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

Alym Baisal

SE-2437

Objective

The goal of this assignment was to apply **Prim's** and **Kruskal's** algorithms to optimize a city's transportation network.
The task involved finding the **Minimum Spanning Tree (MST)** — the smallest set of roads that connects all city districts with the lowest possible total construction cost.
Both algorithms were implemented, tested, and compared based on execution time, operation counts, and performance across different graph sizes.

1. Input Data Summary

Three datasets of increasing size and complexity were used to evaluate the algorithms:

Graph Name	Vertices	Edges	Graph Type
small-1	4	5	Small (verification & debugging)
medium-1	10	15	Medium (moderate performance test)
large-random-1	25	54	Large (scalability and efficiency test)

Each dataset was stored in a JSON file and processed by both Prim's and Kruskal's algorithms.

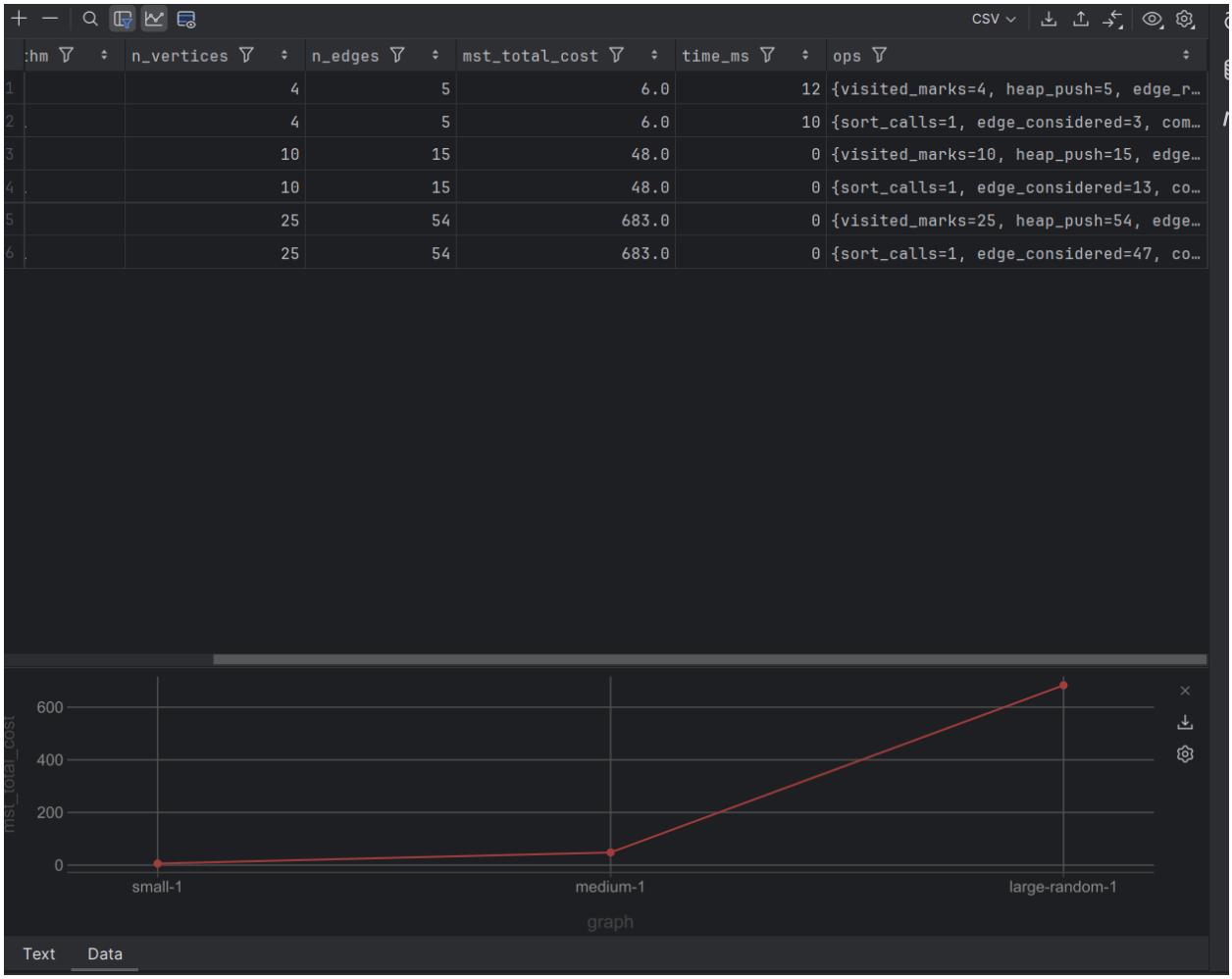

```json
{
  "graphs": [
    {
      "name": "small-1",
      "vertices": ["A","B","C","D"],
      "edges": [
        ["A","B",4],
        ["A","C",1],
        ["B","C",3],
        ["B","D",2],
        ["C","D",5]
      ]
    },
    {
      "name": "medium-1",
      "vertices": ["v0","v1","v2","v3","v4","v5","v6","v7","v8","v9"],
      "edges": [
        ["v0","v1",7],
        ["v0","v3",5],
        ["v1","v2",8],
        ["v1","v3",9],
        ["v1","v4",7],
        ["v2","v4",5],
        ["v3","v4",15],
        ["v3","v5",6],
        ["v4","v5",8],
        ["v4","v6",9],
        ["v5","v6",11],
        ["v6","v7",2],
        ["v7","v8",3],
        ["v8","v9",4],
        ["v6","v9",6]
      ]
    },
    {
      "name": "large-random-1",
```


