Data Structures Final Project

Abeer Hussain, Judy Abuquta, Sarah Eid

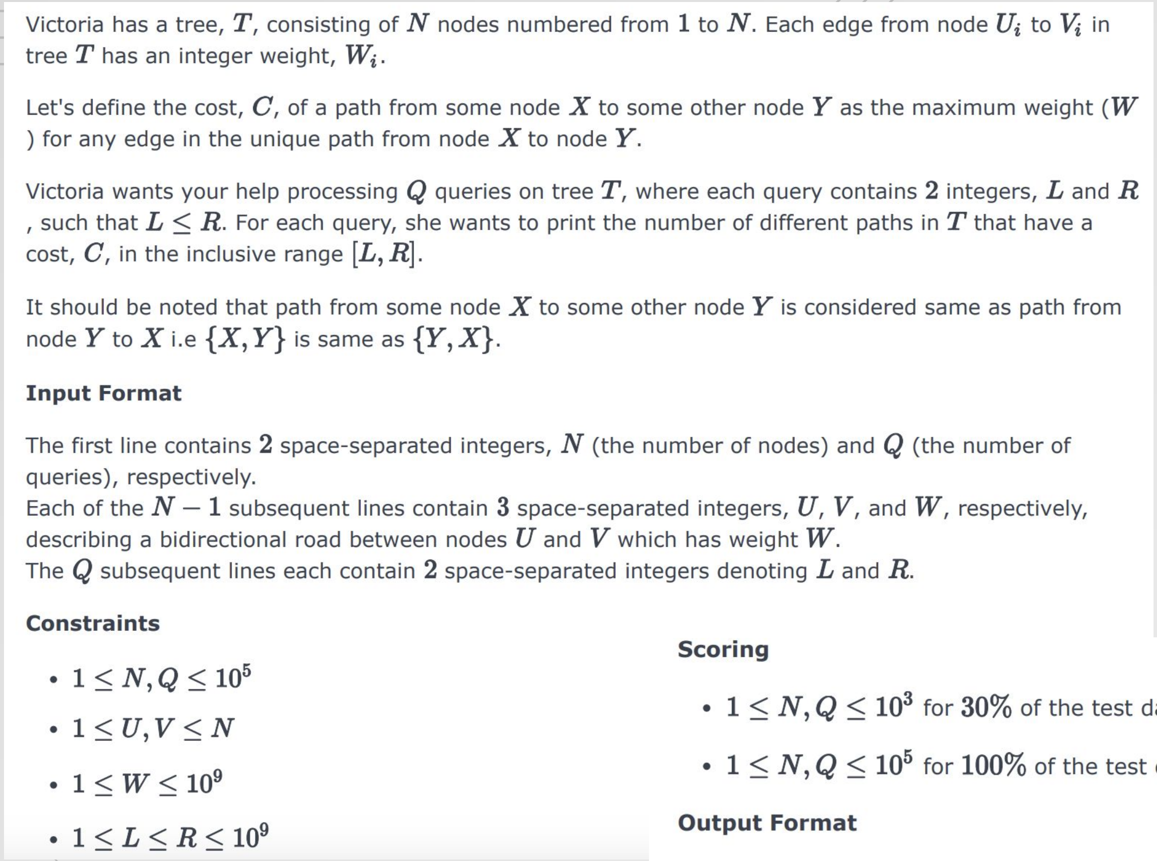
Table of contents

01
Super Maximum Cost Queries

02
Library Query

03
Rooted Tree

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 $[L, R]$.

