
Problem: *UVA558 — Wormholes problem in Onlinejudge*
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=499

Mapping the Universe: Is Time Travel Possible via Wormholes?

The year is 2163. Wormholes have been discovered, opening pathways for rapid transit between star systems. However, these tunnels don't just bend space—they bend time. Some wormholes might send you 15 years into the future, while others might deposit you 42 years into the past!

Our brilliant physicist, whose goal is to witness the Big Bang, knows that the only way to travel indefinitely into the past is to find a specific wormhole cycle. By repeatedly traversing this loop, she could go back as far in time as necessary.

The central question is: Does such a Time Loop exist in the vast routing map of the universe? Let's explore how our C++ program uses algorithms to find the answer.

1. Transforming the Problem into a Graph

This exciting science fiction concept translates perfectly into a classic computer science "shortest path" problem:

1. **Nodes or Vertices (Star Systems):** We treat every star system as a point or node. Earth is designated as Node `0`.
2. **Edges (Wormholes):** Every one-way wormhole is a directed edge connecting the nodes.
3. **Weights (Time Difference):** The weight of the edge is the time difference experienced when passing through the wormhole.
    - Traveling into the future (+1000 years) = Positive Weight.
    - Traveling into the past (-42 years) = Negative Weight.

Therefore, searching for a way to travel indefinitely into the past is equivalent to searching for a **Negative Weight Cycle** (a loop where the sum of all edge weights is negative).

2. Bellman-Ford: The Time Loop Detective

Among the various graph algorithms, the Bellman-Ford Algorithm is uniquely suited for finding the shortest paths in a graph that may contain negative weights. Critically, it is the ideal tool for detecting the presence of negative cycles, which is exactly what we need to solve the scientist's dilemma.

Why not Dijkstra's Algorithm?

While Dijkstra's algorithm is faster, it fails when negative edge weights are involved. Bellman-Ford overcomes this limitation and, most importantly, provides the definitive check for the existence of a negative cycle.

3. How the Algorithm Works

Assuming our system has N star systems, the Bellman-Ford process operates in two primary stages:

A) N-1 Iterations of Relaxation

The algorithm iterates through all edges in the graph N-1 times. In each iteration, it performs a relaxation step, checking if a shorter path to any node can be found by passing through the current edge.

If the graph contains no negative cycles, the shortest paths are guaranteed to be found and stabilized after N-1 passes.

B) The N-th Final Check

This is the decisive step. After N-1 passes, if we can still relax any edge (meaning, we find a path that further reduces the total time/weight, i.e., `dist[start] + time < dist[end]`), it unequivocally proves that:

A Negative Weight Cycle Exists!

If distances continue to decrease in the final iteration, it means we are continuously cycling through a time loop, reducing the total time difference further into the past with every loop.

4. The Result

- **If a Negative Cycle is Found (`return 1`):** The scientist has found her loop and can travel backward in time indefinitely. The output is: **possible**.
- **If No Negative Cycle is Found (`return 0`):** There is no viable time loop in the wormhole network. Her journey will either take her forward in time or stabilize. The output is: **not possible**.

Step-by-Step Trace of the First Test Case

Let's trace the Bellman-Ford algorithm on the first sample case to see how the negative cycle is detected.

Input Data:

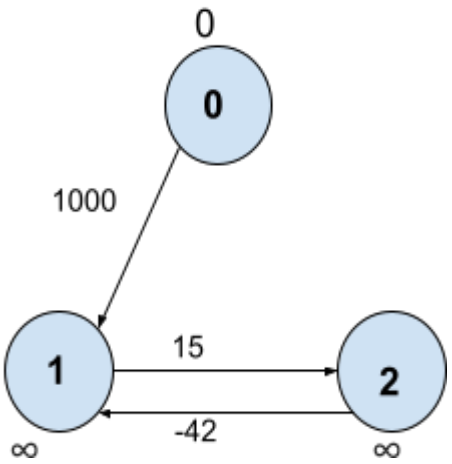
- Systems (N=3): 0, 1, 2

- Wormholes (M=3):
    - E1: 0 → 1, Time +1000
    - E2: 1 → 2, Time +15
    - E3: 2 → 1, Time -42

The path 1 → 2 → 1 forms a cycle with a total time difference of 15 + (-42) = -27 years. This is a negative cycle.

Initial State

Node (System)	**Initial Time Difference (dist)**
0 (Earth)	0
1	INF
2	INF
dist	**[0, INF, INF]**

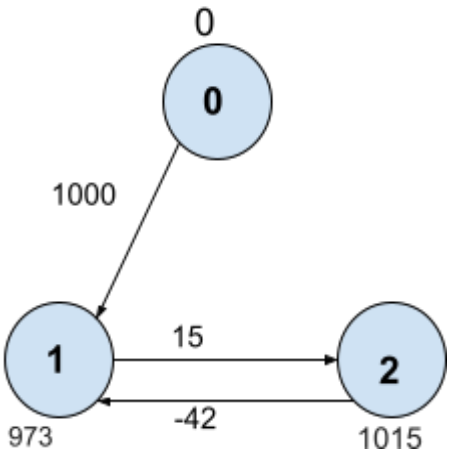


Pass 1 (Relaxation 1 of 2)

We iterate through all 3 edges:

1. **E1 (**0 → 1**, +1000):** 0 + 1000 < INF. Relaxed! `dist[1]` becomes 1000.
2. **E2 (**1 → 2**, +15):** 1000 + 15 < INF. Relaxed! `dist[2]` becomes 1015.
3. **E3 (**2 → 1**, -42):** 1015 - 42 < 1000. (973 < 1000). Relaxed! `dist[1]` becomes 973.

