Bellman–Ford Algorithm

The **Bellman–Ford algorithm** is a classic shortest-path algorithm used in graph theory. It is especially useful when graphs contain **negative edge weights**, something that algorithms like Dijkstra's cannot handle safely.

What Is the Bellman–Ford Algorithm?

Bellman–Ford is a shortest path algorithm that finds the minimum distance from a **single source** node to all other nodes in a weighted graph.

It works even when:

- Edges have **negative weights**
- The graph contains cycles

However, it **cannot** handle graphs with **negative weight cycles** that are reachable from the source — but it can detect them.

When Should You Use Bellman–Ford?

You choose Bellman–Ford over other algorithms when:

- The graph has **negative edge weights**
- You need to detect **negative cycles**
- You don't need extremely fast performance (Bellman–Ford is slower than Dijkstra)

Common applications:

- Currency arbitrage detection (negative cycles represent profit opportunities)
- Routing protocols (e.g., RIP)
- Optimizing paths in networks where weights may decrease

Key Concepts You Need to Know

1. Weighted Graph

Bellman–Ford works on a weighted graph where each edge has a cost (positive or negative).

2. Relaxation

This is the heart of the algorithm.

To "relax" an edge (u → v with weight w) means:

- If the shortest known path to v can be improved by going through u, update it.

3. Negative Weight Cycle

A cycle whose edges sum to a negative value.

Bellman–Ford can detect this by trying one more relaxation step after finishing the main loop.

How Bellman–Ford Works (Step-by-Step)

1. Initialize all distances to infinity, except the source which is set to **0**.
2. Repeat the relaxation process **(V − 1)** times (where V = number of vertices):
    - For every edge, try to update the shortest path.
3. Perform one more relaxation loop:
    - If any distance updates again, it means there is a **negative cycle**.

Bellman–Ford Pseudocode

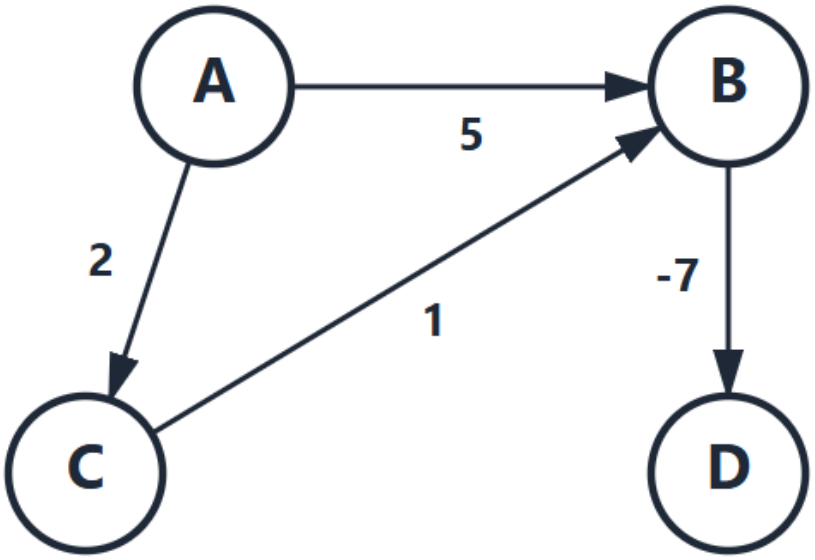
```
BellmanFord(Graph, source):
   for each vertex v in Graph:
      distance[v] = infinity
   distance[source] = 0

   for i from 1 to |V| - 1:
      for each edge (u, v, w) in Graph:
         if distance[u] + w < distance[v]:
            distance[v] = distance[u] + w

   // Check for negative weight cycle
   for each edge (u, v, w) in Graph:
      if distance[u] + w < distance[v]:
         print("Negative weight cycle detected")
         return

   return distance
```

Example (Simple Understanding)



- Start with distance to A = 0, others = ∞
- Relax edges 3 times (V−1 = 4−1)
- Each round improves distances until the shortest values are found

If any further improvement is possible in the extra round, you know a negative cycle exists.

What You Need to Implement Bellman–Ford

- A list of **edges** (edge list is ideal)
- A distance array
- The number of vertices V

Graph representations:

- **Edge list** (best for Bellman–Ford)
- Adjacency list (usable, but requires converting to edges)

Time & Space Complexity

- **Time complexity:** O(V × E)
- **Space complexity:** O(V)

This makes Bellman–Ford slower than Dijkstra, but far more flexible.

Final Thoughts

Bellman–Ford may not be the fastest, but it is one of the most **important** graph algorithms due to its ability to handle **negative weights** and detect **negative cycles**.

Once you understand Bellman–Ford, you're ready to explore:

- Dijkstra's Algorithm
- Floyd–Warshall (all-pairs shortest path)
- Johnson's Algorithm