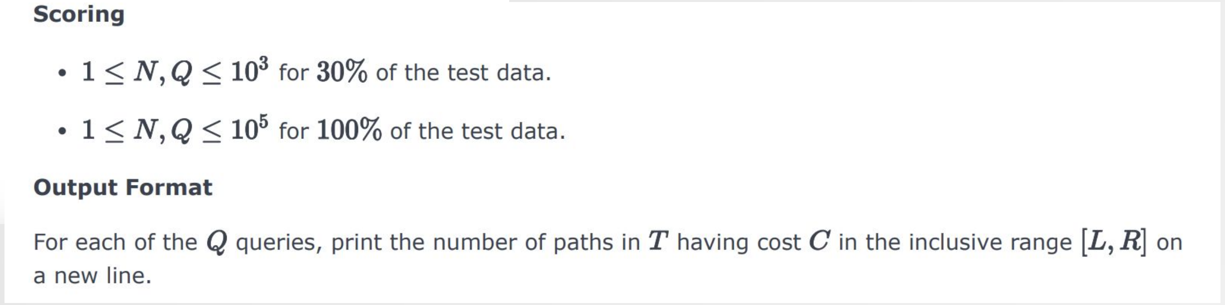
It should be noted that path from some node X to some other node Y is considered same as path from node Y to X i.e $\{X, Y\}$ is same as $\{Y, X\}$.

Input Format

The first line contains 2 space-separated integers, N (the number of nodes) and Q (the number of queries), respectively.
Each of the $N - 1$ subsequent lines contain 3 space-separated integers, U, V, and W, respectively, describing a bidirectional road between nodes U and V which has weight W.
The Q subsequent lines each contain 2 space-separated integers denoting L and R.

Constraints

- $1 \leq N, Q \leq 10^5$
- $1 \leq U, V \leq N$
- $1 \leq W \leq 10^9$
- $1 \leq L \leq R \leq 10^9$

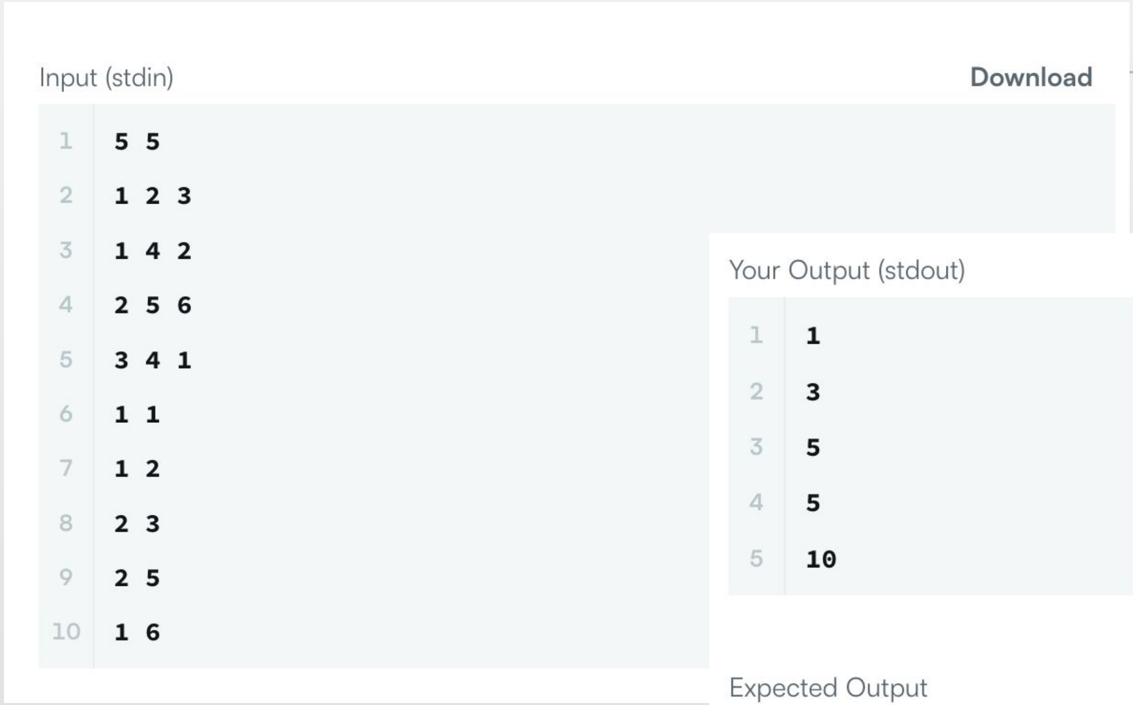


Scoring

- $1 \leq N, Q \leq 10^3$ for 30% of the test data.
- $1 \leq N, Q \leq 10^5$ for 100% of the test data.

