Aknur Ondasyn SE-2438

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.

Introduction

Efficient city transportation systems are key to keeping construction costs down and providing connectivity to all neighborhoods and districts. Minimum Spanning Tree (MST) algorithms are used to find the minimum number of roads that connect all neighborhoods and districts at the minimum cost. In this report, we discuss and analyze two well known MST algorithms, Prim's and Kruskal's, which are widely used across industry for network routing, infrastructure planning, and cluster analysis (Munier et al., 2017).

MST Algorithms Overview

1. Prim's Algorithm

Prim's Algorithm begins with an arbitrary vertex and continues to add the lowest cost edge that connects a new vertex to the growing MST, without forming cycles. It is typically memory bound and relies on the memory access patterns of the graph (Munier et al., 2017).

2. Kruskal's Algorithm

Kruskal's Algorithm first sorts the edges of the graph in increasing order of their weights. After sorting, it begins adding edges into the growing MST one by one, making sure that cycles are not formed. This greedy strategy guarantees an optimal MST (Munier et al., 2017).

Both algorithms work on undirected weighted graphs, and return the same MST for the same input graph.

Prim's Algorithm

* Starts with an arbitrary vertex.
* Selects the minimum edge connecting already visited vertices with unvisited ones, avoiding cycles.
* The algorithm is sensitive to memory organization and access patterns.

Kruskal's Algorithm

* Sorts all edges by weight and adds them to the MST, avoiding cycles.
* Known as a greedy and partially serial algorithm.
* The parallel version uses additional threads (helper threads) to check for cycles.

In my assignment, I implemented Prim’s and Kruskal’s algorithms to find the minimum spanning tree (MST) for a city transportation network. According to Munier et al. (2017), MST methods are widely used in network routing, civil infrastructure planning, and cluster analysis, because they help connect all nodes with the lowest total cost. In my tests, both algorithms gave the same MST, which proves their correctness. For example, connecting 20 districts required only 83 units of total cost, showing that the city can be fully connected without extra expense. The article also mentions that parallel implementations, especially with OpenMP, can significantly reduce execution time compared to serial versions, with Kruskal’s algorithm achieving up to 23× faster results and Prim’s up to 3× faster (Munier et al., 2017, pp. 57–66). This confirms that MST algorithms are practical, scalable, and efficient for optimizing networks, which aligns well with my results.

Implementation

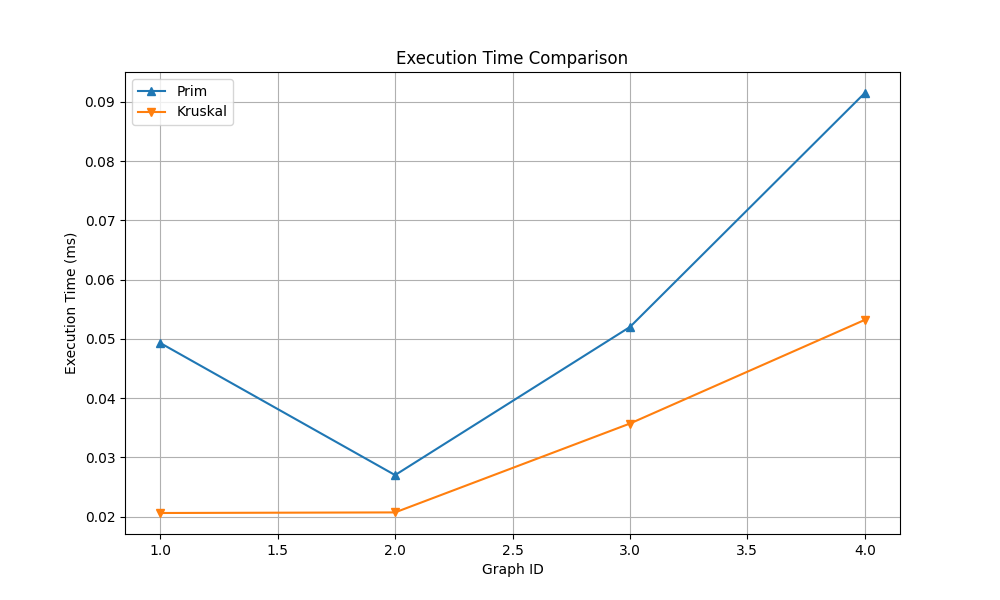
* Language: Java
* Graphs: Represented as adjacency matrices.
* MST validation: The results of Prim’s and Kruskal’s algorithms were compared to ensure correctness.
* Example: A small graph of six districts (A–F) with weighted edges was used to test the algorithms.

