  5
  6
  79
               public GraphEdgeAdd() {
  8
                    adjList = new HashMap<>();
  9
               }
 10
               public void addNode(int node) {
 110
                    adjList.putIfAbsent(node, new ArrayList<>());
 12
 13
 14
 159
               public boolean addEdge(int from, int to) {
                    addNode(from);
 16
 17
                    addNode(to);
                    adjList.get(from).add(to);
 18
 19
 20
                    if (hasCycle()) {
 21
                         adjList.get(from).remove((Integer) to);
 22
                         return false;
 23
 24
                    return true;

☑ GraphEdgeAdd.java ×

              II (Hascycie()) \
                 adjList.get(from).remove((Integer) to);
21
22
                 return false;
 23
             }
 24
             return true;
 25
 26
 27⊖
          private boolean hasCycle() {
             Set<Integer> visited = new HashSet<>();
 29
             Set<Integer> recStack = new HashSet<>();
 30
 31
             for (Integer node : adjList.keySet()) {
                 if (hasCycleUtil(node, visited, recStack)) {
 32
 33
                    return true;
                 }
 34
             }
 36
             return false;
 37
38⊖
          private boolean hasCycleUtil(int node, Set<Integer> visited, Set<Integer> recStack) {
 39
             if (recStack.contains(node)) {
 40
                 return true;
41
42
             if (visited.contains(node)) {
43
                 return false;
44
             }
```

```
☑ GraphEdgeAdd.java ×
42
               if (visited.contains(node)) {
43
                    return false;
44
               }
45
46
               visited.add(node);
47
               recStack.add(node);
48
49
               List<Integer> neighbors = adjList.get(node);
               if (neighbors != null) {
50
                    for (Integer neighbor : neighbors) {
51
52
                        if (hasCycleUtil(neighbor, visited, recStack)) {
53
                            return true;
54
                        }
55
                    }
56
               }
57
58
               recStack.remove(node);
59
               return false;
           }
60
61
629
           public static void main(String[] args) {
63
               GraphEdgeAdd graph = new GraphEdgeAdd();
64

☑ GraphEdgeAdd.java ×
                              (Hascycleucti(Heighbol, Visited, Fecstack)) (
53
                               return true;
54
                          }
55
                     }
56
                 }
57
58
                 recStack.remove(node);
59
                 return false;
60
             }
61
629
             public static void main(String[] args) {
                 GraphEdgeAdd graph = new GraphEdgeAdd();
63
64
65
66
                 System.out.println(graph.addEdge(1, 2));
67
                 System.out.println(graph.addEdge(2, 3));
68
                 System.out.println(graph.addEdge(3, 4));
69
                 System.out.println(graph.addEdge(4, 1));
70
                 System.out.println(graph.addEdge(4, 5));
71
             }
72
        }
73
74
```



5] 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.

Solution:-

```
☑ BFSImplimentation.java ×
 1 package com.assignment;
 3 import java.util.*;
 4
 5
        public class BFSImplimentation {
            private Map<Integer, List<Integer>> adjList;
 6
 7
            public BFSImplimentation() {
 89
 9
                 adjList = new HashMap<>();
10
            }
11
129
            public void addNode(int node) {
13
                 adjList.putIfAbsent(node, new ArrayList<>());
14
            }
15
169
            public void addEdge(int node1, int node2) {
17
                 addNode(node1);
18
                 addNode(node2);
19
                 adjList.get(node1).add(node2);
20
                 adjList.get(node2).add(node1);
            }
21
22
239
            public void bfs(int startNode) {
24
                 Set<Integer> visited = new HashSet<>();

