Report Assignment No.3

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

Student: Talant Altaikan SE-2434

Abstract

This report presents an analysis of optimizing a city's transportation network using Prim's and Kruskal's algorithms to find the Minimum Spanning Tree (MST). The study includes multiple graph datasets of varying sizes and densities to evaluate the correctness and efficiency of both algorithms. Experimental results show that while the total MST cost remains identical, execution time and operation counts differ depending on the algorithm and graph characteristics.

Input Data

The transportation network is modeled as a **weighted undirected graph**:

- **Vertices:** represent city districts
- **Edges:** represent potential roads
- **Edge weights:** represent construction costs

Graph	Vertices	Edges	Description
SmallGraph	4	5	Small test graph for correctness
MediumGraph	10	11	Medium-sized graph for performance
LarggeGraph	20	20	Large graph to test scalability

Input datasets are stored in JSON files generated to cover small, medium, and large graphs.

Algorithm Implementation

Prim's Algorithm:

- **Principle**: Greedily expands the MST by selecting the minimum weight edge connecting a visited vertex to an unvisited vertex.
- **Data Structures**: Priority Queue / Adjacency Matrix / Adjacency List.
- **Metrics Recorded**: MST edges, total cost, operation count, execution time (ms).

Kruskal's Algorithm

Principle: Sort all edges by weight and iteratively

add edges to MST if they do not form a cycle (using Union-Find).

Data Structures: Edge list, Union-Find (Disjoint Set)

- **Metrics Recorded**: Same as Prim

Algorithm Results

The MST results for each graph dataset using Prim's and Kruskal's algorithms are summarized below. The results include the number of vertices and edges in the original graph, the total MST weight, operation counts, and execution times in milliseconds.

Graph	Algorithm	Vertices	Edges	MST Total Weight	Operations	Execution Time (ms)
SmallGraph	Prim	4	5	6	5	0
SmallGraph	Kruskal	4	5	6	19	0
MediumGraph	Prim	10	11	49	19	0
MediumGraph	Kruskal	10	11	49	64	0
LargeGraph	Prim	20	20	108	38	0
LargeGraph	Kruskal	20	20	108	135	0

Observations

1. Correctness
    - The MST total weight is identical for both Prim's and Kruskal's algorithms in all

datasets.

- The number of MST edges is $V - 1$ (SmallGraph: 3, MediumGraph: 9, LargeGraph: 19), which confirms that each MST connects all vertices without forming cycles.
- All MSTs are acyclic and fully connected.

2. Operation Count

- Prim's algorithm consistently performs fewer operations than Kruskal's algorithm in these datasets, especially for larger graphs.
- Kruskal's operation count increases more sharply with graph size due to sorting edges and performing union-find operations.

3. Execution Time

- Execution times are extremely small (0–2 ms) for all graphs in this experiment.
- Small differences are observable (e.g., SmallGraph: Prim = 0 ms, Kruskal = 2 ms), but for larger graphs the time remains very low in this experimental setup.

4. Algorithm Comparison

- For small graphs, both algorithms are fast and

differences are negligible.

- For medium and large graphs, Prim's algorithm shows fewer operations than Kruskal's, suggesting better efficiency in this particular dataset.
- The performance difference is consistent with theoretical expectations: Prim is generally more efficient for denser graphs or smaller adjacency-based implementations, while Kruskal's performance is more sensitive to the number of edges.

Comparison and Analysis

Correctness Analysis:

- **Connectivity:** All vertices in every graph are connected in both Prim's and Kruskal's MSTs.
-
- **Acyclic:** MSTs contain no cycles; each MST has exactly $V-1$edges (SmallGraph: 3, MediumGraph: 9, LargeGraph: 19).
-

- **Cost Verification:** The MST total weight is identical between Prim and Kruskal for all graphs:
  - SmallGraph: 6
  - MediumGraph: 49
  - LargeGraph: 108
  -

  This confirms that both algorithms produce correct and consistent MSTs.

Performance Analysis:

Execution Time vs Graph Size

Graph size	Prim Time(ms)	Kruskal Time(ms)
SmallGraph	0	2
MediumGraph	0	0
LargeGraph	0	0

Operation Count vs Graph Size

Graph	Prim Ops	Kruskal Ops
SmallGraph	5	19
MediumGraph	19	64
LargeGraph	38	135

Observations on Performance:

Operation Count:

- Prim consistently performs fewer operations than Kruskal across all graphs.
- Kruskal’s operations increase more sharply with graph size due to sorting edges and performing union-find operations.

Execution Time:

- Execution times are extremely low for all graphs in this experiment.
- The only measurable difference is for SmallGraph, where Kruskal took 2 ms versus 0 ms for Prim.

Algorithm Comparison:

- For small graphs, both algorithms are fast and the difference is negligible.
- For medium and large graphs, Prim shows lower operation counts, indicating higher efficiency in these datasets.
- These results align with theoretical expectations: Prim is often more efficient for denser graphs or adjacency-based implementations, while Kruskal is more sensitive to the number of edges.