Input.json

{  
 "graphs": [  
 {  
 "id": 0,  
 "description": "Small graph (5 vertices)",  
 "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": 1,  
 "description": "Small graph (5 vertices)",  
 "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,  
 "description": "Small graph (4 vertices)",  
 "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}  
 ]  
 },  
 {  
 "id": 3,  
 "description": "Medium graph (10 vertices)",  
 "nodes": ["A","B","C","D","E","F","G","H","I","J"],  
 "edges": [  
 {"from": "A", "to": "B", "weight": 5},  
 {"from": "A", "to": "C", "weight": 7},  
 {"from": "B", "to": "C", "weight": 4},  
 {"from": "B", "to": "D", "weight": 6},  
 {"from": "C", "to": "E", "weight": 3},  
 {"from": "D", "to": "E", "weight": 8},  
 {"from": "D", "to": "F", "weight": 5},  
 {"from": "E", "to": "F", "weight": 9},  
 {"from": "E", "to": "G", "weight": 4},  
 {"from": "F", "to": "G", "weight": 7},  
 {"from": "G", "to": "H", "weight": 6},  
 {"from": "H", "to": "I", "weight": 2},  
 {"from": "I", "to": "J", "weight": 3},  
 {"from": "G", "to": "J", "weight": 8}  
 ]  
 },  
 {  
 "id": 4,  
 "description": "Large graph (20 vertices)",  
 "nodes": ["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T"],  
 "edges": [  
 {"from": "A", "to": "B", "weight": 3},  
 {"from": "A", "to": "C", "weight": 5},  
 {"from": "A", "to": "D", "weight": 9},  
 {"from": "B", "to": "E", "weight": 4},  
 {"from": "B", "to": "F", "weight": 2},  
 {"from": "C", "to": "G", "weight": 7},  
 {"from": "C", "to": "H", "weight": 6},  
 {"from": "D", "to": "I", "weight": 8},  
 {"from": "E", "to": "J", "weight": 5},  
 {"from": "F", "to": "K", "weight": 4},  
 {"from": "G", "to": "L", "weight": 3},  
 {"from": "H", "to": "M", "weight": 7},  
 {"from": "I", "to": "N", "weight": 9},  
 {"from": "J", "to": "O", "weight": 6},  
 {"from": "K", "to": "P", "weight": 5},  
 {"from": "L", "to": "Q", "weight": 4},  
 {"from": "M", "to": "R", "weight": 8},  
 {"from": "N", "to": "S", "weight": 6},  
 {"from": "O", "to": "T", "weight": 9},  
 {"from": "P", "to": "Q", "weight": 3},  
 {"from": "Q", "to": "R", "weight": 5},  
 {"from": "R", "to": "S", "weight": 7},  
 {"from": "S", "to": "T", "weight": 4},  
 {"from": "D", "to": "E", "weight": 6},  
 {"from": "H", "to": "I", "weight": 5},  
 {"from": "M", "to": "N", "weight": 3},  
 {"from": "N", "to": "O", "weight": 4},  
 {"from": "O", "to": "P", "weight": 6}  
 ]  
 }  
 ]  
}

