Practice Compete Jobs Rank Leaderboard gadhiya

Dashboard > Algorithms > Graph Theory > Jack goes to Rapture

Badge Progress ★★★★

Points: 4727.88 Rank: 491

Jack goes to Rapture

by amititkgp

Problem Submissions Leaderboard Discussions Editorial

Jack has just moved to a new city called Rapture. However, he is confused by Rapture's public transport system. The rules of the public transport are as follows:

1. Every pair of connected stations has a fare assigned to it.
2. If a passenger travels from station A to station B, he only has to pay the difference between the fare from A to B and the cumulative fare that he has paid to reach station A [*fare(A,B) - total fare to reach station A*]. If the difference is negative, he can travel free of cost from A to B.

Since Jack is new to the city, he is unemployed and low on cash. He needs your help to figure out the most cost efficient way to go from the first station to the last station. You are given the number of stations N (numbered from 1 to N), and the fare between the *E* pair of stations that are connected.

Input Format

The first line contains two integers, N and E, followed by E lines containing three integers each: the two stations that are connected to each other and the fare between them (C).

Constraints

- $1 \leq N \leq 50000$
- $1 \leq E \leq 500000$
- $1 \leq C \leq 10^7$

Output Format

The minimum fare to be paid to reach station N from station 1. If the station N cannot be reached from station 1, print "NO PATH EXISTS" (without quotes).

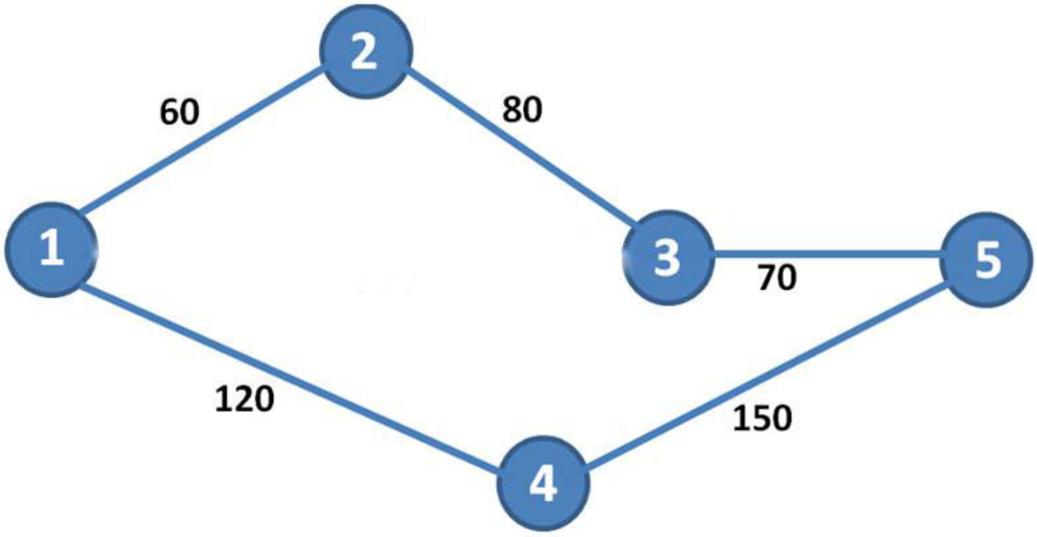
Sample Input 0

```
5 5
1 2 60
3 5 70
1 4 120
4 5 150
2 3 80
```

Sample Output 0

```
80
```

Explanation 0



There are two ways to go from first station to last station.

