**SE2408 - ADILDABEK NURASSYL**

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

**1. Objective**

The purpose of this assignment is to optimise a city’s transportation network by determining the minimum set of roads that connect all city districts with the lowest possible total construction cost.

Two classical algorithms for finding the ***Minimum Spanning Tree (MST)*** were implemented and analysed:

* **Prim’s Algorithm**
* **Kruskal’s Algorithm**

Both algorithms were implemented in **Java**, using data read from a JSON file, and executed on the following system:

**Test System:**

* CPU: Intel Core i7-12700F (12 cores, 20 threads)
* GPU: NVIDIA RTX 3060 8GB
* RAM: 16 GB DDR4 3200 MHz
* OS: Windows 10 x64
* JDK: OpenJDK 17
* Build System: Maven 3.9.6

**2. Input Data Summary**

The input JSON file contained two graphs representing potential road networks between city districts.  
Each vertex represents a district, and each weighted edge represents the cost of constructing a road between two districts.

| **Graph ID** | **Vertices** | **Edges** | **Description** |
| --- | --- | --- | --- |
| 1 | 5 | 7 | Medium-sized network with districts A–E |
| 2 | 4 | 5 | Smaller network with districts A–D |

**3. Results Summary**

The MSTs and performance results for both algorithms are presented below.

**Graph 1**

**Vertices:** 5  **Edges:** 7

| **Algorithm** | **MST Edges** | **Total Cost** | **Operations** | **Execution Time (ms)** |
| --- | --- | --- | --- | --- |
| **Prim’s** | A–C(3), C–B(2), B–D(5), D–E(6) | **16** | 5 | **3.184** |
| **Kruskal’s** | B–C(2), A–C(3), B–D(5), D–E(6) | **16** | 45 | **1.2582** |

**Graph 2**

**Vertices:** 4  **Edges:** 5

| **Algorithm** | **MST Edges** | **Total Cost** | **Operations** | **Execution Time (ms)** |
| --- | --- | --- | --- | --- |
| **Prim’s** | A–B(1), B–C(2), C–D(3) | **6** | 3 | **0.0272** |
| **Kruskal’s** | A–B(1), B–C(2), C–D(3) | **6** | 31 | **0.0197** |

**4. Analysis and Comparison**

**Correctness**

* Both Prim’s and Kruskal’s algorithms produced MSTs with identical **total costs** for each graph, verifying correctness.
* The structure of the MST differs slightly due to the algorithms’ nature but both yield optimal solutions.

**Performance Observations**

* For **Graph 1** (larger, denser), **Kruskal’s algorithm** executed almost **2.5× faster** (1.26 ms vs 3.18 ms).  
  This shows that sorting edges once and using an efficient Union-Find is advantageous when the number of edges is not excessive.
* For **Graph 2** (smaller and sparse), both algorithms performed extremely fast (below 0.03 ms), with **Kruskal** still marginally faster.
* **Prim’s algorithm** performed fewer logical operations (5 vs 45) but used more heap and adjacency-list operations, which increased execution time slightly.

**Efficiency and Complexity**

| **Algorithm** | **Time Complexity** | **Best for** | **Data Structures Used** |
| --- | --- | --- | --- |
| **Prim’s** | O(E log V) | Dense graphs | Priority Queue (Min-Heap), Adjacency List |
| **Kruskal’s** | O(E log E) ≈ O(E log V) | Sparse graphs | Disjoint Set (Union-Find), Sorted Edge List |

**Interpretation**

* On modern CPUs, the difference is small for small graphs.
* On larger datasets, **Prim’s** can outperform **Kruskal’s** when implemented with efficient heaps and adjacency structures, particularly for dense connectivity.
* **Kruskal’s** simplicity and predictable runtime make it ideal for sparse or moderately connected graphs.

**5. Conclusions**

1. Both algorithms correctly generated the MST with identical total construction cost.
2. **Kruskal’s algorithm** achieved lower execution times on both graphs in this test.
3. **Prim’s algorithm** performed fewer explicit operations but incurred higher runtime overhead due to priority queue management.
4. For **dense graphs**, Prim’s algorithm may scale better; for **sparse graphs**, Kruskal’s algorithm is typically more efficient.
5. On the tested system (i7-12700F, RTX 3060, 16 GB RAM), both algorithms execute nearly instantaneously for small networks, showing the efficiency of modern hardware.

**6. References**

1. GeeksforGeeks – *Prim’s Minimum Spanning Tree (MST) Algorithm.*  
   https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/
2. GeeksforGeeks – *Kruskal’s Minimum Spanning Tree Algorithm.*  
   https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/
3. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. *Introduction to Algorithms*, 3rd Edition.
4. Java Platform SE 17 Documentation – Oracle.

**7. Deliverables Summary**

| **File** | **Description** |
| --- | --- |
| Main.java | Java implementation of Prim’s and Kruskal’s algorithms |
| pom.xml | Maven configuration with Gson dependency |
| ass\_3\_input.json | Input graph data |
| ass\_3\_output.json | Program output with results |
| report.pdf | Analytical report (this document) |
| README.md | Optional GitHub summary |