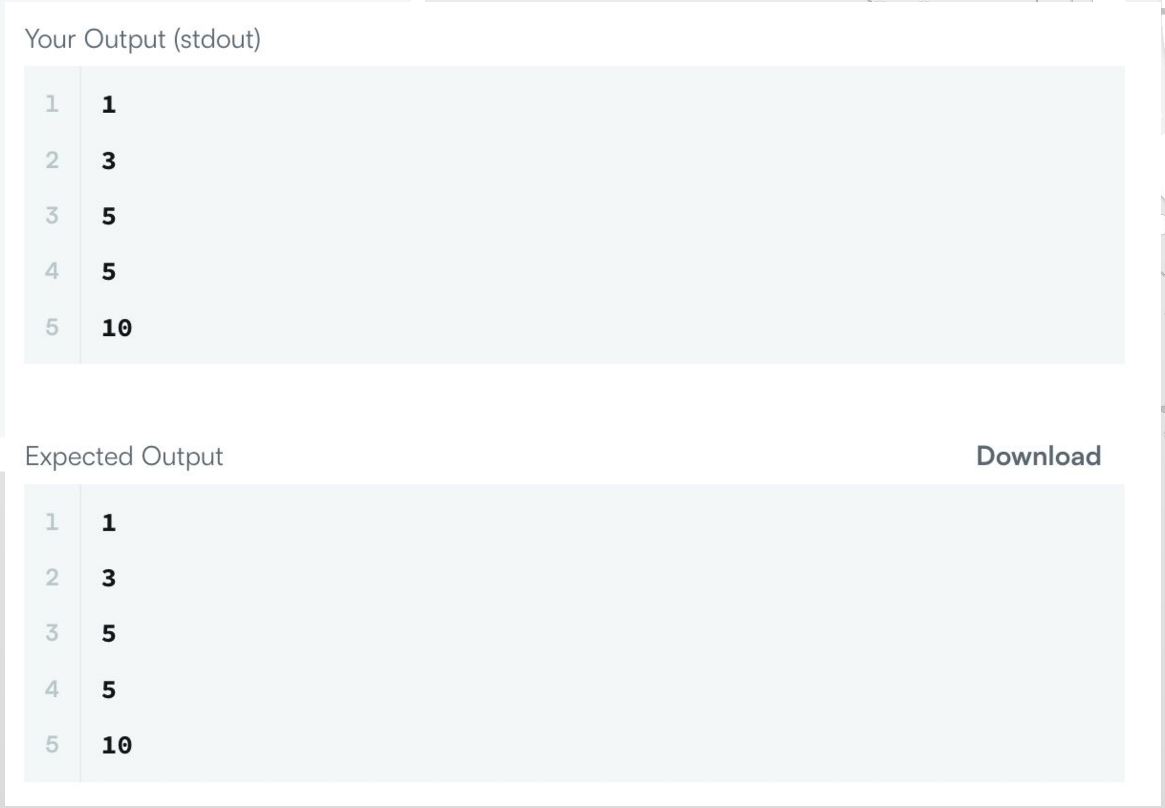
Output Format

For each of the Q queries, print the number of paths in T having cost C in the inclusive range $[L, R]$ on a new line.



