Assignment 3 — Minimum Spanning Tree (MST)

Course: Design and Analysis of Algorithms
Student: Nurhan Turganbek

Project Overview

This project implements two fundamental algorithms for finding the Minimum Spanning Tree (MST) in a weighted, undirected graph: Prim’s Algorithm and Kruskal’s Algorithm. The implementation is written in Java and organized as a Maven project using IntelliJ IDEA. The system reads input graphs from a JSON file, computes MSTs using both algorithms, and outputs results to JSON and CSV files.

Implementation Details

The project includes the following components: - Edge and Graph classes to represent graph structure. - PrimAlgorithm and KruskalAlgorithm classes implementing MST logic. - MSTResult for structured output of results. - MSTMain as the main entry point handling input/output. - CSVExporter for generating CSV performance summaries. - Automated tests (JUnit 5) ensuring correctness and consistency.

Example Output (JSON)

```
{
  "results": [
    {
      "graph_id": 1,
      "input_stats": {"vertices": 5, "edges": 6},
      "prim": {"total_cost": 8, "execution_time_ms": 44.76},
      "kruskal": {"total_cost": 8, "execution_time_ms": 1.85}
    }
  ]
}
```

Example Output (CSV)

```
Graph_ID,Vertices,Edges,Kruskal_Cost,Prim_Cost,All_Costs_Match
1,5,6,8,8,YES
2,12,12,48,48,YES
3,20,19,90,90,YES
```

Automated Tests

JUnit tests verify correctness and performance by checking: - Equality of MST total costs (Prim == Kruskal) - Edge count equals V–1 - No cycles (acyclic MST) - All vertices connected - Non-negative and consistent metrics - Reproducibility of results

Conclusion

Both Prim’s and Kruskal’s algorithms produced identical MSTs for all connected graphs. Automated tests confirmed correctness, acyclicity, and consistent performance results. The CSV output further summarizes algorithmic efficiency across varying graph sizes.

Full source code and datasets are available in the GitHub repository.