Bellman-Ford Algorithm

Definition:
The **Bellman-Ford Algorithm** is a graph algorithm used to find the **shortest paths** from a **single source node** to all other nodes in a **weighted graph**, even if some edges have **negative weights**.

Key Points

1.Single Source Shortest Path:
Finds shortest distances from one starting node to all others.

2.Handles Negative Weights:
Works correctly even if some edges have negative values, unlike Dijkstra.

3.Detects Negative Cycles:
If a cycle reduces distance endlessly (negative cycle), Bellman-Ford can detect it.

4.Based on Relaxation:
The algorithm repeatedly tries to improve the shortest distance to each node.

5.Time Complexity:
O(V × E), where V = number of vertices, E = number of edges.

Part 1: What problem does Bellman-Ford solve?

1.Find shortest distance from one node to all others

2.Works for graphs with **negative edge weights**

3.Can **detect negative cycles**

Example:
Road network where some roads give “negative cost” (like discounts) — Bellman-Ford can handle it.

Part 2: Main Idea (Step-by-Step)

1.Start with **distance of source = 0**

2.Set all other distances = **infinity**

3.Repeat **V-1 times** (V = number of vertices):

For every edge (u → v with weight w):

If dist[u] + w < dist[v]:

→ update dist[v] = dist[u] + w

4.After V-1 iterations, check all edges:

If you can **still relax** any edge → **negative cycle exists**

Part 3: Key Concept

Relaxation: Trying to improve the shortest distance

Why V-1 times?
→ In a graph with V nodes, the longest possible path without a cycle has at most **V-1 edges**

Part 4: Step-by-Step Example

Graph:

```
      4
    A ------> B
   / \        |
2/    \ -2    | 3
 /      \     v
C -------> D
    2
```

Edges with weights:

A → B = 4

A → C = 2

C → D = 2

B → D = 3

A → D = -2

Step 1: Initialize distances

dist[A] = 0
dist[B] = ∞
dist[C] = ∞
dist[D] = ∞

Step 2: First Iteration (Relax all edges)

A → B: 0 + 4 < ∞ → dist[B] = 4

A → C: 0 + 2 < ∞ → dist[C] = 2

C → D: 2 + 2 < ∞ → dist[D] = 4

B → D: 4 + 3 = 7 → dist[D] = min(4,7) = 4

A → D: 0 + (-2) < 4 → dist[D] = -2

After 1st iteration:

A=0, B=4, C=2, D=-2

Step 3: 2nd, 3rd… up to V-1 iterations

Distances remain the same → no further updates

Part 5: Diagram

```
     4
   A ------> B
   | \        |
  2|   \ -2    |3
   |     \      v
   v       C -----> D
             2
```

Part 6: Pseudocode

```
function BellmanFord(Graph, source):
   for each node v:
      dist[v] = ∞
   dist[source] = 0

   for i = 1 to V-1:
      for each edge (u, v) with weight w:
         if dist[u] + w < dist[v]:
            dist[v] = dist[u] + w
```

```
for each edge (u, v) with weight w:
    if dist[u] + w < dist[v]:
        print("Graph contains negative weight cycle")
        return

print("Shortest distances from source:", dist[])
```