Input (stdin) Download

```
5 5
1 2 3
1 4 2
2 5 6
3 4 1
1 1
1 2
2 3
2 5
1 6
```



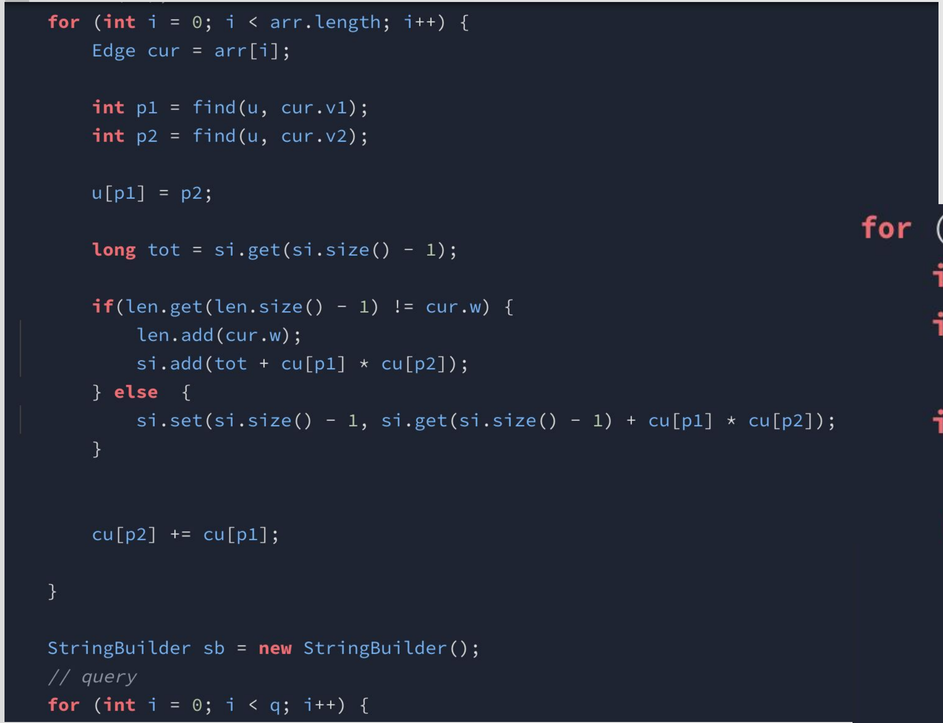
Your Output (stdout)

```
1
3
5
5
10
```

