Experiment 6

Student Name: Zatch **UID:**

Branch: CSE **Section/Group:**

Semester:5 **Date of Performance:**

Subject Name: AP **Subject Code: 22CSP-314**

1. **Aim:**

   Problem Statement: - You are given a tree (a simple connected graph with no cycles). Find the maximum number of edges you can remove from the tree to get a forest such that each connected component of the forest contains an even number of nodes.

2. **Objective:**

   The objective of this experiment is to determine how many ways you can remove an edge from a tree to create two separate subtrees such that both resulting subtrees have an even number of nodes.

3. **Implementation/Code:**

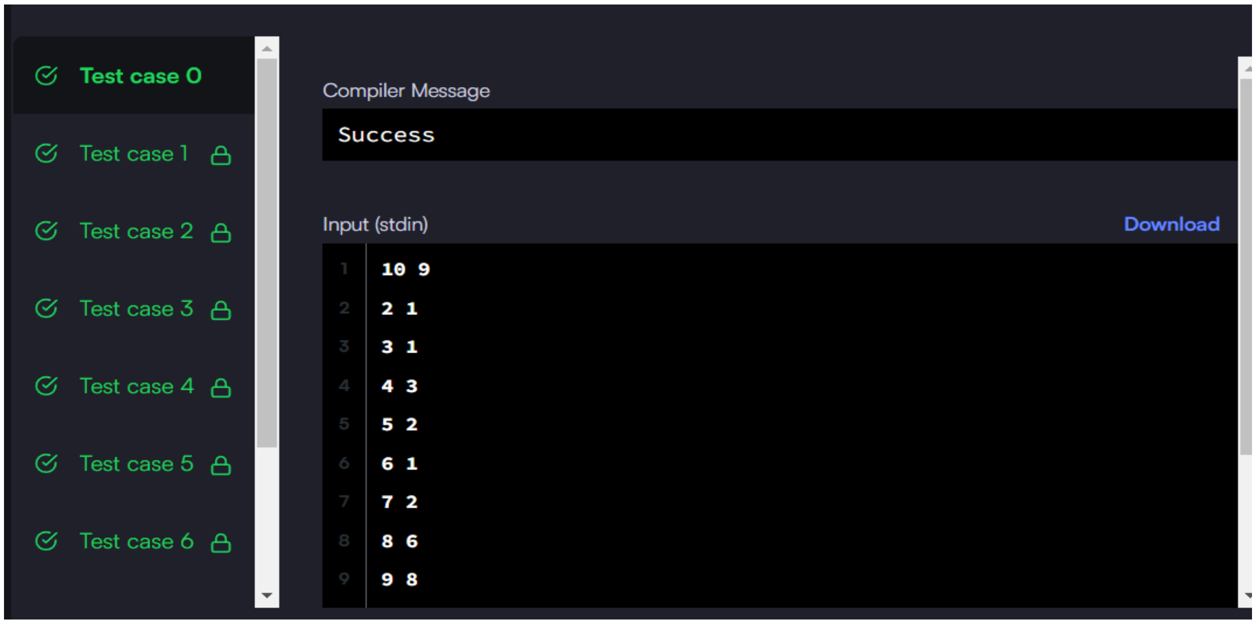
```
public class Solution {
    // Function to compute the number of removable edges
    static int evenForest(int t_nodes, int t_edges, List<Integer> t_from,
List<Integer> t_to) {
        // Create an adjacency list
        Map<Integer, List<Integer>> adjacencyList = new HashMap<>();
        for (int i = 1; i <= t_nodes; i++) {
            adjacencyList.put(i, new ArrayList<>());
        }
        for (int i = 0; i < t_edges; i++) {
            int u = t_from.get(i);
            int v = t_to.get(i);
            adjacencyList.get(u).add(v);
            adjacencyList.get(v).add(u);
        }
        // Array to store the size of each subtree
        int[] subtreeSize = new int[t_nodes + 1];
        boolean[] visited = new boolean[t_nodes + 1];

        // Start DFS from node 1 (root)
        dfs(1, adjacencyList, subtreeSize, visited);

        // Count the number of edges that can be removed
        int removableEdges = 0;
        for (int i = 2; i <= t_nodes; i++) {
            if (subtreeSize[i] % 2 == 0) {
                removableEdges++;
            }
        }
        return removableEdges;
    }
    // DFS to calculate subtree sizes
    private static void dfs(int node, Map<Integer, List<Integer>> adjacencyList, int[] subtreeSize, boolean[] visited) {
        visited[node] = true;
        subtreeSize[node] = 1;

        for (int neighbor : adjacencyList.get(node)) {
            if (!visited[neighbor]) {
                dfs(neighbor, adjacencyList, subtreeSize, visited);
                subtreeSize[node] += subtreeSize[neighbor];
            }
        }
    }
```

