كليـــة الهنـــدســـة
College of Engineering

FINAL PROJECT

CS2011

OBJECTIVE

Applying the lessons we learned in the duration of this course

INSTRUCTOR: Fidaa Abed

Data Structures & Algorithms

Student(s) Name (s)	**Judy Abuquta- Sarah Eid- Abeer Hussain**
Student ID #	**S22107883- S22107757- S22107714**
Date of submission	**8/5/2024**

Question 1: Super Maximum Cost Queries

Victoria has a tree, T, consisting of N nodes numbered from 1 to N. Each edge from node U_i to V_i in tree T has an integer weight, W_i.

Let's define the cost, C, of a path from some node X to some other node Y as the maximum weight (W) for any edge in the unique path from node X to node Y.

Victoria wants your help processing Q queries on tree T, where each query contains 2 integers, L and R, such that $L \leq R$. For each query, she wants to print the number of different paths in T that have a cost, C, in the inclusive range $[L, R]$.

It should be noted that path from some node X to some other node Y is considered same as path from node Y to X i.e $\{X, Y\}$ is same as $\{Y, X\}$.

CODE:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.Scanner;

public class Solution {

	static class Edge {
		int v1, v2, w;

		Edge(int v1, int v2, int w) {
			this.v1 = v1;
			this.v2 = v2;
			this.w = w;
		}

		@Override
		public String toString() {
			return v1 + " - " + v2 + " : " + w;
		}
	}

	static int find(int[] subsets, int i) {
		if (subsets[i] != i) {
			subsets[i] = find(subsets, subsets[i]);
		}

		return subsets[i];
	}


	public static void main(String[] args) {

		Scanner sc = new Scanner(System.in);

		int n = sc.nextInt();
		int q = sc.nextInt();

		Edge[] arr = new Edge[n - 1];

		// get edge and sort length
		for (int i = 0; i < arr.length; i++) {
```

```
                arr[i] = new Edge(sc.nextInt() - 1, sc.nextInt() - 1, sc.nextInt());
        }

        Arrays.sort(arr, new Comparator<Edge>() {

                @Override
                public int compare(Edge o1, Edge o2) {
                        if (o1.w > o2.w)
                                return 1;
                        else if (o1.w < o2.w)
                                return -1;
                        return 0;
                }

        });

        // get size per length
        int[] u = new int[n];
        long[] cu = new long[n];
        for (int i = 0; i < u.length; i++) {
                u[i] = i;
                cu[i] = 1;
        }

        ArrayList<Integer> len = new ArrayList<>();
        ArrayList<Long> si = new ArrayList<>();

        len.add(0);
        si.add(0L);
        for (int i = 0; i < arr.length; i++) {
                Edge cur = arr[i];

                int p1 = find(u, cur.v1);
                int p2 = find(u, cur.v2);

                u[p1] = p2;

                long tot = si.get(si.size() - 1);

                if(len.get(len.size() - 1) != cur.w) {
                        len.add(cur.w);
                        si.add(tot + cu[p1] * cu[p2]);
                } else  {
                        si.set(si.size() - 1, si.get(si.size() - 1) + cu[p1] * cu[p2]);
                }


                cu[p2] += cu[p1];

        }

        StringBuilder sb = new StringBuilder();
        // query
        for (int i = 0; i < q; i++) {
                int L = sc.nextInt();
                int R = sc.nextInt();

                if(L > len.get(len.size() - 1) || R < len.get(1))
                        sb.append("0\n");
                 else {
                         int i1 = Collections.binarySearch(len, L - 1);
                         int i2 = Collections.binarySearch(len, R);
```

```
                                        i1 = i1 < 0 ? -i1 - 2 : i1;
                                        i2 = i2 < 0 ? -i2 - 2 : i2;
                                        sb.append(si.get(i2) - si.get(i1)).append("\n");
                                }
                        }

                        sc.close();
                        System.out.println(sb.toString());

        }

}
```

Question 2: Library Query

A giant library has just been inaugurated this week. It can be modeled as a sequence of N consecutive shelves with each shelf having some number of books. Now, being the geek that you are, you thought of the following two queries which can be performed on these shelves.

- Change the number of books in one of the shelves.
- Obtain the number of books on the shelf having the k^{th} rank within the range of shelves.

A shelf is said to have the k^{th} rank if its position is k when the shelves are sorted based on the number of the books they contain, in ascending order.

The number of books on each shelf is always guaranteed to be between `1` and `1000`.

CODE:

```
import java.util.*;
public class Solution {

        public static void main(String[] args) {
                Scanner s=new Scanner(System.in);
                int t=s.nextInt();
                for(int j=0;j<t;j++){
                        int n=s.nextInt();
                        int a[]=new int[n];
                        for(int i=0;i<a.length;i++)
                                a[i]=s.nextInt();
                        int q=s.nextInt();
                        int x=0,y=0,k=0;

                        for(int i=0;i<q;i++){
                                int tmp=s.nextInt();
                                 x=s.nextInt();
                                if(tmp==1){
                                        k=s.nextInt();
                                        a[x-1]=k;
                                }else{
                                        int ar[]=new int[1000];
                                        y=s.nextInt();
                                        k=s.nextInt();
                                        for(int l=x;l<=y;l++){
                                                ar[a[l-1]-1]++;
                                        }
                                        for(int l=0;l<ar.length;l++){
                                                k=k-ar[l];
```

