Experiment 6 (Trees)

Student Name: SANSKAR AGRAWAL **UID: 20BCS5914**

Branch: BE CSE **Section/Group: 806/B**

Semester: 5th **Date of performance: 18.10.22**

Subject: Competitive Coding **Subject Code: 20CSP_314**

1. Aim/Overview of the Practical:

a. Tree Huffman Decoding.
b. Balanced Forest.

2. Task to be done / Which logistics used:

a. You are given pointer to the root of the Huffman tree and a binary coded string to decode. You need to print the decoded string. Complete the function decode_huff in the editor below. It must return the decoded string.
decode_huff has the following parameters:
(i) root: a reference to the root node of the Huffman tree.
(ii) s: a Huffman encoded string

b. Complete the balancedForest function in the editor below. It must return an integer representing the minimum value of c[w] that can be added to allow creation of a balanced forest, or -1 if it is not possible.
balancedForest has the following parameters:
(i) c: an array of integers, the data values for each node
(ii) edges: an array of 2 element arrays, the node pairs per edge.

Steps for experiment/practical/Code:

a. Tree Huffman Decoding:

```
void decode(String s, Node root) {
    Node temp=root;
    String ans="";
    for(int i=0;i<s.length();i++){
     //  System.out.println("er1");
        if(s.charAt(i)=='0')
            temp=temp.left;
        else
            temp=temp.right;
        if(temp.right==null && temp.left==null)
            {
            ans+=(temp.data);
            temp=root;
        }
    }
    System.out.println(ans);


}
```

b. Balanced Forest:

```
import  java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;

public class Solution {

    private static Scanner scn;

    private static int n;
```

```
private static long ret;

private static int[] c, p;

private static long[] s;

private static List<Integer>[] adj;

private static void visit(int k, int i) {
    s[i] = c[i];
    for (int j : adj[i]) {
        if (j == k) {
            continue;
        }
        p[j] = i;
        visit(i,  j);
        s[i] += s[j];
    }
}

private static void check(long x, long y, long z) {
    long[] t = new long[] {x, y, z};
    for (int i = 0; i < 3; i++) {
        for (int j = i + 1; j < 3; j++) {
            if (t[i] != t[j]) {
                continue;
            }
            long h = -t[i] + -t[j] + t[0] + t[1] + t[2];
            if (h <= t[i]) {
                if (ret < 0) {
                    ret = t[i] - h;
                } else {
                    ret = Math.min(ret, t[i] - h);
                }
            }
        }
    }
}

private static void solve() {
```

```
ret = -1;
n = scn.nextInt();
c = new int[n];
s = new long[n];
adj = new List[n];
p = new int[n];
Arrays.fill(p, -1);
for (int i = 0; i < n; ++i) {
   c[i] = scn.nextInt();
   adj[i] = new ArrayList<Integer>();
}
for (int i = 0; i < n - 1; i++) {
   int x = scn.nextInt();
   int y = scn.nextInt();
   x--;
   y--;
   adj[x].add(y);
   adj[y].add(x);
}
visit(-1, 0);
Map<Long, Set<Integer>> sSet = new HashMap<Long, Set<Integer>>();
for (int i = 0; i < n; ++i) {
   if (sSet.containsKey(s[i])) {
      if (s[i] * 3 >= s[0]) {
         long h = s[i] * 3 - s[0];
         if (ret < 0) {
            ret = h;
         } else {
            ret = Math.min(ret, h);
         }
      }
   }
   Set<Integer> si = sSet.get(s[i]);
   if (si == null) {
      si = new HashSet<Integer>();
   }
   si.add(i);
   sSet.put(s[i], si);
}
for (int i = 0; i < n; ++i) {
```

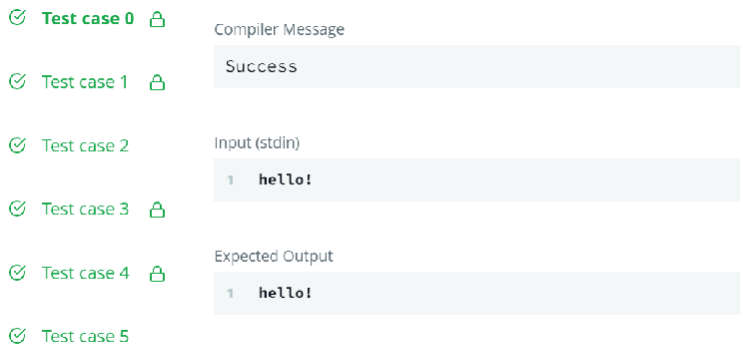
```
            if (s[i] * 3 < s[0] || s[i] * 2 > s[0]) {
                continue;
            }
            long t = s[0] - s[i] * 2;
            Set<Integer> si = sSet.get(t);
            if (si == null) {
                continue;
            }
            for (int j : si) {
                int k = j;
                boolean ok = true;
                while (k >= 0) {
                    if (k == i) {
                        ok = false;
                        break;
                    }
                    k = p[k];
                }
                if (ok) {
                    long h = s[i] * 3 - s[0];
                    if (ret < 0) ret = h;
                    else ret = Math.min(ret, h);
                }
            }
        }
        for (int i = 0; i < n; ++i) {
            int j = i;
            while (j >= 0) {
                j = p[j];
                if (j >= 0) {
                    check(s[i], s[j] - s[i], s[0] - s[j]);
                }
            }
        }
        System.out.println(ret);
    }

    public static void main(String[] args) {
        scn = new Scanner(System.in);
        int nTest = scn.nextInt();
```

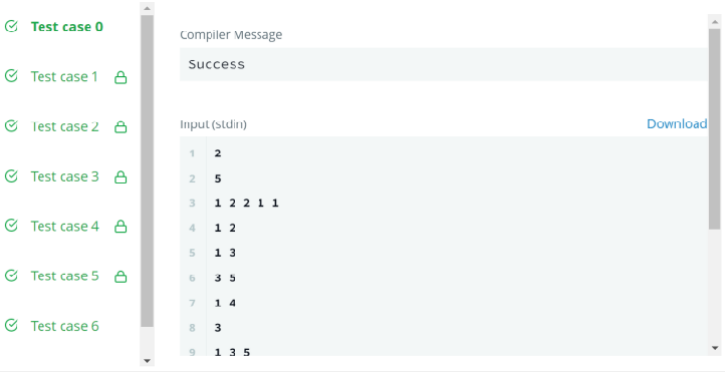
```
        for (int i = 0; i < nTest; ++i) {
            solve();
        }
    }
}
```

Result/Output/Writing Summary:

a. **Tree Huffman Decoding:**



b. **Balanced Forest:**



Learning outcomes (What I have learnt):

a. Learnt about maps.
b. Got an overview of the maps and hashing.
c. Get to know about crucial test cases.
d. Got an understanding about referencing of maps.
e. Learn about trees.