Expected Output Download

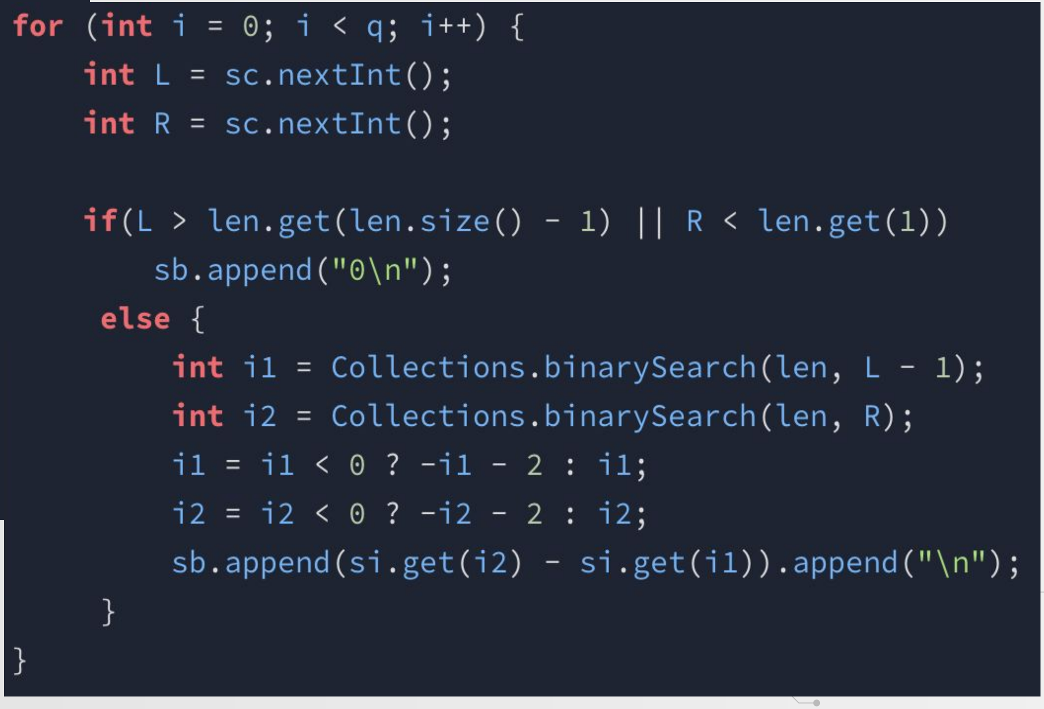
```
1
3
5
5
10
```

CODE



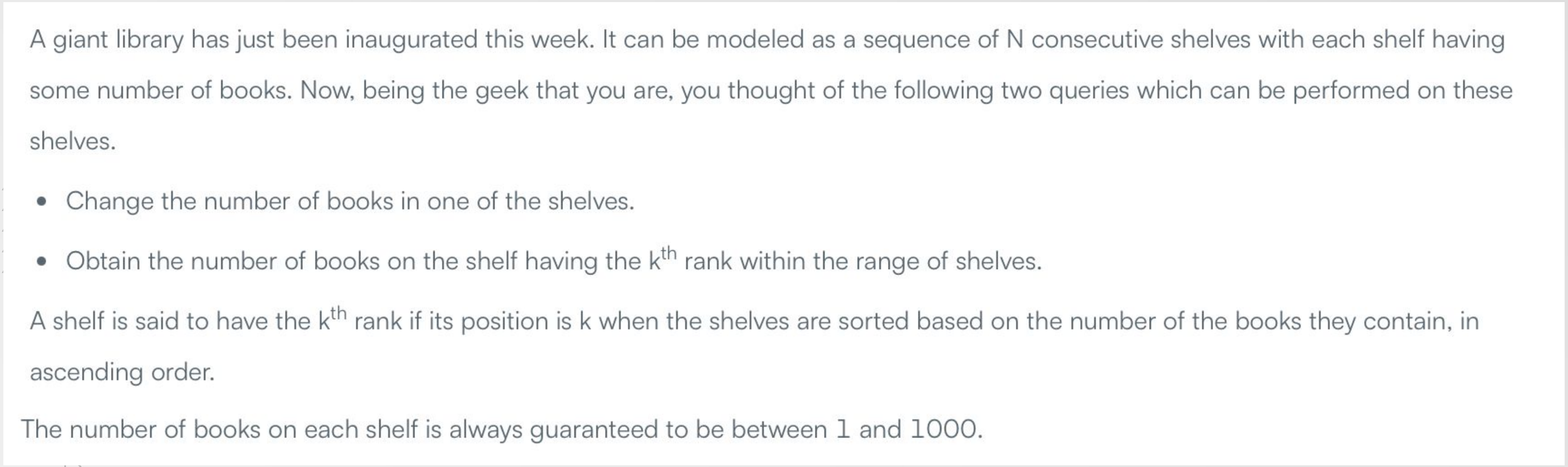
```java
for (int i = 0; i < arr.length; i++) {
    Edge cur = arr[i];

    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++) {
```

Query



```java
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");
     }
}
```

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.

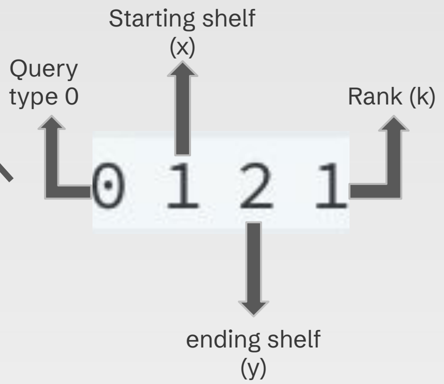
The number of books on each shelf is always guaranteed to be between `1` and `1000`.

Input

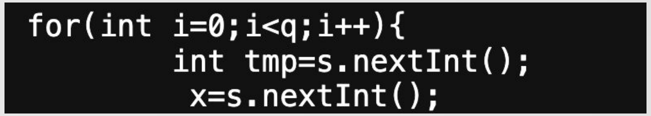
```
2
2
1 2
2
0 1 2 1
0 1 2 2
4
4 3 2 1
4
0 1 1 1
1 1 1
0 1 1 1
0 1 4 3
```

