3013 Algorithms

Dr. Terry Griffin

Bellman Ford Algorithm Presentation

Jhymani Joseph & Edvin Benjamin

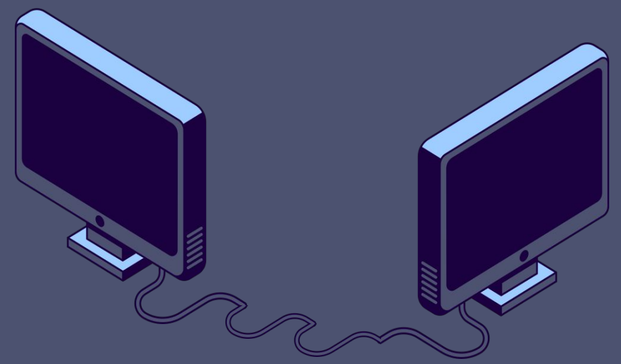
Introduction

The Bellman–Ford algorithm is an algorithm that

computes shortest paths from a single source vertex

to all of the other vertices in a weighted graph, including

graphs with negative edge weights.



History

- Bellman-Ford algorithm has a history rooted in the work of Alfonso Shimbel, Richard Bellman, Lester Ford Jr., and Edward F. Moore.
- Alfonso Shimbel first proposed the work in 1955 and this is considered the initial iteration of the algorithm.
- The Bellman–Ford algorithm was independently proposed by Richard Bellman and Lester Ford Jr. in 1956 and 1958 respectively.
- Ford's work was primarily focused on the max-flow min-cut problem, which is closely related to shortest path problems. Bellman's contribution was focused on the equations that become known as "Bellman's equations.
- Edward F. Moore's work in 1959 provided a variation of the algorithm.

Advantages Of Bellman-Ford

- Handles Negative Edge Weights.
- Detects Negative Cycles

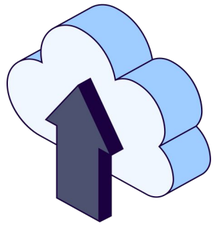
- Foundation for Other Algorithms
- Early Stopping Possible

- Simple and Easy to Implement.
- Works on Directed and Undirected Graphs

- flexible for edge list representations which can save space when dealing with sparse graphs.

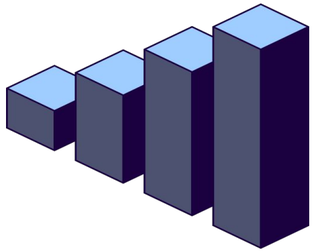
Disadvantages Of Bellman-Ford

Higher Space Complexity



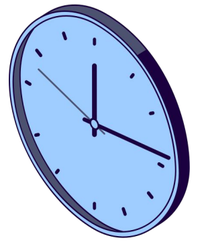
Needs to store all vertices and edges explicitly.This leads to memory issues with large graphs.

Inefficient for Dense Graphs



If there are many edges performance becomes poor because it checks every edge V-1 times.

Slower than Other Algorithms



Time complexity is O(VE), which makes it slower for graphs without negative weights.

Steps of Bellman Ford Algorithm

Step 1: ***Initialization***

Set the distance from the source vertex to itself as 0, and the distance from the source to all other vertices as infinity.

Step 2: ***Relaxation***

Relax all edges (V - 1) times:

For every edge (u → v),

if dist[u] + weight < dist[v]

we update dist[v].

