Bellman-Ford

1. Definition

The **Bellman–Ford algorithm** is a single-source shortest path algorithm that computes the minimum distance from a starting node to all other nodes in a weighted graph.
Unlike Dijkstra's algorithm, **Bellman–Ford can handle negative weight edges** and can also **detect negative weight cycles**.
It works on both:

- Directed graphs ✓
- Undirected graphs (if edges are added twice) ✓

2. Solution Approach (How Bellman–Ford Works)

Key Idea

Bellman–Ford performs *edge relaxation* repeatedly.
Relaxation means:
If the distance to vertex **v** can be improved using edge (u → v), update it.

Algorithm Steps

1. **Initialize distances**
   - dist[source] = 0
   - dist[others] = +∞
2. **Relax all edges exactly (V − 1) times**
   - For every edge (u, v, w), check if
     dist[v] > dist[u] + w
     → if yes, update.

Why **V − 1 times**?

   - The longest possible simple path in a graph has (V − 1) edges.

3. **Check for Negative Weight Cycles**
   - Run one more relaxation round.
   - If any distance still improves → cycle exists.

3. Pseudocode (C++ Style)

```
// Bellman-Ford Algorithm
void bellmanFord(int n, int m, int src, vector<Edge>& edges) {
   const long long INF = 1e18;
   vector<long long> dist(n + 1, INF);

   dist[src] = 0;

   // Step 1: Relax all edges (n-1) times
   for (int i = 1; i <= n - 1; i++) {
      for (auto &e : edges) {
         int u = e.u;
         int v = e.v;
         long long w = e.weight;

         if (dist[u] != INF && dist[v] > dist[u] + w) {
            dist[v] = dist[u] + w;
         }
```

```
        }
    }

    // Step 2: Check for negative weight cycle
    for (auto &e : edges) {
        int u = e.u;
        int v = e.v;
        long long w = e.weight;

        if (dist[u] != INF && dist[v] > dist[u] + w) {
            cout << "Negative weight cycle detected!\n";
            return;
        }
    }

    // Print distances
    for (int i = 1; i <= n; i++) {
        cout << "Distance to " << i << ": " << dist[i] << "\n";
    }
}
```

Edge Structure

```
struct Edge {
    int u, v;
    long long weight;
};
```

4. Time Complexity

Time Complexity: O(V × E)

Because:

- You relax all **E edges**,
- For **(V – 1)** iterations → approximately **V × E**.

Space Complexity: O(V)

Only the distance array is stored.