Output

```
1
2
4
1
2
```

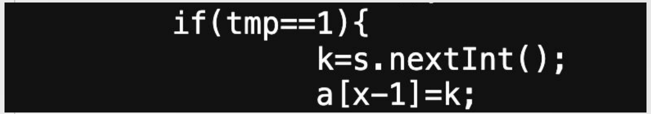


Code



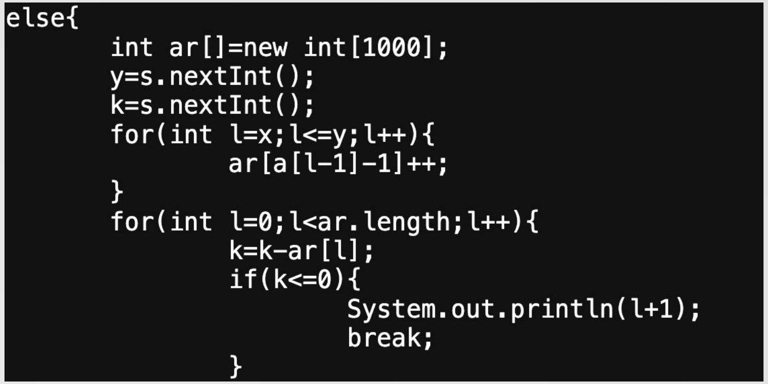
```
for(int i=0;i<q;i++){
        int tmp=s.nextInt();
         x=s.nextInt();
```

Query type 1



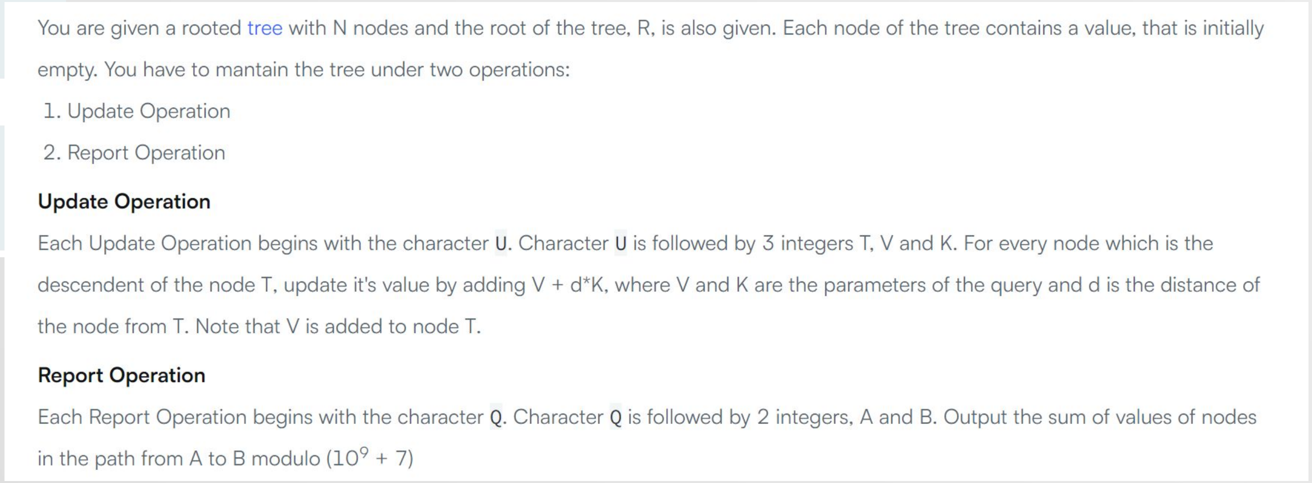
```
        if(tmp==1){
                k=s.nextInt();
                a[x-1]=k;
```

Query type 0



```
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;
                }
```

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:

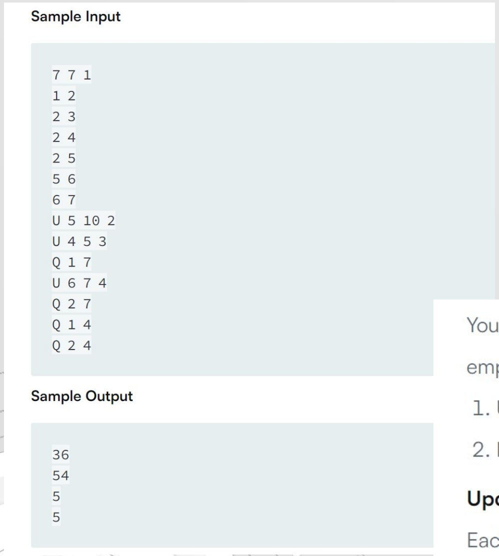
1. Update Operation
2. Report Operation

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)



