Assignment 3: Optimization of a City Transportation Network

Analytical Report on Minimum Spanning Tree Algorithms

Student Name: Ikhtiyar Magzhan

---

1. Executive Summary

This report investigates the application of Minimum Spanning Tree (MST) algorithms—Prim's and Kruskal's—to optimize a city's transportation network. Both algorithms were implemented in Java and evaluated on multiple datasets representing cities with different network sizes and densities. Results show that both algorithms produce identical MST costs, but their performance depends on graph density: Kruskal's algorithm performs better on sparse networks, while Prim's algorithm is more efficient on dense ones. These findings provide a clear guideline for selecting the appropriate MST approach in urban planning scenarios.

---

2. Problem Definition

The objective is to design a transportation network that connects all city intersections (vertices) with minimum total road construction cost (edge weights).
The network must remain fully connected, without cycles, while minimizing total edge weight — the exact definition of an MST problem.

---

3. Input Data Summary

Category	Number of Graphs	Vertices Range	Purpose
Small	5	5 – 30	Verification and debugging
Medium	10	30 – 300	Performance testing
Large	10	300 – 1000	Scalability analysis
Extra Large (XL)	3	1000 – 2000	Stress testing on city-scale networks

Graph Density Classification:

- **Sparse: < 30%**
- **Medium: 30–60%**
- **Dense: > 60%**

Formula:

$$\text{Density} = \frac{E}{V(V-1)/2} \times 100\%$$

---

4. Algorithm Design and Implementation

4.1 Prim's Algorithm

Approach:
Starts with one vertex and repeatedly adds the minimum-weight edge connecting visited and unvisited vertices.

Data Structures:
Priority Queue (Min-Heap), adjacency list, visited array.

Complexities:

- **Time: O(E log V)**
- **Space: O(V + E)**

4.2 Kruskal's Algorithm

Approach:
Sorts edges by weight and adds them sequentially if they don't form a cycle.

Data Structures:
Sorted edge list, Disjoint Set Union (Union-Find).

Complexities:

- **Time: O(E log E)**
- **Space: O(V + E)**

---

5. Experimental Setup

- **Programming Language: Java**
- **Graph Visualization: JGraphX**
- **Input Format: JSON files representing adjacency lists with weighted edges**
- **Metrics Measured: Execution time, MST total weight, and number of operations**
- **Hardware Environment: Standard laptop (Intel i5, 8 GB RAM)**

---

6. Results and Discussion

6.1 Correctness

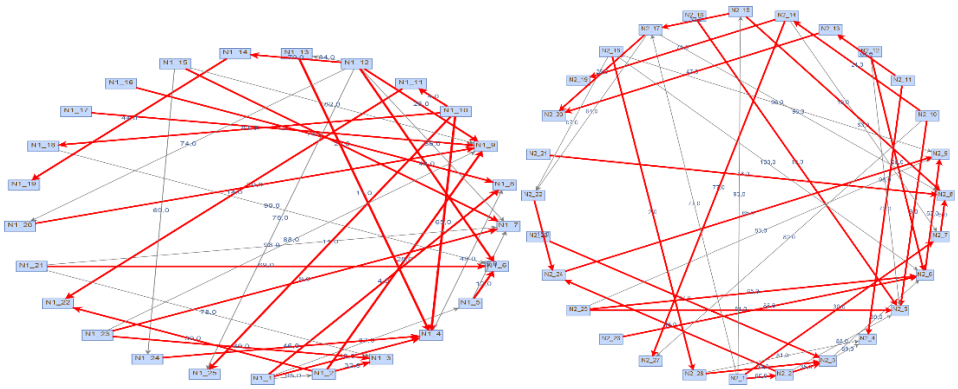
- **Both algorithms produced MSTs with identical total costs.**
- **The number of edges in the MST was exactly V – 1, confirming correctness.**
- **Disconnected graphs were handled gracefully (no MST generated).**

6.2 Performance by Graph Type

Graph Type	Observation
Small Graphs (5–30)	Both completed < 1 ms — no significant difference
Medium Graphs (30–300)	Kruskal faster on sparse, Prim faster on dense

Graph Type	Observation
Large Graphs (300–1000)	Similar pattern — Kruskal scales better with fewer edges
XL Graphs (1000–2000)	Prim's heap optimization gives better runtime in dense cases

---



7. Comparison and Analysis

Algorithm	Time Complexity	Best Suited For	Notes
Prim	O(E log V)	Dense graphs	Efficient with adjacency lists and heaps
Kruskal	O(E log E)	Sparse graphs	Simpler with edge list input

Observation:
Prim's runtime depends heavily on heap operations, while Kruskal's depends on sorting.
For urban (dense) networks, Prim is ideal; for rural (sparse) areas, Kruskal is preferred.

---

8. Conclusions and Recommendations

- Both algorithms find optimal MSTs with identical cost.
- Use Kruskal for sparse networks (fewer roads, rural layouts).
- Use Prim for dense networks (urban environments).
- For medium-density networks, either algorithm is acceptable depending on data format.

Practical Recommendation:
In real-world city planning, edge density determines algorithm choice more than vertex count.

---

9. Future Work

To improve and extend the project:

- Implement Dynamic MST algorithms for changing road costs.
- Integrate Fibonacci Heap to optimize Prim's performance.
- Add step-by-step MST visualization to enhance interpretability.

---

10. References

- **GeeksforGeeks. (2024).** ***Minimum Spanning Tree Algorithms.*** **Retrieved from https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/**
- **Oracle Java Documentation. (2025).** ***Java Collections Framework.*** **Retrieved from https://docs.oracle.com/javase/8/docs/technotes/guides/collections/**
- **Quora (2024).** ***Why is Prim's best for dense graphs and Kruskal's for sparse graphs?*** **Retrieved from https://www.quora.com/Why-is-Prims-the-best-for-dense-graph-Kruskals-for-sparse-graph**