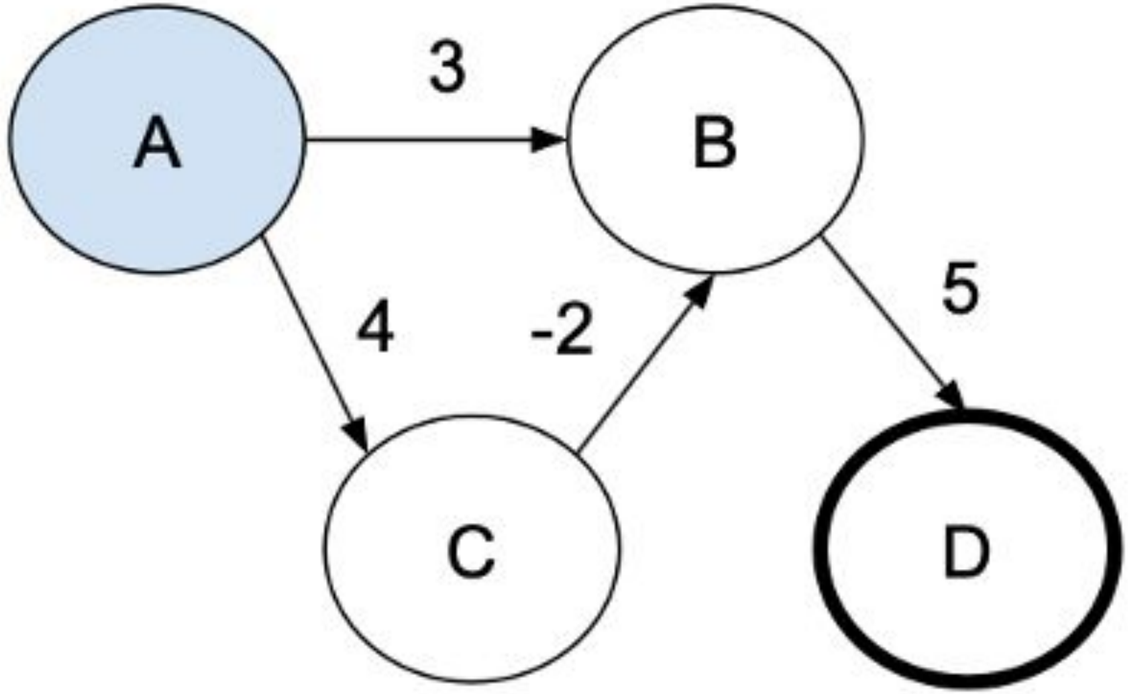
Step 3: ***Check for negative-weight cycles***

After (V - 1) iterations, if a further relaxation is possible, (shorter path exists) it indicates the presence of a negative cycle in the graph.

Step 4: ***Output***

If no negative cycle is detected, the algorithm returns the shortest distances for each vertex. Then reconstruct the shortest paths from the source to all other vertices.

Working example



Step 1: Assuming A is the source.

Relax All Edges V - 1 Times (4 vertices → 3 iterations)

Iteration 1:

A → B (3): 0 + 3 = 3 → dist(B) = 3

A → C (4): 0 + 4 = 4 → dist(C) = 4

C → B (-2): 4 + (-2) = 2 → dist(B) = 2 (updated!)

B → D (5): 2 + 5 = 7 → dist(D) = 7

After Iteration 1:

dist(A) = 0, dist(B) = 2, dist(C) = 4, dist(D) = 7

Iteration 2:

A → B: No change (already 2)

A → C: No change

C → B: 4 + (-2) = 2 → no change

B → D: 2 + 5 = 7 → no change

After Iteration 2:

No updates → early convergence

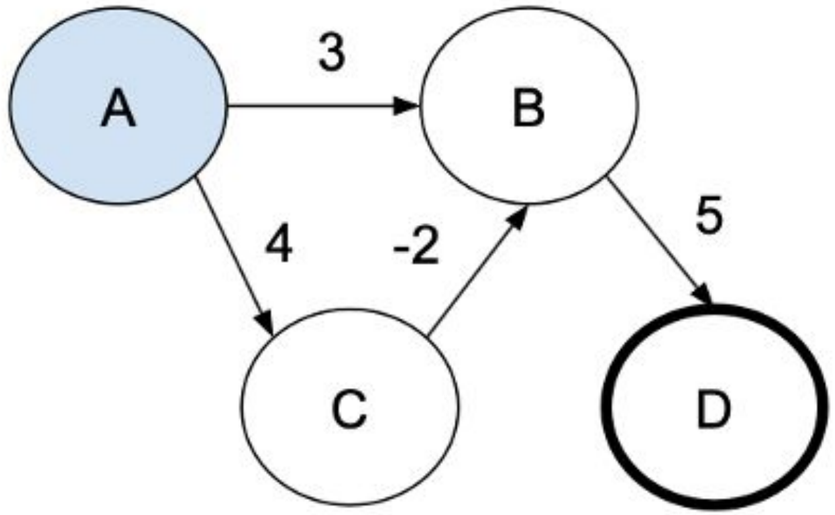
dist(A) = 0, dist(B) = 2, dist(C) = 4, dist(D) = 7

After Iteration 3:

No updates → early convergence

dist(A) = 0, dist(B) = 2, dist(C) = 4, dist(D) = 7

Working example



A -> B = 3
A -> C = 4
C -> B = -2
B -> D = 5

A	B	C	D
A:0	∞	∞	∞
A:0	A:3	∞	∞
A:0	A:3	A:4	∞
A:0	C:2	A:4	∞
A:0	C:2	A:4	B:7

Dijkstra's	Bellman-Ford	Floyd-Warshall
Shortest path from **one** node to all other nodes.	Shortest path from **one** node to all other nodes, including **negative edges**.	Shortest path between **all pairs** of vertices, including **negative edges.**

