1. Problem Statement

You are given a directed graph representing star systems connected by wormholes.
Each wormhole goes one way, and when you travel through it, you arrive t years in the future (if $t > 0$) or t years in the past (if $t < 0$).
The travel time through a wormhole is negligible.

You start from star system 0 (our Solar System). It is guaranteed that every star system is reachable from system 0 through some sequence of wormholes.

The goal is to determine whether there exists any cycle reachable from star system 0 such that the total time change along the cycle is negative.
If such a cycle exists, then by looping through it repeatedly, one can go back in time indefinitely — making it possible to reach the Big Bang.

For each test case, print:

- "possible" — if there exists a reachable negative time cycle
- "not possible" — otherwise

2. **Hint**

This problem can be transformed into checking whether a negative-weight cycle exists in a directed graph that is reachable from the starting node (0).
Interpret each time constraint t as the weight of an edge between two nodes.
Which well-known shortest-path algorithm is specifically designed to detect negative cycles that can be reached from a given source node?

3. Solution Approach

The problem is a direct application of the Bellman–Ford algorithm, which detects negative cycles in a weighted directed graph.

Key observations:

- Each wormhole is a directed edge with weight = time shift.
- You start from node 0.
- You're looking for any negative cycle reachable from node 0.

- Bellman–Ford relaxes edges (N − 1) times to find shortest paths.
- A further relaxation on the Nth pass indicates a negative cycle.

Steps:

1. Initialize distances with dist[0] = 0 and all others as INF.
2. Relax all edges for N − 1 iterations.
3. If no update occurs in an iteration, you can stop early (optimization).
4. On the Nth iteration, if any edge can still be relaxed, a negative cycle exists.

Output:

- If negative cycle detected → "possible"
- Otherwise → "not possible"

4.Pseudocode:

```
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
```

5. **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.