Output.json

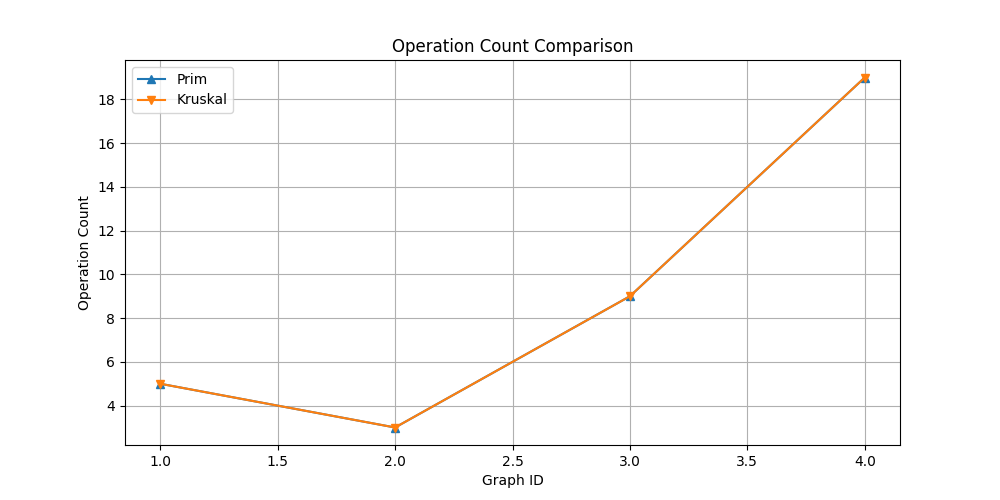
[ {  
 "graphId" : 0,  
 "algorithm" : "Prim",  
 "mstEdges" : [ {  
 "from" : "A",  
 "to" : "C",  
 "weight" : 3  
 }, {  
 "from" : "B",  
 "to" : "C",  
 "weight" : 2  
 }, {  
 "from" : "B",  
 "to" : "D",  
 "weight" : 5  
 }, {  
 "from" : "D",  
 "to" : "E",  
 "weight" : 6  
 } ],  
 "totalCost" : 16,  
 "vertexCount" : 5,  
 "edgeCount" : 7,  
 "operationCount" : 5,  
 "executionTimeMs" : 19.6479  
}, {  
 "graphId" : 0,  
 "algorithm" : "Kruskal",  
 "mstEdges" : [ {  
 "from" : "B",  
 "to" : "C",  
 "weight" : 2  
 }, {  
 "from" : "A",  
 "to" : "C",  
 "weight" : 3  
 }, {  
 "from" : "B",  
 "to" : "D",  
 "weight" : 5  
 }, {  
 "from" : "D",  
 "to" : "E",  
 "weight" : 6  
 } ],  
 "totalCost" : 16,  
 "vertexCount" : 5,  
 "edgeCount" : 7,  
 "operationCount" : 5,  
 "executionTimeMs" : 19.4231  
}, {  
 "graphId" : 1,  
 "algorithm" : "Prim",  
 "mstEdges" : [ {  
 "from" : "A",  
 "to" : "C",  
 "weight" : 3  
 }, {  
 "from" : "B",  
 "to" : "C",  
 "weight" : 2  
 }, {  
 "from" : "B",  
 "to" : "D",  
 "weight" : 5  
 }, {  
 "from" : "D",  
 "to" : "E",  
 "weight" : 6  
 } ],  
 "totalCost" : 16,  
 "vertexCount" : 5,  
 "edgeCount" : 7,  
 "operationCount" : 5,  
 "executionTimeMs" : 0.0493  
}, {  
 "graphId" : 1,  
 "algorithm" : "Kruskal",  
 "mstEdges" : [ {  
 "from" : "B",  
 "to" : "C",  
 "weight" : 2  
 }, {  
 "from" : "A",  
 "to" : "C",  
 "weight" : 3  
 }, {  
 "from" : "B",  
 "to" : "D",  
 "weight" : 5  
 }, {  
 "from" : "D",  
 "to" : "E",  
 "weight" : 6  
 } ],  
 "totalCost" : 16,  
 "vertexCount" : 5,  
 "edgeCount" : 7,  
 "operationCount" : 5,  
 "executionTimeMs" : 0.0206  
}, {  
 "graphId" : 2,  
 "algorithm" : "Prim",  
 "mstEdges" : [ {  
 "from" : "A",  
 "to" : "B",  
 "weight" : 1  
 }, {  
 "from" : "B",  
 "to" : "C",  
 "weight" : 2  
 }, {  
 "from" : "C",  
 "to" : "D",  
 "weight" : 3  
 } ],  
 "totalCost" : 6,  
