https://course.acciojob.com/idle?question=30eaae97-49d8-4a2b-a6ac-6f8ce9e2281f

- MEDIUM
- Max Score: 40 Points
-
-
-
-
-

Network Delay Time

You are given a network in the form of a directed graph `G`. Here, the edge `(u,v,w)` implies that data travels from computer `u` to computer `v`, and takes a time of `w` seconds.

Assuming the data originates from computer `1`, determine the time it takes for data to reach all other computers in the network. If data cannot reach a computer, return `-1`.

Input Format

The first line contains two integers `n` and `m`, the number of nodes and edges in the graph, respectively. Each of the next `m` lines contains three integers `u`, `v`, and `w`, describing an edge connecting node `u` to node `v` with weight `w`.

Output Format

Output a single integer, the time it takes for data to reach all other computers in the network. If data cannot reach a computer, return `-1`.

Example 1

Input

```
3 4
```

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

Output

```
5
```

Explanation

The shortest path from `1` **to** `2` **is** `1 -> 3 -> 2`**, and takes a time of** `5` **seconds. All other paths take less time, so the answer is** `5`**.**

Example 2

Input

```
3 1
1 2 1
```

Output

```
-1
```

Explanation

There is no path from `1` **to** `3`**, so the answer is** `-1`**.**

Constraints

- `1 <= n <= 100000`
- `0 <= m <= 100000`
- `-1e9 <= w <= 1e9`

Topic Tags

- Graphs

My code

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

public class Main {
    public static void main(String[] args) throws Throwable {
        Scanner sc = new Scanner(System.in);
        int v,e;
        v = sc.nextInt();
        e = sc.nextInt();
        //Create adjacency list of edges
        LinkedList<List<Integer>> adj[] = new LinkedList[v+1];
        for(int i=0;i<=v;i++)
            adj[i] = new LinkedList<>();

        for(int i=0;i<e;i++){
            int u,v1,w;
            u = sc.nextInt();
            v1 = sc.nextInt();
            w = sc.nextInt();
            adj[u].add(Arrays.asList(v1,w));
        }
        Solution obj = new Solution();
        System.out.println(obj.delayTime(v, adj));

    }
}
```

```
class Solution{
    static int delayTime(int V, LinkedList<List<Integer>> adj[]){
            PriorityQueue<int[]> pq = new PriorityQueue<>(new
Comparator<int[]>() {
                public int compare(int[] a, int[] b) {
                    return Integer.compare(a[1], b[1]);
                }
            });

            int n = V + 1;

            int dist[] = new int[n];
            boolean vis[] = new boolean[n];

            Arrays.fill(dist, Integer.MAX_VALUE);
            Arrays.fill(vis, false);

            pq.add(new int[]{1, 0});

            while(pq.size() > 0) {
                int[] curr = pq.poll();

                int u = curr[0];
                int dist_u = curr[1];

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

                vis[u] = true;
                dist[u] = dist_u;
                for(int i = 0; i < adj[u].size(); i++) {
                    int v = adj[u].get(i).get(0);
                    int g_uv = adj[u].get(i).get(1);

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

            int ans = -1;
            for(int i = 1; i < n; i++) {
                if(vis[i] == false) return -1;
                ans = Math.max(ans, dist[i]);
            }

            return ans;

    }
}
```