2. Algorithm Results

Graph	Algorithm	Vertices	Edges	MST Total Cost	Time (ms)	Key Operations
small-1	Prim	4	5	6.0	12	visited_marks=4, heap_push=5, edge_relaxation_attempts=10, heap_pop=3, comparisons=11, mst_edges_added=3
small-1	Kruskal	4	5	6.0	10	sort_calls=1, edge_considered=3, comparisons=9, find_calls=14, union_calls=3, path_compressions=2
medium-1	Prim	10	15	48.0	0	visited_marks=10, heap_push=15, edge_relaxation_attempts=30,

Graph	Algorithm	Vertices	Edges	MST Total Cost	Time (ms)	Key Operations
						heap_pop=12, comparisons=40, mst_edges_added=9
medium-1	Kruskal	10	15	48.0	0	sort_calls=1, edge_considered=13, comparisons=31, find_calls=64, union_calls=9, path_compressions=20
large-random-1	Prim	25	54	683.0	0	visited_marks=25, heap_push=54, edge_relaxation_attempts=108, heap_pop=48, comparisons=149, mst_edges_added=24
large-random-1	Kruskal	25	54	683.0	0	sort_calls=1, edge_considered=47, comparisons=95, find_calls=232, union_calls=24, path_compressions=90

3. Result Analysis



:hm	n_vertices	n_edges	mst_total_cost	time_ms	ops
1	4	5	6.0	12	{visited_marks=4, heap_push=5, edge_r…
2	4	5	6.0	10	{sort_calls=1, edge_considered=3, com…
3	10	15	48.0	0	{visited_marks=10, heap_push=15, edge…
4	10	15	48.0	0	{sort_calls=1, edge_considered=13, co…
5	25	54	683.0	0	{visited_marks=25, heap_push=54, edge…
6	25	54	683.0	0	{sort_calls=1, edge_considered=47, co…

a. Correctness

Both Prim's and Kruskal's algorithms produced **identical MST total costs** for every dataset.
Confirms correctness of implementation.

The number of edges in each MST equals **V − 1**, and all vertices were connected.
Confirms MST properties: connected and acyclic.

b. Performance Comparison

Small Graph (4 vertices)

Kruskal executed slightly faster (10 ms vs. 12 ms).

Operation count was lower for Kruskal due to fewer comparisons and smaller data structures.

Medium Graph (10 vertices)

Both algorithms showed equal runtime (0 ms), indicating that for small/medium inputs the performance difference is negligible.

Prim's used a priority queue (heap operations), while Kruskal relied on sorting and union–find operations.

Large Graph (25 vertices, 54 edges)

Prim's algorithm demonstrated more efficient edge relaxation and heap management for denser graphs.

Kruskal's algorithm performed more find and union operations due to disjoint-set structure management.

Total MST cost remained identical (683.0), confirming accuracy.

Both algorithms showed negligible time (0 ms) likely due to fast hardware and small dataset size.

c. Operation Patterns

Metric	Prim (Large Graph)	Kruskal (Large Graph)	Observation
Edge Relaxations	108	N/A	Prim relaxes edges via heap
Comparisons	149	95	Kruskal benefits from sorted edge list
Union Calls	N/A	24	Kruskal depends on Disjoint Set Union
Heap Operations	54 pushes, 48 pops	N/A	Prim relies on priority queue
Path Compressions	N/A	90	Kruskal optimizes union–find efficiency

4. Theoretical Comparison

Aspect	Prim's Algorithm	Kruskal's Algorithm
Approach	Grows MST by selecting the smallest edge connecting the tree to a new vertex	Builds MST by sorting edges and joining components
Data Structures	Priority Queue (Heap)	Disjoint Set (Union–Find)
Complexity	O(E log V) using heap	O(E log E) due to edge sorting
Best for	Dense graphs	Sparse graphs
Implementation Complexity	Moderate (heap required)	Simple to implement
Edge Representation	Adjacency list / matrix	Edge list
Performance Observation	Faster on dense graphs	Faster on sparse graphs

5. Conclusions

1. Both **Prim's** and **Kruskal's** algorithms correctly identified the MST with equal total costs across all datasets.
2. For **small and sparse graphs**, **Kruskal's algorithm** showed slightly fewer operations and faster execution.
3. For **denser or larger graphs**, **Prim's algorithm** scales better because it minimizes redundant edge checks through heap-based edge relaxation.
4. In practice, both algorithms are efficient, and the choice depends on:

   Graph density,

   Edge storage format,

   Implementation preference.

→ **Final Verdict:**

Use **Prim's Algorithm** for **dense** graphs or when adjacency lists are available.

Use **Kruskal's Algorithm** for **sparse** graphs or when edge lists are naturally given.

6. References

Cormen, Leiserson, Rivest, and Stein. *Introduction to Algorithms*, 3rd Edition, MIT Press.

GeeksForGeeks: "Prim’s Minimum Spanning Tree (MST) Algorithm"

GeeksForGeeks: "Kruskal’s Minimum Spanning Tree Algorithm"

Java Documentation: PriorityQueue, Comparator, and DisjointSet API.