```
                                        if(k<=0){
                                            System.out.println(l+1);
                                            break;
                                        }
                                    }
                                    ar=null;
                                }
                            }
                        }
                    }
}
```

Question 3: Rooted Tree

You are given a rooted tree with N nodes and the root of the tree, R, is also given. Each node of the tree contains a value, that is initially empty. You have to mantain the tree under two operations:

1. Update Operation
2. Report Operation

Update Operation

Each Update Operation begins with the character `U`. Character `U` is followed by 3 integers T, V and K. For every node which is the descendent of the node T, update it's value by adding V + d*K, where V and K are the parameters of the query and d is the distance of the node from T. Note that V is added to node T.

Report Operation

Each Report Operation begins with the character `Q`. Character `Q` is followed by 2 integers, A and B. Output the sum of values of nodes in the path from A to B modulo (10^9 + 7)

CODE:

```
import java.io.*;
import java.util.*;

public class Solution {

  static final int D = 17;
  static final int MOD = 1_000_000_007;
  static List<Integer>[] e;
  static int[][] par;
  static int[][] fenwick;
  static int[] dep;
  static int[] dfnl;
  static int[] dfnr;
  static int tick = 0;


  static int lowestCommonAncestor(int u, int v) {
    if (dep[u] < dep[v]) {
      int temp = u;
      u = v;
      v = temp;
    }
    for (int i = D; --i >= 0; ) {
      if (dep[u]-(1<<i) >= dep[v]) {
        u = par[i][u];
      }
    }
    if (u == v) {
      return u;
    }
```

```
  for (int i = D; --i >= 0; ) {
   if (par[i][u] != par[i][v]) {
    u = par[i][u];
    v = par[i][v];
   }
  }
  return par[0][u];
 }

 static class Node {
  int u;
  int p;
  boolean start = true;
  Node(int u, int p) {
   this.u = u;
   this.p = p;
  }
 }

 static void dfs(int u, int p) {
  Deque<Node> queue = new LinkedList<>();
  queue.add(new Node(u, p));
  while (!queue.isEmpty()) {
   Node node = queue.peek();
   if (node.start) {
    dfnl[node.u] = tick++;
    for (int v: e[node.u]) {
     if (v != node.p) {
      par[0][v] = node.u;
      dep[v] = dep[node.u]+1;
      queue.addFirst(new Node(v, node.u));
     }
    }
    node.start = false;
   } else {
    dfnr[node.u] = tick;
    queue.remove();
   }
  }
 }

 static void add(int fenwick[], int x, int v) {
  for (; x < fenwick.length; x |= x+1) {
   fenwick[x] = (fenwick[x] + v) % MOD;
  }
 }

 static int getSum(int fenwick[], int x) {
  int s = 0;
  for (; x > 0; x &= x-1)
   s = (s + fenwick[x-1]) % MOD;
  return s;
 }

 static int get(int u) {
  long pw = 1;
  long s = 0;
  for (int i = 0; i < 3; i++) {
   s = (s + pw * getSum(fenwick[i], dfnl[u]+1)) % MOD;
   pw = (pw * dep[u]) % MOD;
  }
  return (int) (((MOD+1l) / 2 * s)%MOD);
 }


 static int query(int u, int v) {
  int w = lowestCommonAncestor(u, v);
  long s = ((long)(get(u))+get(v)-get(w))%MOD;
  if (par[0][w] >= 0) {
   s = (s - get(par[0][w])) % MOD;
```

```
  }
  return (int) s;
 }

 static void upd(int fenwick[], int l, int r, int v) {
  add(fenwick, l, v);
  add(fenwick, r, -v);
 }

 static void update(int u, int x, int y) {
  int l = dfnl[u];
  int r = dfnr[u];
  upd(fenwick[2], l, r, y);
  upd(fenwick[1], l, r, (int)(((long)(1 - 2 * dep[u]) * y + 2l * x) % MOD));
  upd(fenwick[0], l, r, (int)((dep[u] * (dep[u] - 1l) * y + 2 * (1l - dep[u]) * x) % MOD));
 }

 public static void main(String[] args) throws IOException {
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
  BufferedWriter bw = new BufferedWriter(new FileWriter(System.getenv("OUTPUT_PATH")));

  StringTokenizer st = new StringTokenizer(br.readLine());
  int n = Integer.parseInt(st.nextToken());
  int m = Integer.parseInt(st.nextToken());
  int rt = Integer.parseInt(st.nextToken())-1;


  e = new List[n];
  for (int i = 0; i < n; i++) {
   e[i] = new LinkedList<>();
  }
  for (int i = 0; i < n-1; i++) {
   st = new StringTokenizer(br.readLine());
   int u = Integer.parseInt(st.nextToken())-1;
   int v = Integer.parseInt(st.nextToken())-1;
   e[u].add(v);
   e[v].add(u);
  }

  dep = new int[n];
  par = new int[D][n];
  dfnl = new int[n];
  dfnr = new int[n];

  tick = 0;
  dep[rt] = 0;
  par[0][rt] = -1;
  dfs(rt, -1);

  for (int k = 1; k < D; k++) {
   for (int i = 0; i < n; i++) {
    par[k][i] = par[k-1][i] == -1 ? par[k-1][i] : par[k-1][par[k-1][i]];
   }
  }

  fenwick = new int[3][n];

  while (m-- > 0) {
   st = new StringTokenizer(br.readLine());
   char op = st.nextToken().charAt(0);
   int u = Integer.parseInt(st.nextToken()) - 1;
   int v = Integer.parseInt(st.nextToken());
   if (op == 'Q') {
    v--;
    int result = (query(u, v)+MOD)%MOD;
    bw.write(result + "\n");
   } else {
    int w = Integer.parseInt(st.nextToken());
    update(u, v, w);
   }
```

```
    }

    bw.newLine();
    bw.close();
    br.close();
  }
}
```