- 1 -> 2 -> 3-> 5
- 1 -> 4 -> 5

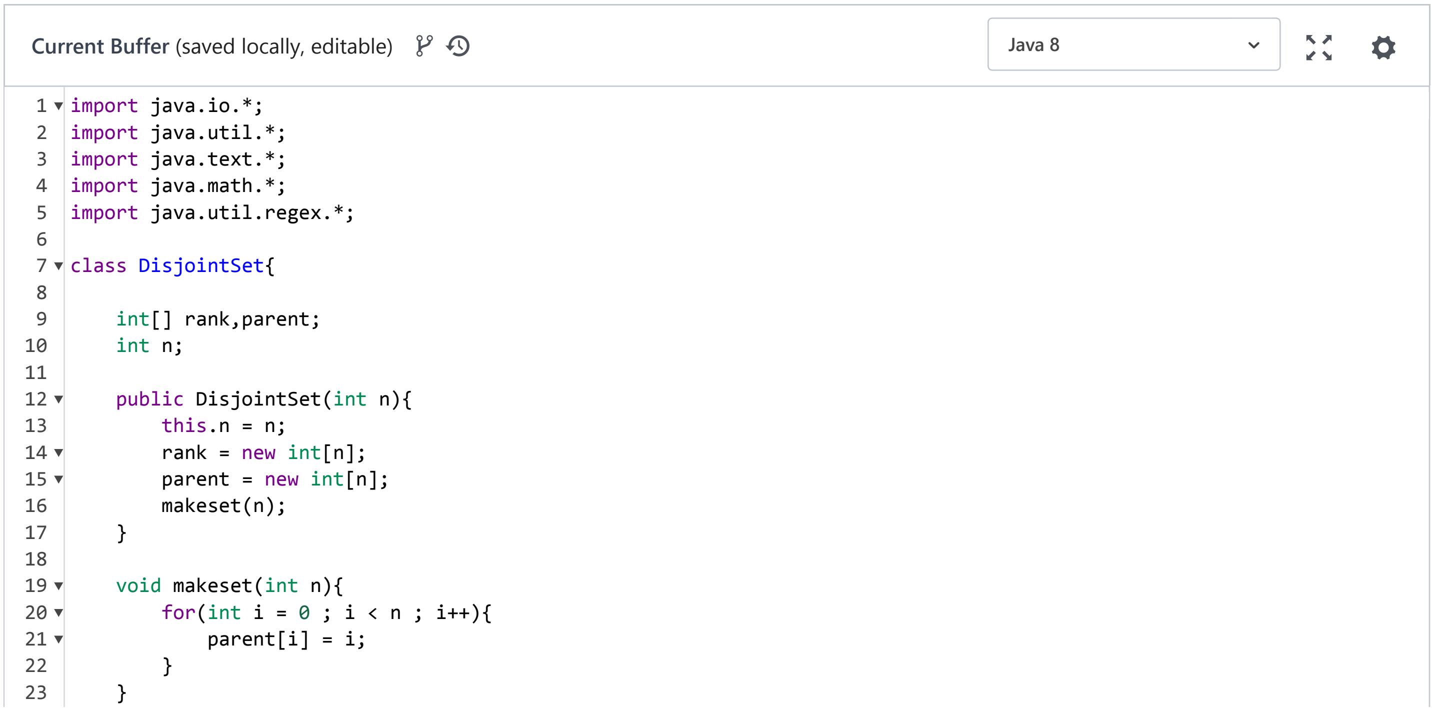
For the first path, Jack first pays 60 units of fare to go from station 1 to 2. Next, Jack has to pay 80-60 = 20 units to go from 2 to 3. Now, to go from 3 to 5, Jack has to pay 70-(60+20) = -10 units, but since this is a negative value, Jack pays 0 units to go from 3 to 5. Thus the total cost of this path is (60+20) = 80 units.

For the second path, Jack pays 120 units to reach station 4 from station 1. To go from station 4 to 5, Jack has to pay 150-120 = 30 units. Thus the total cost becomes (120+30) = 150 units. So, the first path is the most cost efficient, with a cost of 80.

Submissions: 2229
Max Score: 80
Difficulty: Medium

Rate This Challenge:
☆☆☆☆☆

More



Java 8

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

class DisjointSet{

    int[] rank,parent;
    int n;

