| Feature | Dijkstra's Algorithm | Bellman-Ford Algorithm |
|---|---|---|
| **Approach** | Greedy | Dynamic Programming |
| **Edge Weights Supported** | Only non-negative weights | Handles negative weights (except negative cycles in undirected graphs) |
| **Negative Cycle Detection** | Cannot detect negative cycles | Can detect negative cycles in directed graphs |
| **Graph Types Supported** | Works on directed and undirected graphs with non-negative weights | Works on directed graphs with negative weights; undirected only if all weights non-negative |
| **Time Complexity** | $O(V^2)$ (simple), O((E+V) log V) with binary heap, O(V log V + E) with Fibonacci heap | O(VE) |
| **Space Complexity** | O(V) + priority queue | O(V) (simple arrays for distances) |
| **Path Update Mechanism** | Selects the node with the minimum tentative distance, relaxes its outgoing edges | Relaxes all edges V-1 times, updating distances if a shorter path is found |
| **Algorithm Steps** | - Initialize distances, use priority queue<br>- Pick closest unvisited node<br>- Update neighbors | - Initialize distances<br>- Repeat V-1 times: relax all edges |
| **Distributed Implementation** | Difficult (needs global knowledge of graph) | Easier (can use only local neighbor info, suitable for distributed routing) |
| **Typical Use Cases** | Link-state routing protocols (OSPF, IS-IS), GPS navigation | Distance-vector routing protocols (RIP, IGRP), graphs with negative weights |

| | | |
|---|---|---|
| **Handling of Undirected Graphs** | Works for non-negative weights | Negative weights in undirected graphs lead to negative cycles (problematic) |
| **Optimality** | Always finds the shortest path if all weights are non-negative | Finds shortest path even with negative weights, unless negative cycle exists |
| **Implementation Complexity** | Needs priority queue, more complex data structures | Simpler: just loops and arrays |
| **Limitations** | Fails with negative weights; cannot detect negative cycles | Slower on large graphs; fails on undirected graphs with negative weights |

**Summary of Key Differences:**

- Dijkstra is faster and more efficient for graphs with non-negative weights, but cannot handle negative weights or cycles.
- Bellman-Ford is slower but can handle negative weights and detect negative cycles in directed graphs, making it essential for certain network and optimization problems.
- Dijkstra relies on a global view and priority queue, while Bellman-Ford is amenable to distributed and asynchronous computation.
- Bellman-Ford is more robust in edge cases (negative weights/cycles), while Dijkstra is preferred for performance when the graph is well-behaved (no negative weights).