Assignment 3: Optimization of a City Transportation Network (MST)

Student: Zhanassyl Sherkenov

1. Objective

The goal of this assignment is to optimize a city transportation network using **Minimum Spanning Tree (MST)** algorithms — **Prim's** and **Kruskal's** — and to compare their performance in terms of total cost, number of operations, and execution time.

The transportation network is represented as a **weighted undirected graph**, where:

- vertices represent city districts;
- edges represent possible roads;
- edge weights represent road construction costs.

2. Algorithms Overview

Prim's Algorithm

Prim's algorithm grows the MST starting from one vertex and repeatedly adds the cheapest edge connecting the tree to a new vertex.

Complexity:

- Using a priority queue: **O(E log V)**
- Efficient for **dense graphs**

Kruskal's Algorithm

Kruskal's algorithm sorts all edges by weight and adds them to the MST in order, skipping edges that would form a cycle (using Union-Find).

Complexity:

- **O(E log E)** or **O(E log V)**
- Efficient for **sparse graphs**

3. Implementation Summary

Both algorithms were implemented in **Java** with the following classes:

Package	File	Purpose
model	Graph.java, Edge.java	Custom graph data structure
algorithms	Prim.java, Kruskal.java	MST algorithms
util	JsonIO.java	JSON input/output
main	Main.java	Driver program

Graphs are loaded from input.json and results are written to output.json.

4. Input Datasets

Several graph datasets were used to evaluate correctness and performance.

Dataset	Vertices (V)	Edges (E)	Description
Small	5	7	Simple test for debugging
Medium	10	20	Moderate graph
Large	25	70	Performance test

Example JSON for a small graph:

```
{
  "vertices": 5,
  "edges": [
    {"src": 0, "dest": 1, "weight": 2},
    {"src": 0, "dest": 3, "weight": 6},
    {"src": 1, "dest": 2, "weight": 3},
    {"src": 1, "dest": 3, "weight": 8},
    {"src": 1, "dest": 4, "weight": 5},
    {"src": 2, "dest": 4, "weight": 7},
    {"src": 3, "dest": 4, "weight": 9}
  ]
}
```

5. Results

5.1 Example Output (Small Graph)

Metric	Prim	Kruskal
Total MST Cost	16.0	16.0
Operations Count	9	11
Execution Time (ms)	1	0
MST Edges	4	4

Both algorithms produced the same MST total cost, proving correctness.

5.2 Medium Graph (10 vertices, 20 edges)

Metric	Prim	Kruskal
Total MST Cost	42.0	42.0
Operations Count	47	62
Execution Time (ms)	2	1

Observation: Prim's algorithm used fewer operations due to better priority queue performance on a moderately dense graph.

5.3 Large Graph (25 vertices, 70 edges)

Metric	Prim	Kruskal
Total MST Cost	118.0	118.0
Operations Count	176	194
Execution Time (ms)	4	3

Observation: Kruskal's algorithm was slightly faster due to efficient sorting and union-find operations on larger sparse graphs.

6. Analysis and Discussion

Criterion	Prim's Algorithm	Kruskal's Algorithm
Implementation complexity	Moderate	Easy

Criterion	Prim’s Algorithm	Kruskal’s Algorithm
Efficiency on dense graphs	Better	Slightly worse
Efficiency on sparse graphs	Slightly worse	Better
Data structure dependency	Priority Queue	Union-Find
MST Cost correctness	Same as Kruskal	Same as Prim

Both algorithms produce the same total MST cost, but their performance depends on graph structure:

- Prim’s performs better when many edges exist between vertices (dense graphs);
- Kruskal’s performs better when the graph is sparse.

7. Conclusions

1. Both Prim’s and Kruskal’s algorithms correctly compute the Minimum Spanning Tree (MST).
2. Execution times and operation counts vary depending on the graph density.
3. **Prim’s algorithm** is more efficient for dense graphs due to its use of a priority queue.
4. **Kruskal’s algorithm** is simpler and faster for sparse graphs because it mainly relies on sorting.
5. The total cost of MST is always identical, confirming correctness.