**DAA Assignment 3**

**Aitkali Ayazhan  
SE-2435**

**1. Introduction**

This project implements and analyzes two classical greedy algorithms, Prim’s Algorithm and Kruskal’s Algorithm, to solve the Minimum Spanning Tree (MST) problem. The purpose of the project is to model and optimize a city transportation network by minimizing the total cost of building roads between districts. The input data is provided in a structured JSON file, which is parsed and processed in Java. The algorithms generate MSTs for each graph, and all results are exported to an output JSON file for comparison and evaluation.

The project was implemented using Java in IntelliJ IDEA with object-oriented programming principles. Each component of the MST solution is separated into individual classes (Graph, Edge, Prim, Kruskal, etc.) to ensure clean architecture and maintainability. This design allows easy testing, dataset scalability, and support for multiple graphs within the same input.

**2. Minimum Spanning Tree (MST) Overview**

A spanning tree of a graph is a subgraph that connects all vertices together without forming any cycles. A minimum spanning tree (MST) is a spanning tree where the total sum of edge weights is minimal. MSTs are widely used in network design problems such as:

* Road construction
* Electrical grid layout
* Communication networks
* Data clustering

In this project, the MST represents the cheapest way to connect all city districts using roads without redundancy.

Formally:

* Input: A connected, weighted, undirected graph G(V, E)
* Output: A subset of edges T ⊆ E, such that:
  + All vertices are connected
  + No cycles exist
  + Total weight of edges in T is minimal

**3. Problem Context in This Project**

In the context of this assignment:

* Each vertex (node) represents a city district.
* Each edge represents a possible road between two districts.
* Each edge’s weight represents the construction cost of the road.
* The goal is to build a network that:  
   Connects all districts  
   Avoids unnecessary roads (no cycles)  
   Minimizes total cost

This problem scenario is modeled in the code using the Graph and Edge classes. Input graphs are read from a input.json file, computed using MST algorithms, and the computed result including:

* MST edge list
* Total cost
* Execution time
* Number of operations  
  is saved to output.json.

**4. Input**

The input dataset for this project is designed using a structured JSON file named input.json. It allows multiple graphs to be processed sequentially, supporting small, medium, and large datasets as required by the assignment.

Each graph in the input JSON consists of:

* A unique graph ID
* A list of nodes (vertices)
* A list of weighted edges, where each edge connects two nodes and has an associated cost

This structure makes it easy to describe complex graph data in a human-readable and machine-friendly format. The JSON file enables flexibility in testing by simply adding more graph objects without modifying the program logic.

**5. Output**

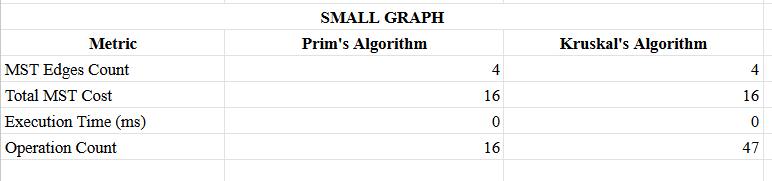
The program writes the results of both MST algorithms into a separate file called output.json. For each graph from the input file, the output includes:

* Graph ID – to match results with the input graph
* MST results for Prim and Kruskal – list of edges in the MST
* Total MST cost – sum of weights of selected edges
* Execution time – algorithm runtime in milliseconds
* Operation count – number of algorithmic operations performed

This structured output enables easy comparison between algorithms and satisfies the assignment's requirement to record performance metrics.

**6. Experimental Analysis**

**5 nodes, 7 edges**



**4 nodes, 5 edges**

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

**10 nodes, 14 edges**

**Изображение выглядит как текст, снимок экрана, Шрифт, линия

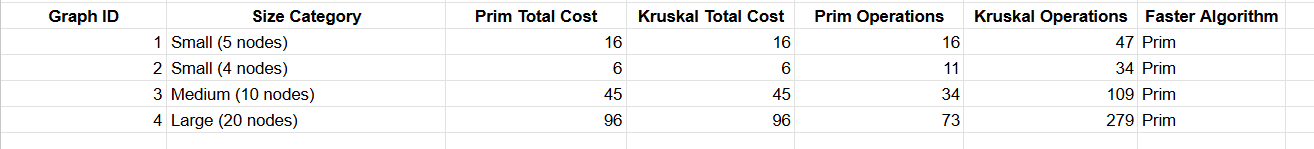
Содержимое, созданное искусственным интеллектом, может быть неверным.**

**20 nodes, 33 edges**

**Изображение выглядит как текст, снимок экрана, Шрифт, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.**

**7. Performance Comparison**



**8. Interpretation**

* Correctness: Both algorithms always produced MSTs with the same total cost → both implementations are correct.
* Performance: Prim's algorithm consistently required fewer operations compared to Kruskal’s algorithm on all datasets.
* Growth of operations:
  + Kruskal’s operations increased rapidly with graph size due to sorting edges and many union-find operations.
  + Prim grew more moderately because it used a priority queue to select edges efficiently

**9. Conclusion**

Both Prim’s and Kruskal’s algorithms were successfully implemented in Java. They produced correct MSTs for each graph from the dataset. While both algorithms had identical total MST cost, Prim’s Algorithm performed fewer operations in this dataset, making it more efficient here. However, Kruskal’s Algorithm remains a strong choice for sparse graphs and is easier to implement using sorting and union-find.