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

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

```java

    int find(int x){

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

    void union(int x, int y){

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

class Graph{

    private int V;
    private HashMap<Integer, LinkedList<Vertex>> adj;
    private HashMap<Integer,LinkedList<Edges>> edges;

    public Graph(int V){
        this.V = V;
         adj = new HashMap<Integer, LinkedList<Vertex>>();
         edges = new HashMap<Integer, LinkedList<Edges>>();
    }

    public void addEdge(int weight, Vertex ver){

        if(adj.containsKey(weight)){
            LinkedList<Vertex> link = adj.get(weight);
            link.add(ver);
            adj.put(weight,link);
        }
        else{
            LinkedList<Vertex> link = new LinkedList<Vertex>();
            link.add(ver);
            adj.put(weight,link);
        }

    }

    public void primsAlgo(){

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

         DisjointSet disjoint = new DisjointSet(V);

         for(int g = 0 ; g < sortedKeys.size() ; g++){

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

            Iterator itr = clone.listIterator();

            while(itr.hasNext()){

                Vertex vert = (Vertex) itr.next();
                int v1 = vert.getV1();
                int v2 = vert.getV2();

                int representativeV1 = disjoint.find(v1);
                int representativeV2 = disjoint.find(v2);

                if(representativeV1 != representativeV2){
                    disjoint.union(representativeV1, representativeV2);

                    if(edges.containsKey(v2)){
                        LinkedList<Edges> edge = edges.get(v2);
                        Edges newEdge = new Edges(v1,(int)sortedKeys.get(g));
                        edge.add(newEdge);
                        edges.put(v2,edge);
                    }
                    else{
                        LinkedList<Edges> edge = new LinkedList<Edges>();
                        Edges newEdge = new Edges(v1,(int)sortedKeys.get(g));
                        edge.add(newEdge);
                        edges.put(v2,edge);
                    }

                    if(edges.containsKey(v1)){
                        LinkedList<Edges> edge = edges.get(v1);
                        Edges newEdge = new Edges(v2,(int)sortedKeys.get(g));
                        edge.add(newEdge);
                        edges.put(v1,edge);
                    }
                    else{
                        LinkedList<Edges> edge = new LinkedList<Edges>();
                        Edges newEdge = new Edges(v2,(int)sortedKeys.get(g));
                        edge.add(newEdge);
                        edges.put(v1,edge);
                    }
                }
            }
        }
    }

    public long doBFS(){

        boolean[] isVisited = new boolean[V];

        long[] dist = new long[V];

        dist[0] = 0;

        for(int i = 1 ; i < V ; i++){
            dist[i] = Integer.MAX_VALUE;
        }

        LinkedList<Integer> queue = new LinkedList<Integer>();
        queue.add(0);

        while(queue.size() != 0){

            int s = queue.poll();
            isVisited[s] = true;

            Iterator<Edges> itr = edges.get(s).listIterator();

            while(itr.hasNext()){

                Edges edge = itr.next();

                    if(!isVisited[edge.getV()]){

                        if((long) edge.getEdge() > dist[s]){

                            long currentVal = (long) edge.getEdge();

                            if(currentVal < dist[edge.getV()]){
                                dist[edge.getV()] = currentVal;
                            }
                        }
                        else{
                            if(dist[edge.getV()] > dist[s]){
                                dist[edge.getV()] = dist[s];
                            }
                        }

                        queue.add(edge.getV());

                    }
            }
        }

        return dist[V - 1];
    }
}

class Edges{

    private int v,edge;

    public Edges(int v, int edge){
        this.v = v;
        this.edge = edge;
    }

    public int getV(){
        return v;
    }

    public int getEdge(){
        return edge;
    }

}

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 n = Integer.parseInt(numbers[0]);
            int m = Integer.parseInt(numbers[1]);

            Graph graph = new Graph(n);

            HashMap<String,Integer> myMap = new HashMap<String,Integer>();

            for(int a1 = 0; a1 < m; a1++){

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

                int x = Integer.parseInt(numbers[0]);
                int y = Integer.parseInt(numbers[1]);
                int r = Integer.parseInt(numbers[2]);

                Vertex vertex = new Vertex(x - 1,y - 1);
                graph.addEdge(r, vertex);
                vertex = new Vertex(y - 1,x-1);
                graph.addEdge(r, vertex);
            }

        graph.primsAlgo();

        long output = graph.doBFS();

        if(output == Integer.MAX_VALUE){
            System.out.println("NO PATH EXISTS");
        }
        else{
            System.out.println(output);
        }


    }
}

```

Line: 1 Col: 1

Upload Code as File ☐ Test against custom input

Run Code Submit Code

Join us on IRC at #hackerrank on freenode for hugs or bugs.

Contest Calendar | Interview Prep | Blog | Scoring | Environment | FAQ | About Us | Support | Careers | Terms Of Service | Privacy Policy | Request a Feature