Bellman-Ford Algorithm in DSA: Beginner to Advanced Guide

Subtitle: Learn how to find shortest paths in weighted graphs with negative edges and detect negative cycles.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

The Bellman-Ford Algorithm is used to compute **shortest paths from a single source in weighted graphs**, including graphs with **negative edge weights**. It also detects **negative weight cycles**, which Dijkstra cannot handle.

Step 1: Core Concepts

- Works on **graphs with negative weights**
- Relaxes all edges **V-1 times** (V = number of vertices)
- Checks for negative weight cycles in the last iteration
- **Time Complexity:** O(V * E)

Why it matters: Essential for graphs where Dijkstra fails due to negative weights.

Step 2: Graph Representation

- Use **edge list** representation for Bellman-Ford

```
let edges = [
  [0,1,4],
  [0,2,5],
  [1,2,-3],
  [2,3,4],
  [1,3,2]
];
let V = 4; // number of vertices
```

- Each edge is [u, v, weight]

Step 3: Implementation in JavaScript

```
function bellmanFord(edges, V, src){
  let dist = Array(V).fill(Infinity);
  dist[src] = 0;

  // Relax all edges V-1 times
  for(let i=0; i<V-1; i++){
    for(let [u,v,w] of edges){
      if(dist[u] !== Infinity && dist[u]+w < dist[v]){
        dist[v] = dist[u]+w;
      }
    }
  }

  // Check for negative weight cycles
  for(let [u,v,w] of edges){
    if(dist[u] !== Infinity && dist[u]+w < dist[v]){
      console.log("Graph contains a negative weight cycle");
      return;
    }
  }

  return dist;
}

// Example usage
let distances = bellmanFord(edges, V, 0);
console.log(distances); // [0,4,1,6]
```

Step 4: Applications

1. **Graphs with negative edges**
2. **Currency arbitrage detection** (negative cycles)
3. **Shortest path in transport or network optimization**
4. **Competitive programming problems with weighted graphs**

Step 5: Mini Exercises

1. Implement Bellman-Ford for a weighted graph
2. Detect negative cycles in a given graph
3. Find shortest paths from node 0 with negative edges
4. Solve a competitive programming problem where Dijkstra fails

Step 6: Common Mistakes

- Forgetting to relax edges V-1 times
- Not checking negative cycles in the last iteration
- Using adjacency list directly instead of edge list → slows implementation
- Forgetting to handle Infinity initialization

Step 7: Practice Plan for Bellman-Ford

Day 1:

- Implement basic Bellman-Ford → 3–5 problems

Day 2:

- Negative cycle detection → 3–5 problems

Day 3:

- Applications in competitive programming → 5–7 problems

Key Takeaways

- Bellman-Ford handles **negative weight graphs and detects cycles**
- Slower than Dijkstra but more versatile in specific cases
- Useful for **network optimization, arbitrage detection, and problem-solving**
- Practice improves **relaxation logic and negative cycle understanding**

Visit haas.dev for more Bellman-Ford guides, problem sets, and interview preparation resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app