Problem: LOJ 1108 - Traffic Problem (Online Judge)

https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1390

Problem Description
This problem deals with finding the shortest path in a weighted graph.
You are given:

1. N cities
2. M roads
3. Every road has a weight (which may be positive or negative)
4. Some roads are two-way, and some are one-way

Goal
Find the shortest path from city 1 to city N.

Graph Details

1. Nodes: cities 1 to N
2. Edges: roads between cities

There are two types of roads:

1. Two-way road:
   You can travel both directions
   Weight = w
2. One-way road:
   You can travel only in one direction
   Weight = w

Objective
Compute the minimum total distance/time required to travel from city 1 → city N.

Why Use Bellman-Ford?
Bellman-Ford is chosen because:

Supports Negative Weights
Some roads may have negative weights.
Dijkstra cannot handle graphs that contain negative edges reliably,
but Bellman-Ford works safely even with negative weights.

Correctness

Bellman-Ford relaxes all edges repeatedly.
This ensures that if a shorter path exists, the distance array gets updated step by step.

Flexible Graph Types

Works with:

- Mixed one-way and two-way edges
- Positive and negative weights
- Disconnected graphs

Suitable for LOJ Constraints

The constraints in LightOJ (≤ 500 nodes, ≤ 1000 edges) are small enough that Bellman-Ford runs efficiently without performance issues

.

Features of the Solution

1. Handles negative edges safely
2. Computes the shortest path distance from source (1) to destination (N)
3. Works for a mix of one-way and two-way roads
4. Efficient and accurate for the input size provided in LOJ
5 .Ensures correctness even in the presence of tricky edge cases

Pseudocode:

```
FUNCTION bellman_ford(edges, N, start):

    // Step 1: Initialize distances
    DEFINE dist[1..N] = INFINITY
    dist[start] = 0

    // Step 2: Relax all edges (N - 1) times
    FOR i = 1 TO N - 1:
        FOR each edge (u, v, w) in edges:
            IF dist[u] + w < dist[v]:
                dist[v] = dist[u] + w

    // Step 3: Optional negative-cycle detection
    FOR each edge (u, v, w) in edges:
        IF dist[u] + w < dist[v]:
            RETURN "Negative cycle exists"

    // Step 4: Return shortest distance to destination N
    RETURN dist[N]
```