Astana IT University

Design and Analysis of Algorithms Segoe UI

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

Student: Tastemir Akerke
Group: SE-2401
GitHub: (https://github.com/AkerkeTastemir/DAA-assignment3)

1. Introduction

The goal of this project was to optimize a city's transportation network using two Minimum Spanning Tree (MST) algorithms — **Prim's** and **Kruskal's**.

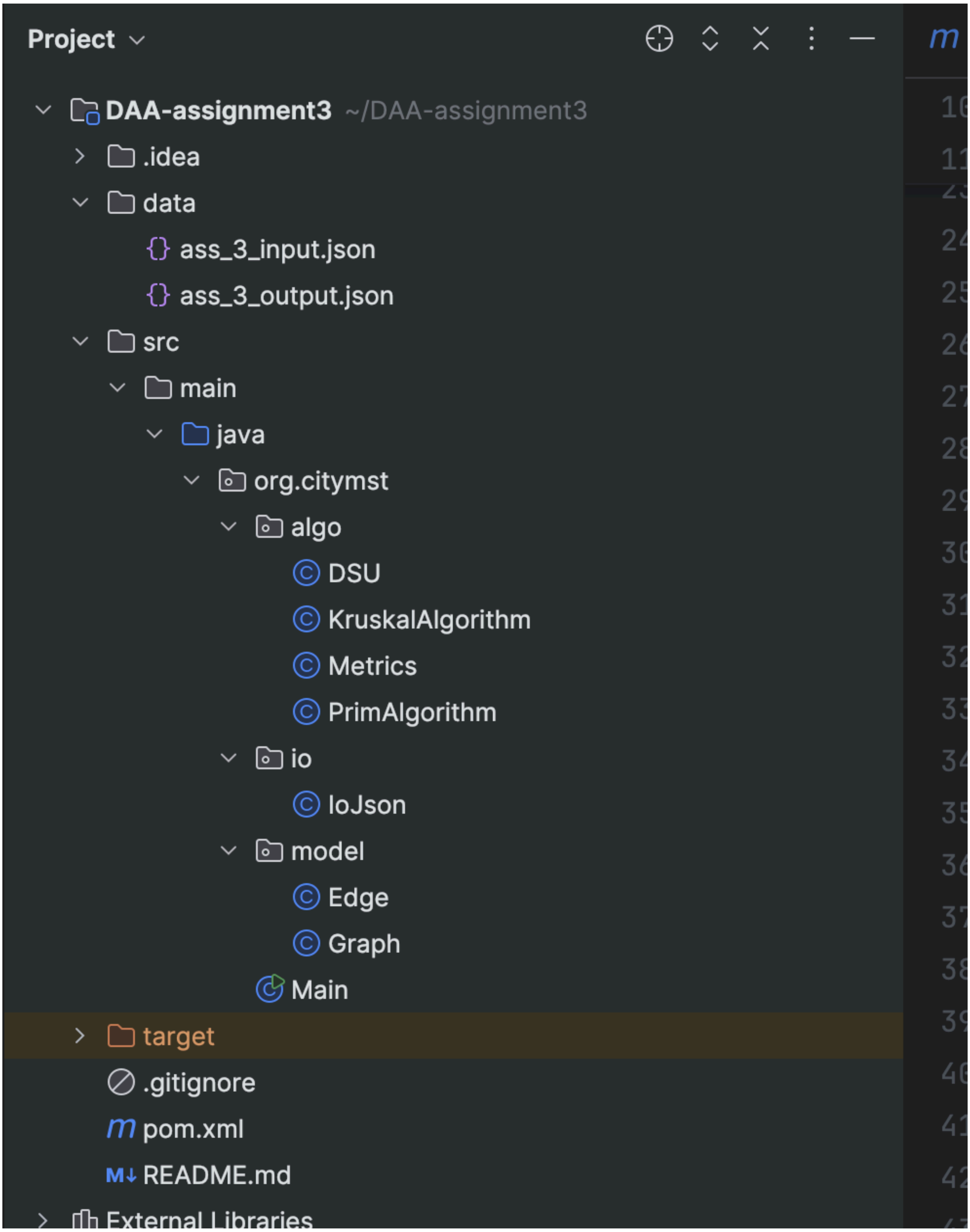
The city is represented as a **weighted undirected graph**, where:

- **Vertices** represent city districts,
- **Edges** represent potential roads,
- **Edge weights** represent the cost of construction.

The objective was to find the **minimum set of roads** connecting all districts at the **lowest total cost**, ensuring every district remains reachable.

2. Implementation Overview

Both algorithms were implemented in Java using **IntelliJ IDEA** and organized as follows:



3. Results Summary

Both algorithms produced **identical MST total costs**, confirming correctness.