Node (System)	**End of Pass 1 (dist)**
0 (Earth)	0
1	973
2	1015
dist	**[0, 973, 1015]**

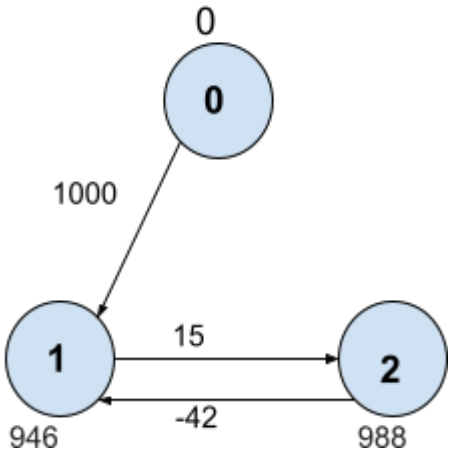


Pass 2 (Relaxation 2 of 2)

We iterate through all 3 edges:

1. **E1 (**0 → 1**, +1000):** 0 + 1000 ≨ 973. No change.
2. **E2 (**1 → 2**, +15):** 973 + 15 = 988. 988 < 1015. Relaxed! `dist[2]` becomes 988.
3. **E3 (**2 → 1**, -42):** 988 - 42 = 946. 946 < 973. Relaxed! `dist[1]` becomes 946.

Node (System)	**End of Pass 2 (dist)**
0 (Earth)	0
1	946
2	988
dist	**[0, 946, 988]**



Final Check (The 3rd Iteration, N-th Pass)

Since N=3, we run a final check (the 3rd iteration) to detect a negative cycle. We only care if *any* relaxation occurs.

1. **E1 (**0 → 1**, +1000):** 0 + 1000 ≦ 946. No change.
2. **E2 (**1 → 2**, +15):** 946 + 15 = 961. 961 < 988. RELAXATION OCCURS! `dist[2]` becomes 961.
    - **STOP:** A relaxation in the N-th pass confirms the existence of a negative cycle.

Conclusion: Because an edge was relaxed in the third pass, the Bellman-Ford algorithm determines that a negative cycle is reachable from system 0. The program outputs **possible**.

5. Pseudocode

The pseudocode outlines the steps necessary to initialize the distances, perform the N-1 relaxation passes, and finally check for the presence of a negative cycle.

Let G = (V, E) be the graph, where V$ is the set of vertices (star systems, |V|=N) and E is the set of edges (wormholes, |E|=M). dist[v] stores the shortest time difference from the source (System 0) to vertex v.

```
FUNCTION BELLMAN_FORD(G, N, M, E)

    // 1. Initialization
    FOR each vertex v IN V:
        dist[v] = INF // Initialize all distances to infinity
    dist[0] = 0       // Set distance for the source vertex (System 0/Earth) to zero

    // 2. Relaxation Passes (Run N-1 times)
    // N is the number of vertices. A path can have at most N-1 edges.
    FOR i FROM 1 TO N - 1:
        relaxed_in_pass = FALSE
        FOR each edge (u, v, t) IN E (where t is the time/weight):
            IF dist[u] is NOT INF:
                IF dist[u] + t < dist[v]:
                    dist[v] = dist[u] + t
                    relaxed_in_pass = TRUE

        // Optimization: If no distance changed in a pass, paths are stable.
        IF relaxed_in_pass is FALSE:
            BREAK

    // 3. Final Check for Negative Cycle (The N-th Pass)
    FOR each edge (u, v, t) IN E:
        IF dist[u] is NOT INF:
            IF dist[u] + t < dist[v]:
                // If we can still relax an edge, a negative cycle exists.
                RETURN 1 // Possible

    // If no negative cycle was found after the N-th check.
    RETURN 0 // Not Possible
```

6. Time Complexity Analysis

The time complexity of the Bellman-Ford algorithm is determined by the number of times we iterate through the edges.

1. **Initialization:** Setting all N distances takes O(N) time.
2. **Relaxation Passes:** This is the main performance factor.
    - The outer loop runs a maximum of N-1 times.
    - The inner loop iterates over all M edges in the graph.
    - Total time for N-1 passes: O((N-1) . M), which simplifies to O(N . M).
3. **Final Check:** The cycle check iterates over all M edges one last time. This takes O(M) time.

Final Complexity

Combining these steps, the overall time complexity of the Bellman-Ford algorithm is:

$O(N) + O(N \cdot M) + O(M) = O(N \cdot M)$

Where:

- N is the number of star systems (vertices, $1 \leq N \leq 1000$).
- M is the number of wormholes (edges, $0 \leq M \leq 2000$).

Performance with Constraints

Given the problem constraints ($N \leq 1000$ and $M \leq 2000$):

$O(1000 \times 2000) = O(2,000,000)$

Two million operations is a very small number for modern computers, meaning the solution is highly efficient and will run almost instantaneously, even for the maximum given inputs.