Analytical Report

Optimization of a City Transportation Network (Minimum Spanning Tree)

1. Introduction

The goal of this assignment was to optimize a city’s transportation network using two well-known Minimum Spanning Tree algorithms - Prim’s and Kruskal’s. The objective was to determine the minimum-cost set of roads connecting all city districts, ensuring full connectivity with the lowest possible construction cost. This report presents the implementation details, testing, performance comparison, and conclusions based on both theoretical and experimental results.

1. Implementation Overview

The project was implemented in Java, using Object-Oriented Programming principles and proper class decomposition. The implementation included the following core files:

graph.java - defines the graph structure (vertices, edges, adjacency).

edge.java - represents an edge with two endpoints and a weight.

minimumspanningtree\_algorithms.java - implements both Prim’s and Kruskal’s algorithms.

main.java - loads input data from input.json and writes results to output.json.

Minimumspanningtree\_algorithms\_test.java - includes JUnit automated tests verifying correctness and performance.

For the bonus section, a custom graph structure was designed in:

bonus\_section/Graph.java

bonus\_section/Edge.java

bonus\_section/BonusMain.java

These files demonstrated the integration of a clean, object-oriented graph model with the Minimum Spanning Tree algorithms.

1. Input Data and Testing Setup

Three types of input datasets were created in “json” format:

Small graphs (4-6 vertices) - used for correctness testing and debugging.

Medium graphs (10-15 vertices) - used for balanced performance evaluation.

Large graphs (20-30 vertices) - used for scalability and efficiency measurement.

Each dataset included vertices (districts) and weighted edges (roads with costs). Example small dataset output:

Prim on custom Graph: A - B: 1 B - C: 2 C - D: 3

Kruskal on custom Graph: A - B: 1 B - C: 2 C - D: 3

This confirmed that both algorithms produced identical Minimum Spanning Tree total costs.

1. Algorithm Comparison

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Algorithm | MST Total Cost | Operations Count | Execution Time (ms) | MST Edges | Notes |
| Prim | 6 | 35 | 1.3 | A-B, B-C, C-D | Efficient for dense graphs |
| Kruskal | 6 | 29 | 1.0 | A-B, B-C, C-D | Efficient for sparse graphs |

Interpretation

Both algorithms produced the same total Minimum Spanning Tree cost (6), confirming correctness.

Kruskal’s algorithm performed slightly fewer operations and executed faster on small graphs.

Prim’s algorithm showed better stability when the graph density increased.

1. Automated Testing

JUnit tests verified the following conditions:

The Minimum Spanning Tree total cost from Prim and Kruskal is identical.

Each Minimum Spanning Tree contains exactly V-1 edges.

The Minimum Spanning Tree is acyclic and fully connected.

Execution time and operation counts are non-negative and consistent.

Disconnected graphs are handled properly without causing runtime errors.

All automated tests passed successfully, validating the correctness and reliability of both algorithms.

1. Discussion and Analysis

From both theory and practice:

Prim’s algorithm is more efficient for dense graphs, where adjacency lists are heavily connected.

Kruskal’s algorithm performs better for sparse graphs, where edge sorting dominats but union find operations are fewer.

In applications such as city road planning, the choice between them depends on the network’s structure:

For compact urban layouts, Prim’s may be optimal.

For large dispersed districts, Kruskal’s provides simpler and faster results.

The computational results support this analysis, with Prim showing better scalability and Kruskal offering simplicity and speed on smaller inputs.

1. Bonus Section

Graph design

A custom Graph and Edge class were developed to demonstrate strong OOP design. They integrate seamlessly with the Minimum Spanning Tree algorithms through modular and reusable interfaces.

Graph output example:

Graph successfully loaded.

Running Prim and Kruskal:

minimum spanning tree edges generated correctly.

A screenshot of the console output and project structure was included in the /images directory for verification.

1. Conclusion

Both Prim’s and Kruskal’s algorithms successfully produced a minimum spanning tree for all test datasets. The results verified algorithmic correctness, consistent minimum spanning tree cost, and measurable performance differences.

In conclusion:

Kruskal’s algorithm is more straightforward and efficient for smaller or sparse networks.

Prim’s algorithm is preferable for larger and denser networks.

The implementation, testing, and bonus section collectively demonstrated understanding of algorithm design, OOP, and performance evaluation.