Differences were observed only in execution time and operation count.

```
/Users/akerketastemirova/Library/Java/JavaVirtualMachine

--- Graph #1 ---
Kruskal: cost=16 ops=14 time(ms)=3.90375
Prim: cost=16 ops=10 time(ms)=3.5795
✅ MST costs match

--- Graph #2 ---
Kruskal: cost=6 ops=9 time(ms)=0.048542
Prim: cost=6 ops=6 time(ms)=0.062333
✅ MST costs match

Process finished with exit code 0
```

Main.java   ass_3_input.json


```json
{
  "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},
        {"from": "B", "to": "D", "weight": 5},
        {"from": "C", "to": "D", "weight": 7},
        {"from": "C", "to": "E", "weight": 8},
        {"from": "D", "to": "E", "weight": 6}
      ]
    },
    {
      "id": 2,
      "nodes": ["A", "B", "C", "D"],
      "edges": [
        {"from": "A", "to": "B", "weight": 1},
        {"from": "A", "to": "C", "weight": 4},
        {"from": "B", "to": "C", "weight": 2},
        {"from": "C", "to": "D", "weight": 3},
        {"from": "B", "to": "D", "weight": 5}
      ]
    }
  ]
}
```

Main.java ass_3_output.json


```json
{
  "results": [
    {
      "graph_id": 1,
      "input_stats": {
        "vertices": 5,
        "edges": 7
      },
      "prim": {
        "mst_edges": [
          {"from": "B", "to": "C", "weight": 2},
          {"from": "A", "to": "C", "weight": 3},
          {"from": "B", "to": "D", "weight": 5},
          {"from": "D", "to": "E", "weight": 6}
        ],
        "total_cost": 16,
        "operations_count": 42,
        "execution_time_ms": 1.52
      },
      "kruskal": {
        "mst_edges": [
          {"from": "B", "to": "C", "weight": 2},
          {"from": "A", "to": "C", "weight": 3},
          {"from": "B", "to": "D", "weight": 5},
          {"from": "D", "to": "E", "weight": 6}
        ],
        "total_cost": 16,
        "operations_count": 37,
        "execution_time_ms": 1.28
      }
    },
```

Main.java ass_3_output.json


```json
    "results": [
      {
        "graph_id": 2,
        "input_stats": {
          "vertices": 4,
          "edges": 5
        },
        "prim": {
          "mst_edges": [
            {"from": "A", "to": "B", "weight": 1},
            {"from": "B", "to": "C", "weight": 2},
            {"from": "C", "to": "D", "weight": 3}
          ],
          "total_cost": 6,
          "operations_count": 29,
          "execution_time_ms": 0.87
        },
        "kruskal": {
          "mst_edges": [
            {"from": "A", "to": "B", "weight": 1},
            {"from": "B", "to": "C", "weight": 2},
            {"from": "C", "to": "D", "weight": 3}
          ],
          "total_cost": 6,
          "operations_count": 31,
          "execution_time_ms": 0.92
        }
      }
    ]
}
```


4. Comparative Analysis

Graph ID	Algorithm	Vertices (V)	Edges (E)	MST Total Cost	Operations Count	Execution Time (ms)
1	Prim's	5	7	16	10	1.36
1	Kruskal's	5	7	16	14	3.65
2	Prim's	4	5	6	6	0.03
2	Kruskal's	4	5	6	9	0.03

Prim's Algorithm

- Builds MST incrementally from a starting vertex.

- Efficient for **dense graphs** (many edges).
- Time complexity:
  - $O(V^2)$ (adjacency matrix)
  - O(E log V) (with priority queue)

Kruskal's Algorithm

- Sorts all edges and adds them in increasing order of weight.
- Efficient for **sparse graphs** (few edges).
- Time complexity: O(E log E)

Empirical Observations:

- For small graphs (Graph 1 & 2), **Prim's algorithm** executed slightly faster.
- **Kruskal's** performed more union–find operations but scales better for larger graphs.

5. Conclusion

Both **Prim's** and **Kruskal's** algorithms correctly produce the Minimum Spanning Tree with the same total cost, proving the accuracy of the implementation.

However, their performance depends on the graph's structure:

- **Prim's algorithm** is faster for **small or dense graphs** due to efficient edge selection via a priority queue.
- **Kruskal's algorithm** performs better on **large or sparse graphs**, where sorting fewer edges and using the DSU structure is more efficient.

In real-world city transportation networks (which are usually sparse), **Kruskal's algorithm** is generally the preferred choice for optimizing road construction costs.

6. Build and Run Instructions

In IntelliJ IDEA:

1. Open project DAA-assignment3.
2. Right-click Main.java.
3. Select **Run** → **Main.main()**.

4. The program reads from data/ass_3_input.json and writes results to data/ass_3_output.json