141124-Problem-10

Title: Bellman Ford Algorithm

Problem Statment

Given a weighted and directed graph of v vertices and edges, Find the shortest distance of all the vertex's from the source vertex, src and return a list of integers where the ith integer denotes the distance of the i-th node from the source node. If a vertices can't be reach from the s then mark the distance as 10^8.

Note: If the graph contains a negative cycle then return an array consisting of only -1.

Input Format

Number of test cases

Number of vertices and number of edges

The next line contains three space separated integers start node, end node, and weight of the edge.

Last Line Source Node

Output

Array representing the shortest distance of all nodes from source is printed.

Example

Input

1

3 4

0 1 5

1 0 3

1 2 -1

2 0 1

2

Output

[1, 6, 0]

Explanation: For nodes 2 to 0, we can follow the path: 2-0. This has a distance of 1. For nodes 2 to 1, we cam follow the path: 2-0-1, which has a distance of 1+5 = 6,

Constraints:

1<=V<=500

1<=E<=V*(V-1)

-10^3<=data of nodes, weight<=10^3

0<=S<=V

Solution

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

class Main {
    public static void main(String args[]) throws IOException {
        BufferedReader read = new BufferedReader(new InputStreamReader(System.in));
        int t = Integer.parseInt(read.readLine());
        while (t-- > 0) {
            String str[] = read.readLine().trim().split(" ");
            int V = Integer.parseInt(str[0]);
            int E = Integer.parseInt(str[1]);

            int[][] edges = new int[E][3];

            for (int i = 0; i < E; i++) {
```

```
            String S[] = read.readLine().trim().split(" ");
            int u = Integer.parseInt(S[0]);
            int v = Integer.parseInt(S[1]);
            int w = Integer.parseInt(S[2]);
            edges[i][0] = u;
            edges[i][1] = v;
            edges[i][2] = w;
        }

        int S = Integer.parseInt(read.readLine());

        Solution ob = new Solution();
        int[] ptr = ob.bellmanFord(V, edges, S);

        System.out.println(Arrays.toString(ptr));
        // System.out.println();
    }
  }
}




class Solution {
   public int[] bellmanFord(int V, int[][] edges, int src) {
      // Initialize distances from the source to all vertices as a large value (infinity)
      int[] dis = new int[V];
      Arrays.fill(dis, (int) 1e8); // Using a large value to represent infinity
      dis[src] = 0; // Distance from source to itself is always 0

      // Relax all edges |V| - 1 times
      for (int i = 0; i < V - 1; i++) {
         for (int[] edge : edges) {
            int u = edge[0]; // Start vertex of the edge
            int v = edge[1]; // End vertex of the edge
            int wt = edge[2]; // Weight of the edge

            // Relaxation step
            if (dis[u] != (int) 1e8 && dis[u] + wt < dis[v]) {
               dis[v] = dis[u] + wt;
            }
         }
      }
```

```
        // Check for negative-weight cycles in the nth iteration
        for (int[] edge : edges) {
            int u = edge[0];
            int v = edge[1];
            int wt = edge[2];

            if (dis[u] != (int) 1e8 && dis[u] + wt < dis[v]) {
                return new int[]{-1}; // Return {-1} to indicate a negative cycle
            }
        }

        return dis; // Return the final distances from the source
    }
}
```