Negative Edges, No Problem: A Deep Dive into Bellman-Ford

Due Date: Friday 5th July 2025

Assignment Overview:

This assignment challenges you to explore and simulate the Bellman-Ford algorithm, a classic solution for finding the shortest path in a weighted graph, including those with negative edge weights. You will analyze how the algorithm detects negative-weight cycles and compare it with Dijkstra's algorithm when applicable.

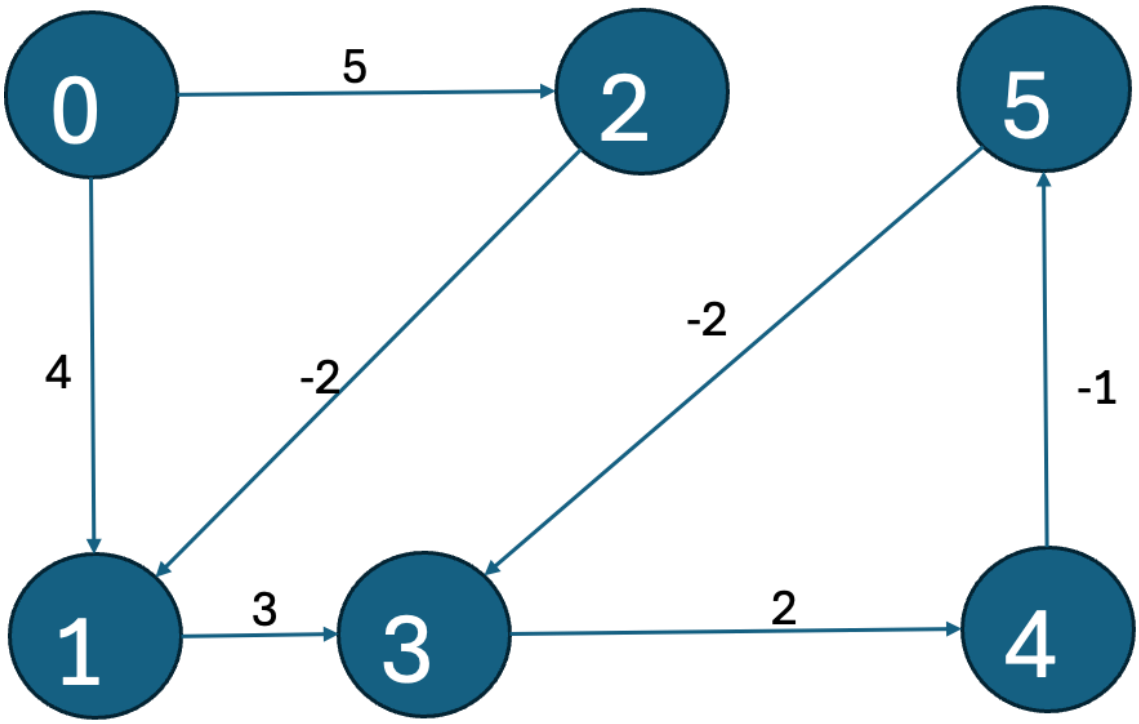
Objectives:

By completing this assignment, you will:

1. Simulate the Bellman-Ford shortest path algorithm for a given weighted directed graph.
2. Implement negative edge weight handling and cycle detection features.
3. Analyze and compare time complexity against Dijkstra's algorithm for the same graph (if applicable).
4. Provide technical documentation and reflection aligned with algorithm design principles.

Graph Specification:

You are provided the following directed weighted graph:



Deliverables:

1. Source Code (Python preferred, but other languages accepted) that:

- Implements the Bellman-Ford algorithm.
- Detects negative-weight cycles and detects Edge Causing Cycle.
- Compares with Dijkstra's algorithm.

2. Report (PDF) containing:

- Algorithm explanation (with pseudocode or flowchart).
- Code explanation (logic and structure).
- Test results (graph output, cycle detection, etc.).
- Comparison with Dijkstra (limitations in graphs with negative weights).
- Conclusion and reflection on runtime, complexity, and suitability.

3. Video Presentation (3–5 minutes):

- Demonstrate your code in action.
- Explain the Bellman-Ford process and findings in your own words.

Assignment Report Template – Bellman-Ford Algorithm

1. Introduction

- **Brief overview of shortest path problems.**

- **Why Bellman-Ford is important (especially with negative edges).**
- **Goal of this report.**

2. Algorithm Explanation

- **Description of how Bellman-Ford works (step-by-step).**
- **How it handles negative edge weights.**
- **How cycle detection is implemented.**
- **Pseudocode or flowchart (optional but recommended).**

3. Source Code Overview

- **Language and libraries used.**
- **Explanation of key modules/functions.**
- **How input is handled (e.g., adjacency list, matrix).**
- **Edge case handling and validation (e.g., cycle detection).**

4. Test Results

- **Display the sample graph and weights.**
- **Show the output: shortest path distances from source.**
- **Indicate detection of negative-weight cycle (if found).**
- **Include screenshots/console logs.**

5. Comparison with Dijkstra

- **Can Dijkstra be applied to this graph? Why or why not?**
- **If implemented, show how it fails or skips negative weights.**
- **Time complexity of both.**

6. Conclusion

- **Reflection on algorithm performance.**

- **Which situations suit Bellman-Ford best?**
- **What did you learn from this project?**

Marking Rubrics:

Criteria	Excellent (20–25%)	Good (15–19%)	Satisfactory (10–14%)	Marginal (0–9%)	Marks
Understanding of Algorithm (10%)	Thorough grasp of Bellman-Ford and negative-weight handling	Good understanding, minor gaps	Basic understanding	Poor grasp	/10
Implementation (10%)	Fully functional, clean, and efficient	Functional with minor bugs	Basic implementation	Major bugs or non-functional	/10
Report (20%)	Clear explanation, insightful analysis	Adequate explanation, some insight	Basic explanation only	Weak or unclear	/20
Complexity Analysis (20%)	Well-justified and accurate	Mostly accurate	Partial or unclear	Missing	/20
Presentation (40%)	Engaging and clear	Understandable	Minimal or dry	Poor or missing	/40
				Total Marks	**/100**