# California State University, Fresno

# DEPARTMENT OF COMPUTER SCIENCE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class: | **Algorithms & Data Structures** | | | Semester: | **Fall 2021** |
|  | | | | | |
| Points |  | Document author: | **Insert your name** | | |
|  | Author’s email: | **zekebranham@mail.fresnostate.edu** | | |
| Laboratory number: | **Laboratory 2** | | |
|  | | | | | |

**1. Statement of Objectives**

In this lab, we were to implement the Bellman-Ford algorithm that finds the single shortest path for a weighted graph. This implementation successful achieves this. In this report, I will be discussing the experimental procedure, an analysis of results, encountered problems, and a conclusion.

**2. Experimental Procedure**

The goal of this lab is to implement the Bellman-Ford algorithm, which is used for finding the shortest paths from a single source vertex to all other vertices in a weighted graph, even if the graph contains negative weight edges. I created a main function that accepts the number of vertices, number of edges, source vertex and array of edges as inputs.

The implementation consists of two main parts: the main function and the bellmanFord function. The main function serves as the entry point for the program, handling user inputs for the number of vertices and edges, the source vertex, and the edge list. Each edge is defined by its start and end vertices, along with the weight, and is stored in a 2D vector.

The bellmanFord function is the core of the program, where the algorithm is implemented. It initializes distances to all vertices as infinite except the source vertex, which is set to zero. Then, it iteratively relaxes all the edges by updating the distance to each vertex if a shorter path is found via another vertex. This relaxation process is repeated for V-1 times, where V is the number of vertices. The function also checks for negative weight cycles, which can lead to infinitely decreasing path lengths. Finally, it outputs the shortest distance from the source vertex to each vertex in the graph.

**3. Analysis**

My results mirror the suggested output given in the lab guidelines. The Bellman-Ford algorithm has a time complexity of O(VE), where V is the number of vertices and E is the number of edges in the graph. This complexity arises because the algorithm needs to iteratively relax each edge in the graph V-1 times. In each iteration, it goes through all the edges and updates the shortest distance to each vertex if a shorter path is found. This repeated process of iterating over all edges V-1 times is what leads to the O(VE) complexity.

**4. Encountered Problems**

There were not any significant issues regarding the implementation. The only struggle was trying to find a way to optimize the runtime of the program. Improving upon this time complexity is challenging with the Bellman-Ford algorithm due to its fundamental design. The algorithm has to iterate over all edges multiple times to ensure that the shortest path to each vertex is correctly calculated, especially in the presence of negative weight cycles. This iterative nature is essential to accommodate negative weights and to detect negative cycles. Optimizations can be made in specific scenarios, but the general worst-case complexity remains O(VE) due to the need to repeatedly process all edges to account for the possibility of shorter paths emerging in each iteration.

**5. Conclusions**

Overall, the implementation of the Bellman-Ford algorithm was successful with few issues.

Through this implementation, the program efficiently calculates the shortest paths in a graph, even when negative weights are involved, providing a versatile solution for various graph-related problems.

**6. References**

<https://www.simplilearn.com/tutorials/data-structure-tutorial/bellman-ford-algorithm>