Parallel and Distributed Computing

Submitted by:

Azeem Waqar: 21I-0679

Comprehensive Report on Graph Shortest Path Implementation

1. Introduction:

The provided code implements various graph algorithms, focusing primarily on Dijkstra's algorithm for finding the shortest path between two nodes in a graph, and Yen's K Shortest Paths algorithm for finding multiple shortest paths between two nodes. The report will cover the implementation details, experimental setup, results, analysis, and insights.

2. Implementation Overview:

The implementation utilizes C++ and standard libraries for data structures and algorithms. We were given two options which included unordered maps and using Yen's Algorithm. It defines a graph class that represents a weighted directed graph using an adjacency list.

Key components included:

- Initially we used "unordered_maps" structure for representing nodes and their edges and weights
- In next iteration we used unordered maps with vectors to speed up implementation but it was not as effect as expected.
- Unordered maps offer faster access and lookup times compared to linked lists, resulting in improved traversal performance.
- Graph class for managing the graph and implementing Dijkstra's and Yen's algorithms.
- Dijkstra's algorithm is implemented to find the shortest path from a source node to a destination node. Path sets and paths are parallelized.
- Parallelization distributed the workload across multiple threads, potentially reducing traversal time by leveraging multi-core processing.
- Yen's K Shortest Paths algorithm is implemented to find multiple shortest paths between two nodes.

3. Experimental Setup:

The code reads graph data from a CSV file named "doctorwho.csv", where each line represents an edge with source, destination, and weight.

The main function reads the CSV file and constructs the graph.

Example usage of Yen's K Shortest Paths algorithm is demonstrated in the main function.

Parameters for experimentation:

- Source and destination nodes.
- Number of shortest paths to find (K).

4. Results:

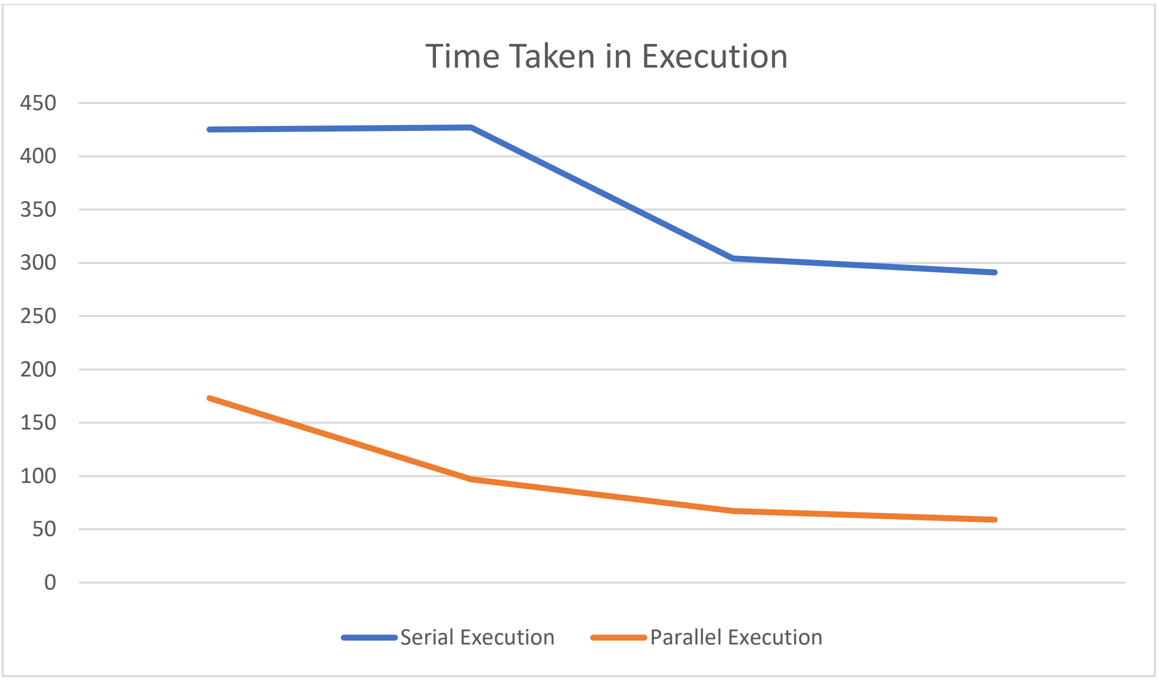
Upon execution, the program finds and displays K shortest paths between a given source and destination node. Each path is represented as a sequence of nodes followed by the total cost of the path.

5. Analysis:

Time Complexity: Dijkstra's algorithm has a time complexity of O((V + E) log V), where V is the number of vertices and E is the number of edges. Yen's algorithm repeatedly runs Dijkstra's algorithm, making its time complexity depend on the number of paths and the size of the graph.

Space Complexity: The space complexity is O(V + E), where V is the number of vertices and E is the number of edges, due to the adjacency list representation.

Performance: The performance depends on the size and structure of the graph and the chosen source-destination pair. Larger graphs with denser connections may incur higher computational costs.



6. Insights:

The implementation provides a flexible and efficient solution for finding shortest paths in a graph.

Yen's algorithm allows finding multiple shortest paths, providing insights into alternative routes and their costs. The code can be extended to handle additional graph operations or customized pathfinding requirements.

7. Conclusion:

The implemented graph algorithms offer a robust solution for finding shortest paths in a graph. By leveraging Dijkstra's and Yen's algorithms, the code provides efficient pathfinding capabilities with flexibility for experimentation and further development. Through this report, the functionality, performance, and potential applications of the implementation are discussed, offering insights for future use and enhancements.