# Shortest Path Algorithms – Design and Analysis Report

## 1. Introduction

Finding the shortest path between nodes in a graph is a fundamental problem in algorithm design with application in GPS navigation. In this project, we implemented three major shortest path algorithms:

- **Bellman–Ford Algorithm**
- **Dijkstra's Algorithm**
- **Johnson's Algorithm**

---

## 2. Bellman–Ford Algorithm

### 2.1 Overview

The Bellman–Ford algorithm computes the shortest paths from a single source vertex to all other vertices in a weighted graph. Unlike Dijkstra's algorithm, it can handle **negative edge weights** and can detect **negative weight cycles**.

### 2.2 Algorithm Design

- Initialize the distance to the source as 0 and all other distances as infinity.
- Relax all edges **|V| − 1 times**, where |V| is the number of vertices.
- Perform an additional relaxation step to check for negative cycles.

### 2.3 Correctness

The algorithm works because the longest possible simple path in a graph contains at most |V| − 1 edges. Repeated relaxation guarantees that the shortest distances are found if no negative cycles exist.

### 2.4 Time Complexity

- **O(V × E)**
  Where V is the number of vertices and E is the number of edges.

### 2.5 Advantages and Disadvantages

**Advantages**

- Handles negative edge weights
- Detect negative cycles

**Disadvantages**

- Slower than Dijkstra's algorithm
- Not efficient for large graphs

---

**3. Dijkstra's Algorithm**

**3.1 Overview**

Dijkstra's algorithm finds the shortest path from a single source to all other vertices in a graph with **non-negative edge weights**.

**3.2 Algorithm Design**

- Initialize distances and use a priority queue (min-heap).
- Repeatedly select the vertex with the smallest tentative distance.
- Relax all adjacent edges of the selected vertex.

**3.3 Correctness**

The greedy choice of selecting the minimum-distance vertex is valid because all edge weights are non-negative, ensuring that once a vertex is processed, its shortest path is finalized.

**3.4 Time Complexity**

- **O((V + E) log V)** using a priority queue.

**3.5 Advantages and Disadvantages**

**Advantages**

- Very efficient for large graphs
- Simple and widely used

**Disadvantages**

- Cannot handle negative edge weights
- Cannot detect negative cycles

---

## 4. Johnson's Algorithm

### 4.1 Overview

Johnson's algorithm is used to compute **all-pairs shortest paths** in sparse graphs. It combines Bellman–Ford and Dijkstra's algorithms.

### 4.2 Algorithm Design

1. Add a new vertex connected to all vertices with zero-weight edges.
2. Run Bellman–Ford to compute vertex potentials and detect negative cycles.
3. Reweight all edges to eliminate negative weights.
4. Run Dijkstra's algorithm from each vertex.
5. Convert distances back to original weights.

### 4.3 Correctness

The reweighting step preserves shortest paths while ensuring all edge weights are non-negative, allowing Dijkstra's algorithm to be safely applied.

### 4.4 Time Complexity

- **$O(V \times E + V^2 \log V)$**

### 4.5 Advantages and Disadvantages

**Advantages**

- Handles negative weights (no negative cycles)
- Efficient for sparse graphs

**Disadvantages**

- More complex to implement
- Higher overhead compared to single-source algorithms

---

## 5. Conclusion

In this project, three shortest path algorithms were designed and analyzed.

- **Bellman–Ford** is suitable for graphs with negative weights.

- **Dijkstra's algorithm** is the fastest choice for graphs with non-negative weights.
- **Johnson's algorithm** efficiently solves the all-pairs shortest path problem in sparse graphs with possible negative edges.

---