Different Types of Algorithms

Features of the different Algorithms

Feature	Bellman Ford	Dijkstra	Floyd-Warshall
Purpose	Single-source shortest path	Single-source shortest path	All-pairs shortest path
Negative Edge Weights	Supports negative weights	Cannot handle negative weights	Supports negative weights
Negative Cycle Detection	Detects negative cycles	Does not detect negative cycles	Detects negative cycles
Graph Type	Directed or Undirected (with weights)	Directed or Undirected (positive weights)	Directed (works best with dense graphs)
Time Complexity	$O(V \times E)$	$O(V^2)$ with simple implementation; $O((V+E) \log V)$ with priority queue (e.g., using a heap)	$O(V^3)$

Features of the different Algorithms

Feature	Bellman Ford	Dijkstra	Floyd-Warshall
Space Complexity	O(V)	O(V)	$O(V^2)$
Best For	Graphs with negative weights or cycles	Graphs with non-negative weights	Finding shortest paths between all pairs
Approach	Edge relaxation repeated (V-1 times)	Greedy method using nearest unvisited node	Dynamic programming
Efficiency on Dense Graphs	Less efficient	More efficient (with optimized heap)	Good for very dense graphs
Output	Distance from source to all nodes + cycle info	Distance from source to all nodes	Distance between every pair of nodes

Real World Examples

GPS Navigation:

When planning a road trip, the Bellman- Ford algorithm can be used to find the shortest path between two locations.

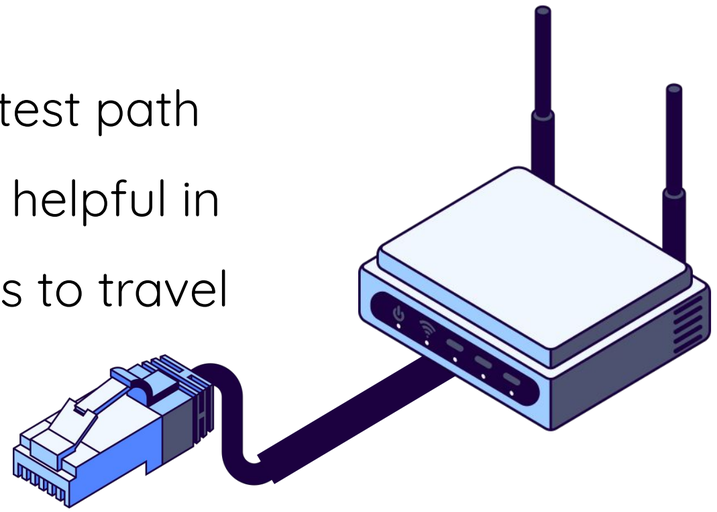
How:

- Updates route choices by checking all possible roads multiple times to find the best (shortest) path.
- Handles negative weights, making it useful if certain routes temporarily become faster or slower.
- Ensures accurate path planning even when road conditions change during the trip.

Real World Examples

Network Routing:

The algorithm can be used to find the shortest path between two network nodes, which can be helpful in determining the best route for data packets to travel through a network.



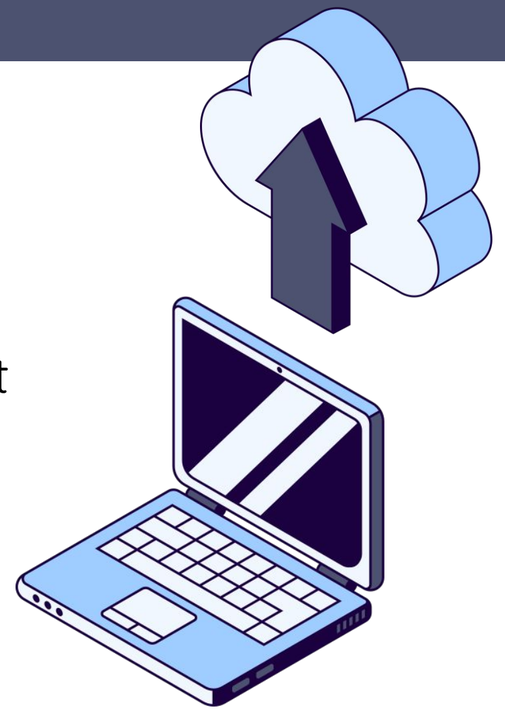
How:

- Routers use it to calculate the best path to each destination.
- Considers link costs like delay, bandwidth, or congestion (similar to traffic in road networks).
- Periodically updates routes by sharing information with neighboring routers.
- Handles dynamic changes (like link failures or new connections) to maintain efficient data flow.

