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

Objective

The purpose of this assignment is to apply **Prim's** and **Kruskal's** algorithms to optimize a city's transportation network by determining the minimum set of roads that connect all city districts with the lowest possible total construction cost. Students will also analyze the efficiency of both algorithms and compare their performance.

TASK DESCRIPTION

The city administration plans to construct roads connecting all districts in such a way that:

- each district is reachable from any other district;
- the total cost of construction is minimized.

This scenario is modeled as a **weighted undirected graph**, where:

- vertices represent city districts,
- edges represent potential roads,
- the edge weight represents the cost of constructing the road.

ASSIGNMENT REQUIREMENTS

1. Read the input data describing the transportation network from a **JSON** file and use as an example to create your own JSON file describing the transportation network data.

2. Implement both:

- Prim's algorithm
- Kruskal's algorithm

to find the Minimum Spanning Tree (**MST**).

3. For each algorithm, determine and record:

- the list of edges forming the MST;
- the total cost of the MST;
- the number of vertices and edges in the original graph;
- the number of operations performed (key algorithmic actions such as comparisons, unions, etc.);
- the execution time in milliseconds.

4. Compare the results of both algorithms. The MST total cost must be identical, though the specific structure of the tree may differ.

TESTING REQUIREMENTS

1. INPUT DATASETS

Generate multiple input datasets with different graph sizes and densities to evaluate both correctness and performance: **(use example from input.json)**

• **Small graphs** (4–6 vertices) – for verifying correctness and debugging.

• **Medium graphs** (10–15 vertices) – for observing performance on moderately sized networks.

• **Large graphs** (20–30+ vertices) – for testing scalability and efficiency differences.

• Each dataset should be stored in a JSON file (for example, assign_3_input.json) and include several graphs of varying complexity.

2. AUTOMATED TESTS

Write and include automated tests (for example, using unittest or JUnit) that verify the following for both Prim's and Kruskal's algorithms:

a. Correctness tests

• The total cost of the MST is identical for both algorithms.

• The number of edges in each MST equals $V - 1$.

• The MST contains no cycles (acyclic).

• Each MST connects all vertices (single connected component).

• Disconnected graphs are handled gracefully (no MST or clear indication).

b. Performance and consistency tests

• Execution time is non-negative and measured in milliseconds.

• Operation counts (comparisons, unions, etc.) are non-negative and consistent.

• Results are reproducible for the same dataset.

3. OUTPUT AND EVALUATION

• Record it into json file (check output.json as an example) and compare for each dataset:

• Total MST cost (Prim vs. Kruskal).

• Execution time (ms).

• Operation count.

• Summarize these results in a **concise table or CSV file** to support the final performance comparison and analysis.

REPORT REQUIREMENTS

A student must submit an analytical report including:

1. A summary of input data and algorithm results (algorithm used, execution time, and operation count for each data case);

2. A comparison between Prim's and Kruskal's algorithms in terms of efficiency and performance **(Theory and In Practice)**;

3. Conclusions discussing which algorithm is preferable under different conditions (e.g., graph density, edge representation, implementation complexity etc.);

4. References (if any external sources were used).

GRADING CRITERIA

Prim's algorithm implementation – **25%**

Kruskal's algorithm implementation – **25%**

Analytical report (results, interpretation, comparison, and conclusions) – **25%**

Code readability, commits, and GitHub repository – **15%**

Testing **– 10%**

BONUS SECTION: Graph Design in Java (10%)

To encourage clean code architecture and proper use of object-oriented programming principles, students may receive **bonus points** for implementing a custom graph data structure in Java.

Students who choose to complete this bonus task should:

1. Include both **Graph.java** and **Edge.java** files in their submission package.
2. Use their Graph class as input for Prim's and Kruskal's algorithm implementations.
3. Provide pictures of graph structure loads correctly and integrates with their MST computation.