Analytical Report

Assignment 3: Optimization of a City Transportation Network (Minimum Spanning Tree)

Student: *Daniyal Shaikenov*
Group: *SE-2426*
Subject: *Design and Analysis of Algorithms*

1. Objective

The objective of this work is to design and implement two classical Minimum Spanning Tree (MST) algorithms — **Prim's** and **Kruskal's** — to optimize a city's transportation network.

The problem aims to find the minimum total construction cost required to connect all districts of a city, represented as a **weighted undirected graph**.

2. Description of the Problem

In the transportation planning scenario, each **district** is represented as a **vertex**, and each **possible road** is represented as an **edge** with a certain **construction cost** (weight).
The goal is to select a subset of roads (edges) that connects all districts with the **minimum total cost** and **no cycles**, ensuring the network remains **fully connected**.

This corresponds to the **Minimum Spanning Tree (MST)** problem in graph theory.

3. Implementation Details

3.1. Algorithms Implemented

Two MST algorithms were implemented and compared:

- **Prim's Algorithm** — a greedy approach that grows the spanning tree one vertex at a time, always choosing the smallest-weight edge connecting the current tree to a new vertex.
- **Kruskal's Algorithm** — a greedy algorithm that sorts all edges by weight and adds them one by one, skipping edges that would form cycles (using a Disjoint Set / Union-Find structure).

Both algorithms were implemented in **Java** and designed to:

- Measure **execution time (ms)**;
- Count **operations** (comparisons, unions, finds, etc.);
- Record **MST edges** and **total cost**;
- Export results to `output.json`.

3.2. Input Data

All input data was stored in a single JSON file named **`input.json`**, following the structure:

```
{
  "graphs": [
```

```
    {
      "id": 1,
      "nodes": ["A", "B", "C", "D", "E"],
      "edges": [
        {"from": "A", "to": "B", "weight": 4},
        {"from": "A", "to": "C", "weight": 3},
        {"from": "B", "to": "C", "weight": 2}
      ]
    }
  ]
}
```

Datasets were categorized by size:

- **Small graphs:** 5–30 vertices (5 graphs)
- **Medium graphs:** 30–300 vertices (10 graphs)
- **Large graphs:** 300–1000 vertices (10 graphs)
- **Extra large graphs:** 1000–2000 vertices (3 graphs)

3.3. Output Data

Each algorithm produces results in **`output.json`** with:

- Total MST cost
- List of edges in the MST
- Number of vertices and edges
- Number of operations
- Execution time (ms)

4. Experimental Results

Graph Size	Vertices	Edges	Algorithm	MST Cost	Operations	Execution Time (ms)
Small (G1)	5	7	Prim	16	42	1.52
Small (G1)	5	7	Kruskal	16	37	1.28

Medium (G7)	50	120	Prim	482	2,143	4.61
Medium (G7)	50	120	Kruskal	482	1,915	3.93
Large (G15)	800	2200	Prim	7,930	112,356	43.12
Large (G15)	800	2200	Kruskal	7,930	105,211	36.84
Extra Large (G28)	1900	4200	Prim	18,430	241,559	114.75
Extra Large (G28)	1900	4200	Kruskal	18,430	224,138	98.41

Values are approximate, measured on test runs using randomly generated graphs.

5. Comparison and Analysis

Criterion	Prim's Algorithm	Kruskal's Algorithm
Approach	Expands tree from one vertex using priority queue	Sorts all edges globally
Time Complexity	O(E log V) (using min-heap)	O(E log E) (using sorting + Union-Find)
Space Complexity	O(V + E)	O(V + E)
Best suited for	Dense graphs	Sparse graphs
Implementation Complexity	Moderate (heap operations)	Simple and modular
Performance on large inputs	Slower due to heap updates	Faster due to linear union-find performance
Practical Observation	Good for adjacency matrix graphs	Good for edge list representations

Summary:

Both algorithms produce identical MST total costs, confirming correctness.
However:

- **Kruskal's algorithm** performed slightly faster in sparse and large networks (due to fewer heap operations).
- **Prim's algorithm** showed better stability and simpler debugging for smaller or dense graphs.

  In practice, Kruskal's algorithm achieved around **10–15% lower execution time** for large datasets.

6. Conclusions

1. Both Prim's and Kruskal's algorithms correctly compute the Minimum Spanning Tree for all datasets.
2. **Kruskal's algorithm** is generally more efficient for **sparse or large graphs**, while **Prim's algorithm** is preferable for **dense graphs**.
3. The total MST cost remains identical across both algorithms, verifying correctness.
4. Performance differences arise mainly from the underlying data structures:
    a. Kruskal benefits from efficient **Union-Find** operations.
    b. Prim depends heavily on **priority queue** updates.
5. Overall, Kruskal's algorithm demonstrates **better scalability** and **lower runtime** for city-scale networks.