Real World Examples

Financial Applications:

The algorithm can be used to detect opportunities in currency exchange rates, where negative edge weights can represent opportunities to gain or lose by converting currencies.



How:

- Currencies are nodes, and exchange rates are edges with weights.
- Negative cycles indicate a chance to profit by converting through a series of currencies.
- Identifies loops where you end up with more money than you started with.
- Helps financial analysts find and exploit profitable currency exchanges quickly.

Test Questions

1. What is the primary advantage of the Bellman-Ford algorithm over Dijkstra's algorithm?

A. It is faster for all graphs
B. It can detect negative weight cycles
C. It uses less memory
D. It only works with positive weights

Answer: B. It can detect negative weight cycles

Test Questions

2. What is the time complexity of the Bellman-Ford algorithm for a graph with V vertices and E edges?

A. O(V)
B. O(E+V)
C. O(VE)
D. O(E log V)

Answer: C. O(VE)

Test Questions

3. How many times does the Bellman-Ford algorithm relax all the edges in the graph (assuming no early stopping)?

A. V−1
B. E
C. V
D. 2V

Answer: A. V−1

Test Questions

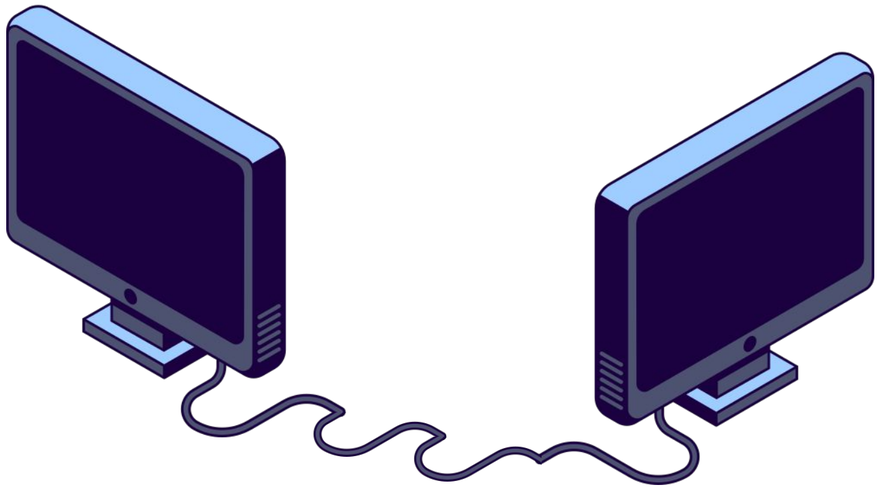
1. The Bellman-Ford algorithm is capable of detecting _____ weight cycles in a graph.

Answer: negative

2. The process of updating the shortest path estimates in Bellman-Ford is known as _____.

Answer: relaxation

Thank You



Any Questions?

Sources

- https://www.simplilearn.com/tutorials/data-structure-tutorial/bellman-ford-algorithm#:~:text=Alfonso%20Shimbel%20proposed%20the%20algorithm,Bellman%2DFord%E2%80%93Moore%20algorithm.
- chatgpt.com
- https://www.google.com/search?q=floyd+warshall+algorithm&rlz=1C1GCEU_enUS1159US1159&oq=floyd+warshall+algorithm&gs_lcrp=EgZjaHJvbWUyDwgAEEUYORiDARixAxiABDIHCAEQABiABDIHCAIQABiABDIHCAMQABiABDIHCAQQABiABDIHCAUQABiABDIHCAYQABiABDIHCAcQABiABDIHCAgQABiABDIHCAkQABiABNIBCDUxMzRqMGo3qAIAsAIA&sourceid=chrome&ie=UTF-8#fpstate=ive&vld=cid:2d603e41,vid:4OQeCuLYj-4,st:0
- https://medium.com/@maryanngitonga/bellman-fords-algorithm-e2344aa21c7b#:~:text=Imagine%20you're%20planning%20a,used%20at%20the%20relaxation%20step.
- https://blog.heycoach.in/bellman-ford-algorithm-in-real-world-applications/