Problem-02: UVA 558 - Wormholes (Online Judge)

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

Problem Description

This problem focuses on detecting negative weight cycles in a graph.
In the story context, wormholes allow time travel by reducing time (negative time cost).
We must determine whether time travel is possible or not.
You are given:

N cities
M two-way roads
W wormholes (one-way time tunnels)

Goal

Check whether the graph contains a negative weight cycle, which would indicate that using wormholes, it is possible to travel back in time indefinitely.

Graph Details

1. Nodes = cities 1 to N
2. Normal roads:
   Undirected (two-way)
   Have positive weight
3. Wormholes:
   Directed (one-way)
   Have negative weight (they reduce time)
4. The graph is a combination of:
   Positive edges (roads)
   Negative edges (wormholes)

Objective

Determine if any negative weight cycle exists in the graph.
If yes → Time travel is possible
If not → Time travel is impossible

Why Use Bellman–Ford?

Bellman-Ford is ideal because:

Handles Negative Weights

Dijkstra fails whenever there is a negative-weight edge.
Bellman-Ford works properly with both positive and negative edge weights.

Detects Negative Cycles

By relaxing edges N times, if any edge can still be relaxed on the Nth iteration:
→ A negative cycle exists
→ Time travel is possible (the traveler can loop indefinitely and reduce time endlessly)

Well-suited for UVA Constraints
With ≤ 500 nodes and limited edges, Bellman-Ford runs efficiently and reliably for this problem.

Features of the Solution

1. Correctly handles negative weights from wormholes
2. Detects whether a negative cycle exists anywhere in the graph
3. Works efficiently for the typical UVA input sizes
4. Supports mixed graphs:
   Positive weighted roads
   Negative weighted wormholes
   Directed + undirected edges combined
5. Simple and robust approach to determine time travel feasibility

Pseudocode:

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

  DEFINE dist[1..N] = INFINITY
  dist[start] = 0

  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

  FOR each edge (u, v, w) in edges:
    IF dist[u] + w < dist[v]:
      RETURN True

  RETURN False
```