**Assignment 3 — Optimization of a City Transportation Network (MST)**

Student: Abylay Dosymbek · Group: SE-2435

Date: October 27, 2025

# Objective

Apply Prim’s and Kruskal’s algorithms to construct a minimum-cost road network connecting all city districts, measure performance, and compare both approaches.

# Input Data

Graphs are loaded from data/ass\_3\_input.json. Each graph defines districts (vertices) and candidate roads (weighted edges).

# Results Summary

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Graph | V | E | Prim cost | Kruskal cost | Prim ops | Kruskal ops | Prim ms | Kruskal ms |
| 1 | 5 | 7 | 16 | 16 | 42 | 37 | 1.52 | 1.28 |
| 2 | 4 | 5 | 6 | 6 | 29 | 31 | 0.87 | 0.92 |

# Comparison: Prim vs Kruskal

• Correctness: For each connected graph, both algorithms produced identical MST total cost (as required).

• Time & Ops: On sparse graphs with adjacency lists, Prim with a binary heap is typically O(E log V); Kruskal is O(E log E) due to sorting plus near-constant DSU operations. In practice, when E is close to V, both are fast; when E grows, Kruskal’s initial sort dominates, while Prim scales with the number of PQ operations.

• Implementation complexity: Kruskal is straightforward if you already have a clean DSU. Prim requires a priority queue and an adjacency representation but integrates well with adjacency lists.

# Conclusions

• Prefer Prim on dense graphs with an adjacency matrix or when you repeatedly grow from a seed node. With a binary heap and adjacency lists, Prim is also strong on sparse graphs.  
• Prefer Kruskal when the graph is naturally represented as an explicit edge list or when the graph may be disconnected and you need a forest (Kruskal handles this naturally). Kruskal is also convenient when edges arrive offline and can be sorted once.

# References

1) Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms, MST chapter.

2) Sedgewick & Wayne — Algorithms, MST (Prim, Kruskal).