Bellman-Ford

What is Bellman-Ford ?

Bellman-Ford is an algorithm in computer science, named after Richard Bellman and Lester Ford Jr., which is used to find the shortest path from a single source to all other vertices in a weighted graph. Unlike some other algorithms, it can handle graphs with negative weight edges.

Main Ideas of Bellman-Ford

Purpose: It finds the shortest path from one starting point (source) to all other points (vertices) in a graph.

Handles negative weights: It works even if some edges have negative values.

Relaxation: The algorithm repeatedly updates the shortest distance to each vertex by checking all edges. If a shorter path is found, it updates the distance.

Iterations: For a graph with `V` vertices, it repeats this update process `(V-1)` times.

Negative cycle detection: After all updates, it checks for cycles where the total weight is negative. If such a cycle exists, the shortest path is not defined.

Time complexity: It is slower than some algorithms like Dijkstra, with **O(V × E)** time, where `V` = number of vertices and `E` = number of edges.

Where Bellman-Ford is used

Shortest Path in Graphs with Negative Weights – Unlike Dijkstra, Bellman-Ford can handle edges with negative weights.

Network Routing Protocols – Used in protocols like Distance Vector Routing (e.g., RIP) to find the best path for data in computer networks.

Detecting Negative Cycles – Helps identify if a graph has a negative weight cycle, which is useful in financial modeling or checking for impossible conditions in networks.

Optimal Path in Road/Transport Networks – Can be applied where costs, tolls, or distances may have negative adjustments (like discounts or rebates).

Dynamic Programming Problems – Its concept of “relaxation” is used in other optimization and DP-based algorithms.

Examples

We have 4 cities: A, B, C, D. The travel costs between them are:

- A → B = 4
- A → C = 5
- B → C = -2
- B → D = 6
- C → D = 1

We want the cheapest cost from A to all cities.

Initialize Costs

- Cost to A = 0 (starting point)
- Cost to B, C, D = ∞

Costs:

- A = 0, B = ∞, C = ∞, D = ∞

Check Negative Cycles

- If any cost can still be lowered → negative cycle exists.
- Here, nothing changes → no negative cycle.

How bellman ford works

1: Start with Initialization

2: Relax All Edges

3: Repeat Relaxation

4: Check for Negative Cycles

5:Read the shortest distances.

Pseudocode :

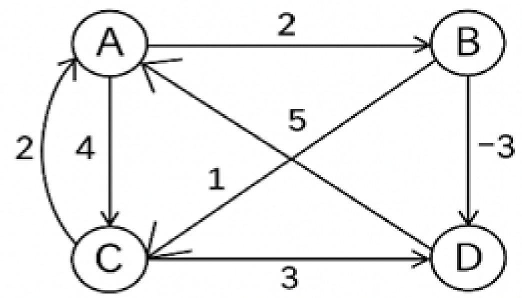
```
            BellmanFord(Graph, source):
for each vertex v in Graph:
    distance[v] = INFINITY
distance[source] = 0

for i = 1 to (number of vertices - 1):
    for each edge (u, v) with weight w in Graph:
        if distance[u] + w < distance[v]:
            distance[v] = distance[u] + w

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

for each vertex v in Graph:
    print "Distance from", source, "to", v, "=", distance[v]
```

Bellman-Ford Diagram



After learning Bellman-Ford :

- Understand how to find shortest paths in a graph.
- Handle graphs with negative weight edges safely.
- Detect negative cycles in a network.
- Apply concepts in network routing (like internet or transport paths).
- Learn relaxation and dynamic programming ideas useful in other algorithms.