Matrix

by HackerRank

Problem Submissions Leaderboard Discussions Editorial

The kingdom of Zions has n cities and $n-1$ bidirectional roads. There is a unique path between any pair of cities.

Morpheus has found out that k machines are planning to destroy the whole kingdom. These machines are initially living in k different cities of the kingdom and they can launch an attack anytime. Neo has to destroy some of the roads in the kingdom to disrupt all connections among the machines. After destroying the necessary roads there should be no path between any two machines.

Since the attack may happen at any moment, Neo has to do this task as fast as possible. Each road in the kingdom takes a certain amount of time to destroy and only one road can be destroyed at a time.

You need to write a program that tells Neo the minimum amount of time he will require to destroy the necessary roads.

Input Format

The first line of the input contains two space-separated integers, n and k. Cities are numbered 0 to $n-1$.

$n-1$ lines follow, each containing three space-separated integers, x y z, which means that there is a bidirectional road connecting city x and city y, and to destroy this road it takes z units of time.

k lines follow, each containing an integer. The i^th integer is the id of the city in which the i^th machine is currently located.

Constraints

- $2 \leq n \leq 10^5$
- $2 \leq k \leq n$
- $1 \leq time\ to\ destroy\ a\ road \leq 10^6$

Output Format

Print in a single line the minimum time required to disrupt the connection among machines.

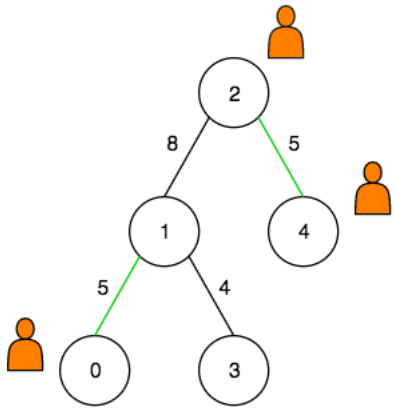
Sample Input

```
5 3
2 1 8
1 0 5
2 4 5
1 3 4
2
4
0
```

Sample Output

```
10
```

Explanation



The machines are located at the cities 0, 2 and 4. Neo can destroy the road connecting city 2 and city 4 of weight 5 , and the road connecting city 0 and city 1 of weight 5. As only one road can be destroyed at a time, the total minimum time taken is 10 units of time. After destroying these roads none of the machines can reach another machine via any path.

Submissions: 1600
Max Score: 70
Difficulty: Hard

Rate This Challenge:

More

Current Buffer (saved locally, editable) Java 8

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

class DisjointSet{

    long[] rank,parent;
    int n;
    boolean[] isMachine;

    public DisjointSet(int n){
        this.n = n;
        rank = new long[n];
        parent = new long[n];
        isMachine = new boolean[n];
        makeset(n);
    }

    void makeset(int n){
        for(int i = 0 ; i < n ; i++){
            parent[i] = (long) i;
        }
    }

    long find(int x){

        if(parent[x] != (long)x){
            parent[x] = find((int)parent[x]);
        }
       return parent[x];
    }

    void union(int x, int y){

        int xRoot = (int) find(x);
        int yRoot = (int)  find(y);

        if(xRoot == yRoot){
```

```
            return;
        }

        if(rank[xRoot] < rank[yRoot]){
            parent[xRoot] = yRoot;
        }
        else if(rank[xRoot] > rank[yRoot]){
            parent[yRoot] = xRoot;
        }
        else{
            parent[yRoot] = xRoot;
            rank[xRoot] = rank[xRoot] + 1;
        }
    }

    void setMachine(int x){
        isMachine[(int) find(x)] = true;
    }

    boolean getMachine(int x){
        return isMachine[(int)find(x)];
    }

}


class Vertex{

    private int v1,v2;

    public Vertex(int v1, int v2){
        this.v1 = v1;
        this.v2 = v2;
    }

    public int getV1(){
        return v1;
    }

    public int getV2(){
        return v2;
    }

}



public class Solution {

    public static void main(String[] args) throws IOException{

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String line = br.readLine();
        String[] numbers = line.split("\\s");

        int V = Integer.parseInt(numbers[0]);
        int E = Integer.parseInt(numbers[1]);

        DisjointSet disjoint = new DisjointSet(V);

        HashMap<Integer, ArrayList<Vertex>> adj = new HashMap<Integer, ArrayList<Vertex>>();;

        long output = 0;

        for(int i = 0 ; i < V - 1  ; i++){

            line = br.readLine();
            numbers = line.split("\\s");

            int v1 = Integer.parseInt(numbers[0]);
            int v2 = Integer.parseInt(numbers[1]);
            int weight = Integer.parseInt(numbers[2]);
            output+= (long) weight;
```

```
            Vertex vertex = new Vertex((v1),(v2));

            if(adj.containsKey(weight)){
                ArrayList<Vertex> clone = (ArrayList)adj.get(weight).clone();
                clone.add(vertex);
                adj.put(weight,clone);
            }
            else{
                ArrayList<Vertex> clone = new ArrayList<Vertex>();
                clone.add(vertex);
                adj.put(weight,clone);
            }
        }

        List sortedKeys=new ArrayList(adj.keySet());
        Collections.sort(sortedKeys);

        for(int i = 0 ; i < E ; i++){
            disjoint.setMachine(Integer.parseInt(br.readLine()));
        }

         for(int g = sortedKeys.size() - 1 ; g >= 0 ; g--){

            ArrayList<Vertex> clone = (ArrayList) adj.get(sortedKeys.get(g)).clone();

             for(int a = 0 ; a < clone.size() ; a++){

                 Vertex vert = clone.get(a);
                 int v1 = vert.getV1();
                 int v2 = vert.getV2();

                 boolean representativeV1 = disjoint.getMachine(v1);
                 boolean representativeV2 = disjoint.getMachine(v2);

                 if(representativeV1 && !representativeV2){
                     output = output - (Integer) sortedKeys.get(g);
                     disjoint.setMachine(v2);
                 }
                 else if(representativeV2 && !representativeV1){
                     output = output - (Integer) sortedKeys.get(g);
                     disjoint.setMachine(v1);
                 }
                 else if(!representativeV2 && !representativeV1){
                     output = output - (Integer) sortedKeys.get(g);
                 }

                 disjoint.union(v1,v2);

             }
        }
        System.out.println(output);
    }
}
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code