https://course.acciojob.com/idle?question=5d8035f8-c8b0-4ceb-9057-90204aafb3f0

- MEDIUM
- Max Score: 40 Points
-
-

Easy Dijkstra Problem

Determine the shortest path between the specified vertices in the directed graph given in the input data.

Input Format

First line consist of the numbers V, K (number of vertices, number of edges).

Then K lines follow, each containing the following numbers separated by a single space:

ai, bi, ci

It means that the graph being described contains an edge from ai to bi, with a weight of ci.

Below the graph description a line containing a pair of integers A, B is present.

The goal is to find the shortest path from vertex A to vertex B.

Output Format

Your program should output (in a separate line) a single number C - the length of the shortest path from vertex A to vertex B. In case there is no such path, your program should output a single word "NO" (without quotes)

Example 1

Input

```
3 2
1 2 5
2 3 7
1 3
```

Output

```
12
```

Example 2

Input

```
3 3
1 2 4
1 3 7
2 3 1
1 3
```

Output

```
5
```

Example 3

Input

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

Output

```
NO
```

Constraints:

All numbers in the input data are integers in the range 0 to 10000.

Topic Tags

- Graphs
- BFS
- DFS

My code

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

class Node{
    int v, wt;
    Node(int v, int wt){
        this.v = v;
        this.wt = wt;
    }
}
class nodeComparator implements Comparator<Node>{
    public int compare(Node node1, Node node2){
        return node1.wt - node2.wt;
    }
}
class graph{
    ArrayList<ArrayList<Node>> g = new
ArrayList<ArrayList<Node>>();
    graph(int n){
        for(int i=0; i<n; i++){
            g.add(new ArrayList<Node>());
        }
    }
    void addNode(int u, int v, int wt){
        g.get(u).add(new Node(v, wt));
    }

     void findPath(int start, int end){
        // Write your code here

                int n = g.size();

                PriorityQueue<Node> pq = new PriorityQueue<>(new
nodeComparator());

                pq.add(new Node(start, 0));

                boolean spt[] = new boolean[n];

                while(pq.size() > 0) {
                        Node curr = pq.poll();
                        int u = curr.v;
                        int dist_u = curr.wt;


                        if(spt[u] == true) continue;


                        spt[u] = true;
            if(u == end) {
                System.out.println(dist_u);
                return;
            }

            for(int i = 0; i < g.get(u).size(); i++) {
                int v = g.get(u).get(i).v;
                int g_uv = g.get(u).get(i).wt;

                if(spt[v] == false) {
                    pq.add(new Node(v, dist_u + g_uv));
                }
            }
        }

        System.out.println("NO");
  }


}

public class Main {
  public static void main(String args[]) {
    Scanner input = new Scanner(System.in);
    int n = input.nextInt(), m = input.nextInt();
    graph g = new graph(n);
    for(int i=0; i < m; i++){
      int u = input.nextInt(), v = input.nextInt();
      int wt = input.nextInt();
      g.addNode(u-1, v-1, wt);
```

```
        }
        int start = input.nextInt(), end = input.nextInt();
        g.findPath(start-1, end-1);
    }
}
```