☑ BFSImplimentation.java ×
22
23⊜
           public void bfs(int startNode) {
24
                Set<Integer> visited = new HashSet<>();
25
                Queue<Integer> queue = new LinkedList<>();
26
27
                visited.add(startNode);
28
                queue.add(startNode);
29
30
               while (!queue.isEmpty()) {
31
                    int currentNode = queue.poll();
                    System.out.print(currentNode + " ");
32
33
34
                    for (int neighbor : adjList.get(currentNode)) {
35
                        if (!visited.contains(neighbor)) {
36
                            visited.add(neighbor);
37
                            queue.add(neighbor);
38
                        }
                   }
39
40
               }
41
           }
42
43⊖
           public static void main(String[] args) {
44
                BFSImplimentation graph = new BFSImplimentation();
```

45

```
■ BFSImplimentation.java ×
37
                             queue.add(neighbor);
                         }
38
39
                     }
                }
40
            }
41
42
43⊖
            public static void main(String[] args) {
44
                 BFSImplimentation graph = new BFSImplimentation();
45
46
47
                 graph.addEdge(1, 2);
48
                 graph.addEdge(1, 3);
                graph.addEdge(2, 4);
49
50
                 graph.addEdge(3, 5);
51
                 graph.addEdge(4, 5);
                 graph.addEdge(5, 6);
52
53
54
                System.out.print("BFS starting from node 1: ");
55
56
                graph.bfs(1);
57
            }
58
        }
59
```

```
☐ Console ×

<terminated> BFSImplimentation [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\java BFS starting from node 1: 1 2 3 4 5 6
```

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

Solution:-

```
☑ DFSImplimentation.java ×
  1 package com.assignment;
 2 import java.util.*;
 3
 4
        public class DFSImplimentation {
            private Map<Integer, List<Integer>> adjList;
  5
 6
            public DFSImplimentation () {
 7⊕
                adjList = new HashMap<>();
 8
 9
 10
            public void addNode(int node) {
 119
 12
                adjList.putIfAbsent(node, new ArrayList<>());
 13
            }
 14
            public void addEdge(int node1, int node2) {
 15⊖
 16
                addNode(node1);
 17
                addNode(node2);
                adjList.get(node1).add(node2);
 18
                adjList.get(node2).add(node1);
 19
            }
 20
 21
            public void dfs(int startNode) {
 220
 23
                Set<Integer> visited = new HashSet<>();
                dfsRecursive(startNode, visited);
 24
```

```
☑ DFSImplimentation.java ×
229
            public void dfs(int startNode) {
23
                Set<Integer> visited = new HashSet<>();
24
                dfsRecursive(startNode, visited);
25
            }
26
279
            private void dfsRecursive(int node, Set<Integer> visited) {
                if (visited.contains(node)) {
28
29
                    return;
30
                }
31
32
                visited.add(node);
                System.out.print(node + " ");
33
34
                for (int neighbor : adjList.get(node)) {
35
36
                    dfsRecursive(neighbor, visited);
37
                }
38
            }
39
409
            public static void main(String[] args) {
41
                DFSImplimentation dfs = new DFSImplimentation ();
42
43
44
                dfs.addEdge(1, 2);
45
                dfs.addEdge(1, 3);

    □ DFSImplimentation.java ×
                 IOI (THE HETRIDOL . aultist . get(Houe)) /
                     dfsRecursive(neighbor, visited);
36
37
                 }
            }
38
39
409
             public static void main(String[] args) {
41
                 DFSImplimentation dfs = new DFSImplimentation ();
42
43
44
                 dfs.addEdge(1, 2);
45
                 dfs.addEdge(1, 3);
46
                 dfs.addEdge(2, 4);
47
                 dfs.addEdge(3, 5);
48
                 dfs.addEdge(4, 5);
49
                 dfs.addEdge(5, 6);
50
51
52
                 System.out.print("DFS starting from node 1: ");
                 dfs.dfs(1);
53
54
            }
        }
55
56
57
```

	•									
	onsole ×	plimenta	tion [Jav	а Арр	olica	atio	n] (:\P	rog	ram Files\Java\jdk-17.0.1\
DFS	starting	from	node	1:	1	2	4	5	3	6