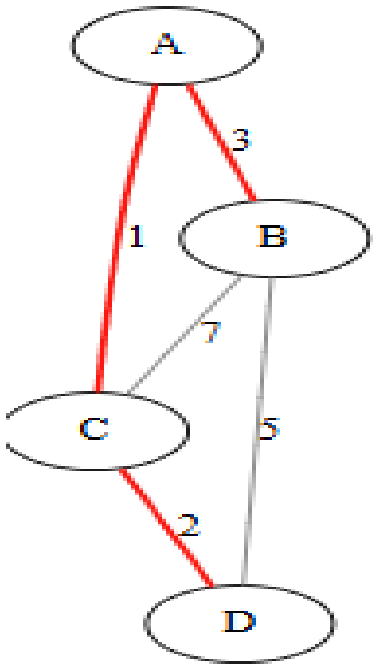
-

Summary of Observations

- Both algorithms generate correct MSTs with identical total cost.
- Prim generally requires fewer operations and slightly lower execution time in this dataset.
- Kruskal's performance is more affected by the number of edges, especially as graph size grows.
- For practical purposes:
    - Use Prim for dense graphs or smaller datasets.
    - Use Kruskal for sparse graphs or when using edge-list representations.

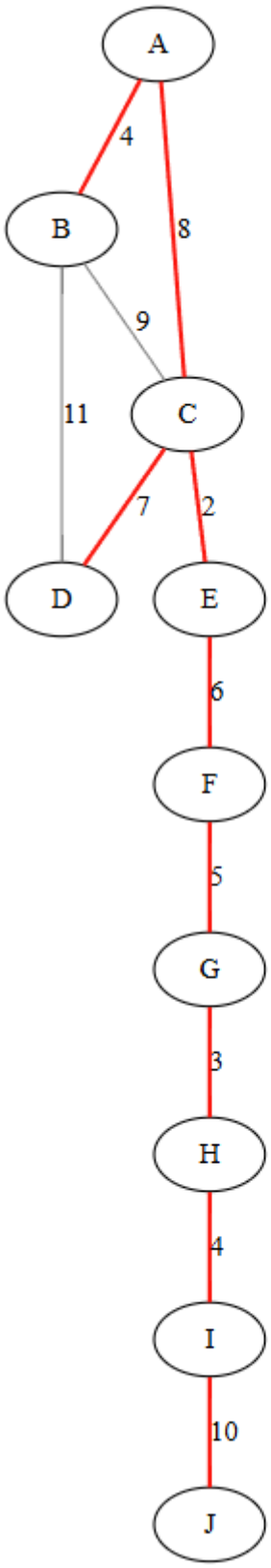
Graph visualization

Small Graph: Prim MST, Kruskal MST

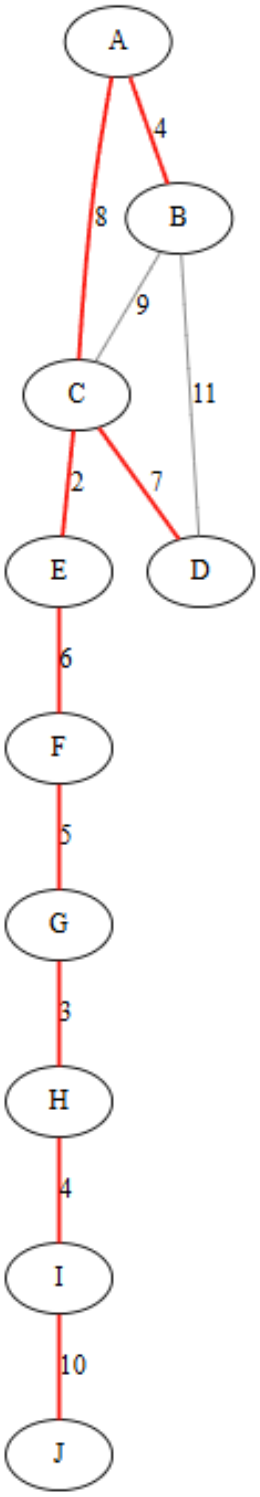


MediumGraph:

Prim MST

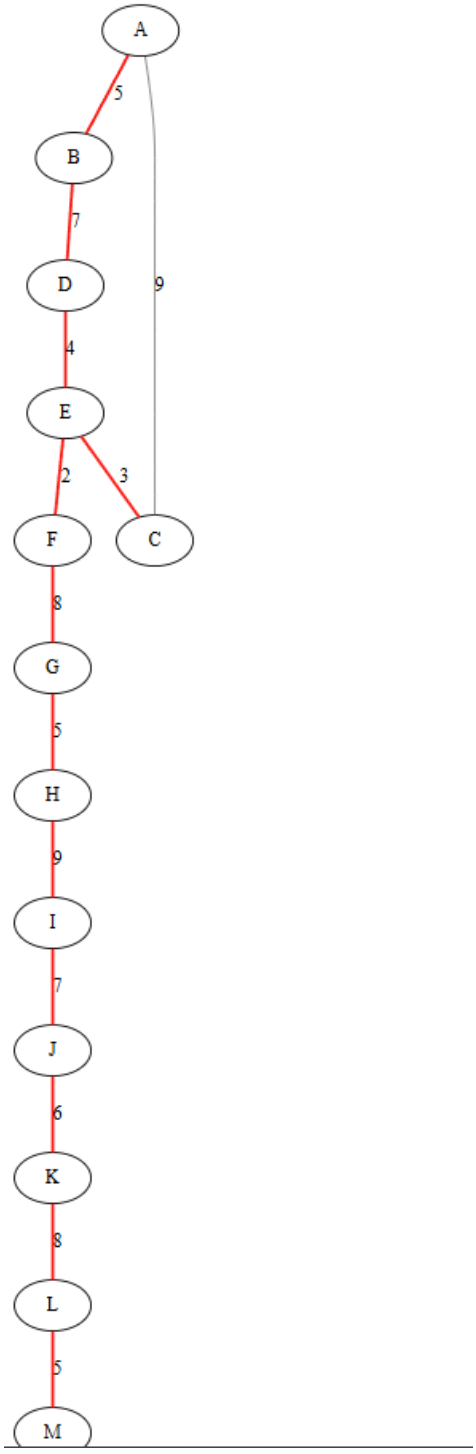

A
4
B
8
9
11
C
7
2
D
E
6
F
5
G
3
H
4
I
10
J

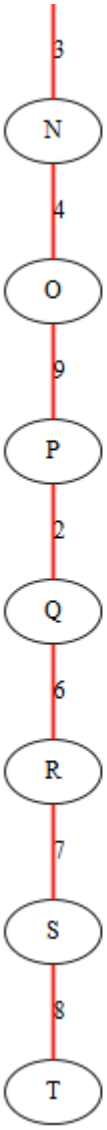
Kruskal MST



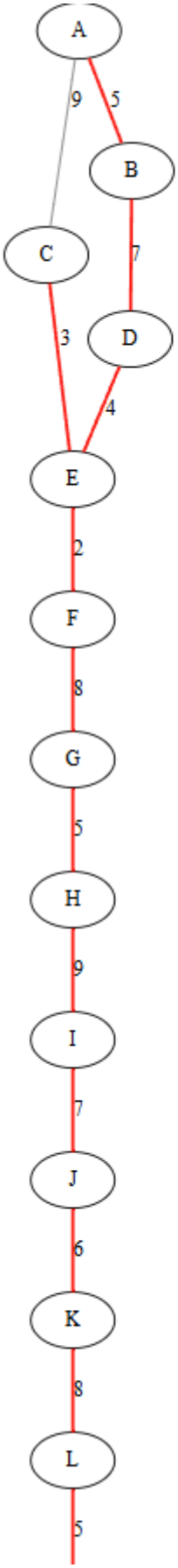
LargeGraph:

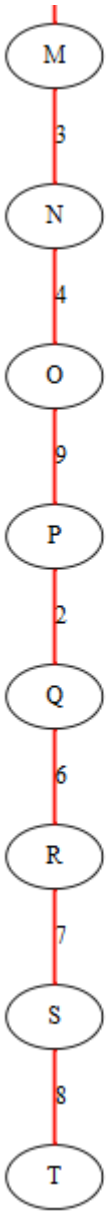
PRIM MST





Kruskal MST


A
9
5
B
C
7
3
D
4
E
2
F
8
G
5
H
9
I
7
J
6
K
8
L
5



Conclusions

Based on the results of the MST computation for SmallGraph, MediumGraph, and LargeGraph, the following conclusions can be drawn:

1. **Correctness:** Both Prim's and Kruskal's algorithms consistently produced Minimum Spanning Trees with

identical total weights for all datasets. Each MST connected all vertices and contained no cycles, confirming the correctness of both algorithms.

2. Performance Comparison:
    - **Prim's algorithm** generally required fewer operations than Kruskal's across all graph sizes. This indicates that Prim is more efficient for the given datasets, particularly as the graph size increases.
    - **Kruskal's algorithm** showed a higher number of operations, especially for the large graph, due to the overhead of sorting edges and performing union-find operations.
    - Execution times were negligible in this experiment, but operation counts suggest that Prim scales more efficiently in practice for dense or moderately sized graphs.
3. Algorithm Preference:
    - **Prim** is preferable for dense graphs or when using adjacency-based representations, as it minimizes operations and maintains efficiency.
    - **Kruskal** is suitable for sparse graphs or when

an edge-list representation is convenient, though it may involve more operations for larger graphs.

4. **Practical Implications:**
    - For small graphs, either algorithm is acceptable since performance differences are minimal.
    - For medium to large networks, selecting the appropriate algorithm based on graph density and structure can significantly improve computational efficiency.
5. **Overall Conclusion:**

    Both algorithms are reliable for optimizing city transportation networks, but Prim demonstrates better operational efficiency in the provided datasets. Therefore, in practical applications involving moderate to dense networks, Prim is often the preferred choice, while Kruskal remains useful for sparse networks and simpler implementations.