4. Output:



Compiler Message

```
Success
```

Input (stdin)

```
10 9
2 1
3 1
4 3
5 2
6 1
7 2
8 6
9 8
```

Problem -2

1. **Aim:**

   Problem Statement: - You are given a pointer to the root of a binary search tree and values to be inserted into the tree. Insert the values into their appropriate position in the binary search tree and return the root of the updated binary tree. You just have to complete the function.

2. **Objective:**

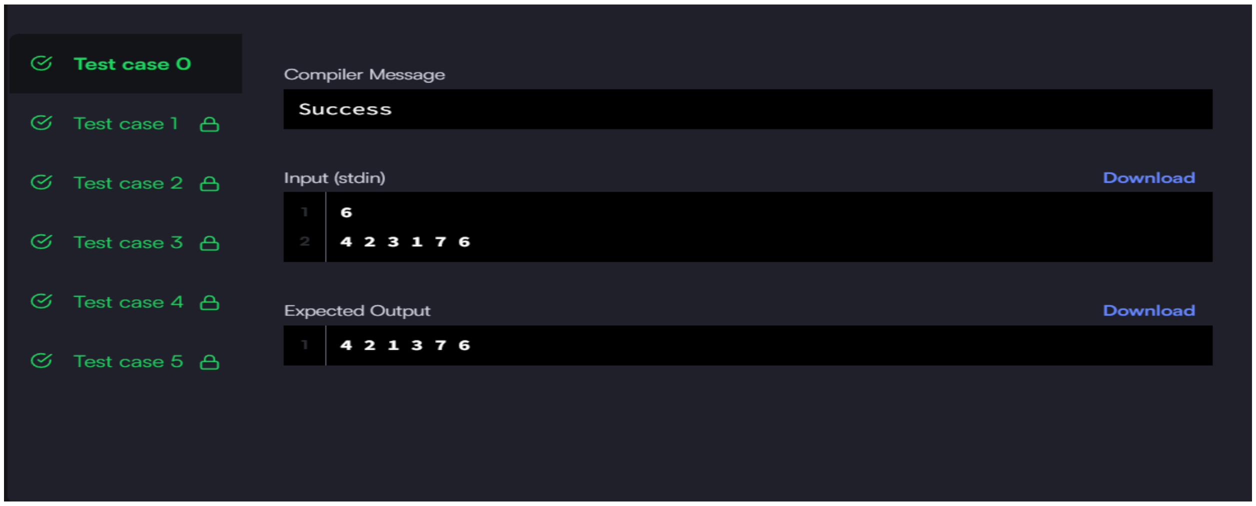
   The objective of this experiment is to implement a function in Java that determines the minimum number of characters needed to make a given password "strong" according to specific criteria.

3. **Code:**

```
public static Node insert(Node root,int data) {

// If the tree is empty, create a new node and return it
        if (root == null) {
            return new Node(data);
        }
        // Otherwise, recur down the tree and insert the node
        if (data <= root.data) {
            root.left = insert(root.left, data);
        } else {
            root.right = insert(root.right, data);
        }
        // Return the (unchanged) node pointer
        return root;
}
```

4. Output:



Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5

Compiler Message

```
Success
```

Input (stdin) Download

```
6
4 2 3 1 7 6
```

Expected Output Download

```
4 2 1 3 7 6
```

5. Learning Outcome

i. Enhance problem-solving skills by breaking down a complex problem into smaller, more manageable components (like node insertion and traversal)

ii. Apply concepts of graph theory to solve problems involving trees and subtrees.

iii. Develop algorithms to process and analyze tree structures, specifically focusing on subtree properties and even node counts.

iv. Gain experience in designing recursive functions for common tree operations such as insertion and traversal.

v. Learn the properties of BSTs and how to maintain these properties during insertion operations.