UVA 558

UVA 558 – Wormholes

This problem involves determining whether a scientist can travel indefinitely **back in time** using a network of wormholes. Wormholes are modeled as **directed edges** between star systems with **time shifts** as weights.

The core challenge is to detect if there exists a **negative weight cycle reachable from the starting star system (node 0)**. Such a cycle would allow the scientist to loop repeatedly, going further back in time with each traversal.

The solution models the star systems as **nodes in a directed graph** and each wormhole from system `x` to `y` with time change `t` as a **directed edge with weight t**. Using the **Bellman-Ford algorithm**, negative cycles are detected by relaxing all edges for **n iterations** (where n = number of nodes). If any distance can still be reduced in the nth iteration, it confirms a negative cycle reachable from the start, resulting in the output `"possible"`. Otherwise, output `"not possible"`.

Steps

1. **Understand the Problem Requirements**
    - Determine if there exists a cycle of wormholes with **negative total time change**.
    - The cycle must be **reachable from start node 0**.
    - A negative cycle allows traveling **back in time indefinitely**.
2. **Model the Scenario as a Graph**
    - Represent each star system as a **node** in a directed graph.
    - Represent each wormhole as a **directed edge** with weight = time change `t`.
    - Positive weights → forward in time, negative weights → backward in time.
3. **Apply Bellman-Ford Concept**
    - Initialize distances: `dist[0] = 0`, all others = ∞.

    - Relax all edges **n-1 times**, updating the earliest reachable times for each node.

4. **Detect Negative Cycles**
    - Perform an additional (nth) iteration over all edges.
    - If any distance can still be reduced, it indicates a **negative cycle reachable from node 0**.

5. **Interpret the Result**
    - If a negative cycle exists → print `"possible"`.
    - If no negative cycle → print `"not possible"`.

Key Points

- Bellman-Ford works even with **negative weights**.
- Cycle must be **reachable from start node 0**, so distances initialized carefully.
- The **nth iteration** is critical to detect negative cycles.
- Complexity: O(n * m), where n = nodes, m = edges.

Input:

4 4

0 1 10

1 2 20

2 3 30

3 0 -60

Execution:

Edges stored as (u, v, weight):

• Edge 0: (0, 1, 10) • Edge 1: (1, 2, 20)

• Edge 2: (2, 3, 30)

• Edge 3: (3, 0, -60)

Step 1: Initialization

• dist[0] = 0

• dist[1] = INF • dist[2] = INF

• dist[3] = INF

Step 2: First Iteration (i=0) - Relax all edges

• Edge 0: dist[0] + 10 = 10 < INF → dist[1] = 10

• Edge 1: dist[1] + 20 = 10 + 20 = 30 < INF → dist[2] = 30

• Edge 2: dist[2] + 30 = 30 + 30 = 60 < INF → dist[3] = 60

• Edge 3: dist[3] + (-60) = 60 + (-60) = 0 ≥ dist[0] → no change

After iteration 1: dist = [0, 10, 30, 60]

Step 3: Second Iteration (i=1) - Relax all edges

• Edge 0: dist[0] + 10 = 10 ≥ dist[1] → no change

• Edge 1: dist[1] + 20 = 10 + 20 = 30 ≥ dist[2] → no change

• Edge 2: dist[2] + 30 = 30 + 30 = 60 ≥ dist[3] → no change

• Edge 3: dist[3] + (-60) = 60 + (-60) = 0 ≥ dist[0] → no change

After iteration 2: dist = [0, 10, 30, 60] (no changes)

Step 4: Third Iteration (i=2) - Relax all edges

• All edges: no changes (same calculations as iteration 2)

After iteration 3: dist = [0, 10, 30, 60] (no changes)

Step 5: Fourth Iteration (i=3) - Check for negative cycles

• Edge 0: 0 + 10 = 10 ≥ 10 → no change

• Edge 1: 10 + 20 = 30 ≥ 30 → no change

• Edge 2: 30 + 30 = 60 ≥ 60 → no change

• Edge 3: 60 + (-60) = 0 ≥ 0 → no change

Step 6: Analysis

• No distance updates occurred in the 4th iteration

• Therefore, no negative cycle reachable from node 0 exists

Output:

not possible

Pseudocode:

```
READ c
FOR each case:
   READ n, m
   edges = []
   FOR m times:
      READ u, v, w
      store edge (u, v, w) in edges

   dist = [INF] * n
   dist[0] = 0

   REPEAT n times:
      FOR each edge (u, v, w) in edges:
         IF dist[u] + w < dist[v]:
            dist[v] = dist[u] + w

   negative_cycle = false
   FOR each edge (u, v, w) in edges:
```

```
        IF dist[u] + w < dist[v]:
            negative_cycle = true

    OUTPUT "possible" if negative_cycle else "not possible"
```

Code:

https://github.com/shanto470/algorithm_plm/blob/main/BellmanFord/UVA%20558/uva558.cpp