Sample Input

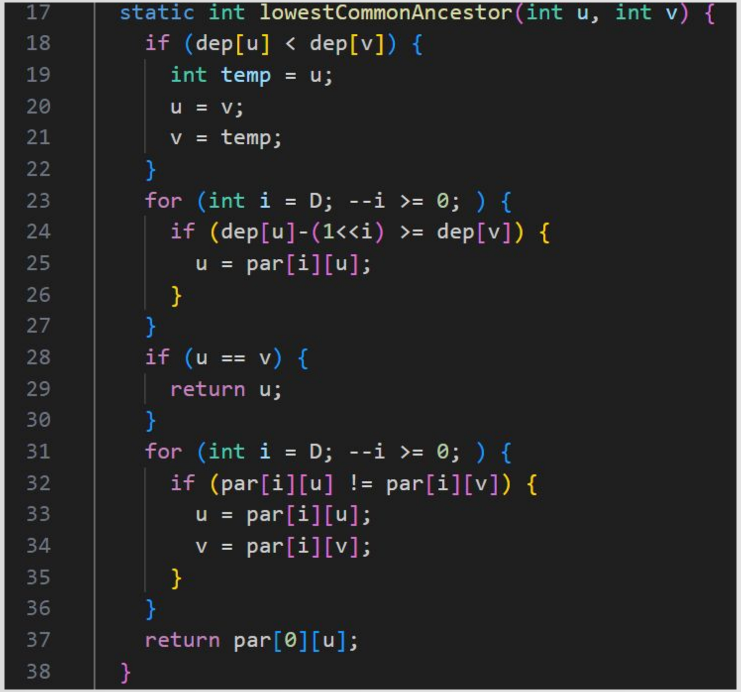
```
7 7 1
1 2
2 3
2 4
2 5
5 6
6 7
U 5 10 2
U 4 5 3
Q 1 7
U 6 7 4
Q 2 7
Q 1 4
Q 2 4
```

Sample Output

```
36
54
5
5
```

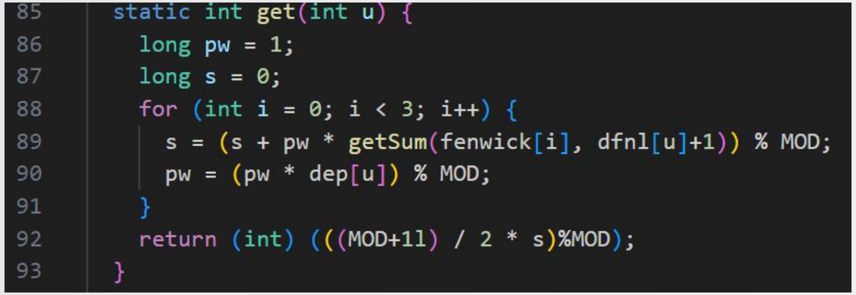
Code

Lowest Common Ancestor (LCA)



```java
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];
}
```

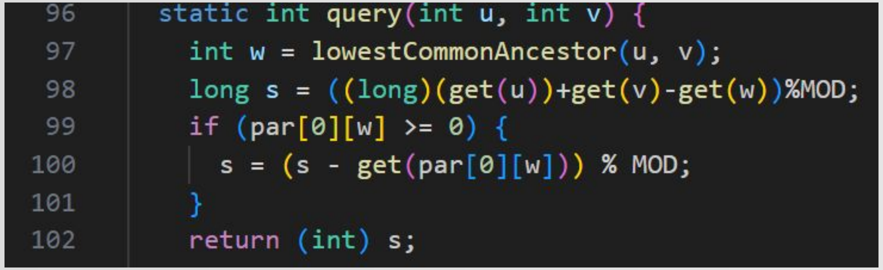
Get Function



```java
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);
}
```

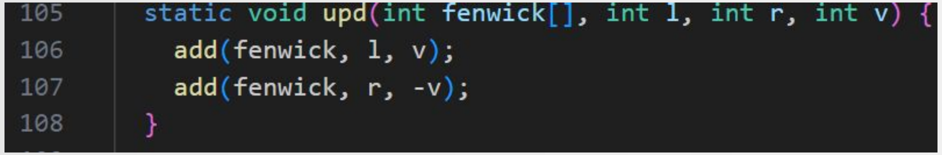
Code

Query function



```
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;
```

UPD Function



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

Thank you for listening