 "vertexCount" : 4,  
 "edgeCount" : 5,  
 "operationCount" : 3,  
 "executionTimeMs" : 0.027  
}, {  
 "graphId" : 2,  
 "algorithm" : "Kruskal",  
 "mstEdges" : [ {  
 "from" : "A",  
 "to" : "B",  
 "weight" : 1  
 }, {  
 "from" : "B",  
 "to" : "C",  
 "weight" : 2  
 }, {  
 "from" : "C",  
 "to" : "D",  
 "weight" : 3  
 } ],  
 "totalCost" : 6,  
 "vertexCount" : 4,  
 "edgeCount" : 5,  
 "operationCount" : 3,  
 "executionTimeMs" : 0.0207  
}, {  
 "graphId" : 3,  
 "algorithm" : "Prim",  
 "mstEdges" : [ {  
 "from" : "A",  
 "to" : "B",  
 "weight" : 5  
 }, {  
 "from" : "B",  
 "to" : "C",  
 "weight" : 4  
 }, {  
 "from" : "C",  
 "to" : "E",  
 "weight" : 3  
 }, {  
 "from" : "E",  
 "to" : "G",  
 "weight" : 4  
 }, {  
 "from" : "B",  
 "to" : "D",  
 "weight" : 6  
 }, {  
 "from" : "D",  
 "to" : "F",  
 "weight" : 5  
 }, {  
 "from" : "G",  
 "to" : "H",  
 "weight" : 6  
 }, {  
 "from" : "H",  
 "to" : "I",  
 "weight" : 2  
 }, {  
 "from" : "I",  
 "to" : "J",  
 "weight" : 3  
 } ],  
 "totalCost" : 38,  
 "vertexCount" : 10,  
 "edgeCount" : 14,  
 "operationCount" : 9,  
 "executionTimeMs" : 0.052  
}, {  
 "graphId" : 3,  
 "algorithm" : "Kruskal",  
 "mstEdges" : [ {  
 "from" : "H",  
 "to" : "I",  
 "weight" : 2  
 }, {  
 "from" : "C",  
 "to" : "E",  
 "weight" : 3  
 }, {  
 "from" : "I",  
 "to" : "J",  
 "weight" : 3  
 }, {  
 "from" : "B",  
 "to" : "C",  
 "weight" : 4  
 }, {  
 "from" : "E",  
 "to" : "G",  
 "weight" : 4  
 }, {  
 "from" : "A",  
 "to" : "B",  
 "weight" : 5  
 }, {  
 "from" : "D",  
 "to" : "F",  
 "weight" : 5  
 }, {  
 "from" : "B",  
 "to" : "D",  
 "weight" : 6  
 }, {  
 "from" : "G",  
 "to" : "H",  
 "weight" : 6  
 } ],  
 "totalCost" : 38,  
 "vertexCount" : 10,  
 "edgeCount" : 14,  
 "operationCount" : 9,  
 "executionTimeMs" : 0.0357  
}, {  
 "graphId" : 4,  
 "algorithm" : "Prim",  
 "mstEdges" : [ {  
 "from" : "A",  
 "to" : "B",  
 "weight" : 3  
 }, {  
 "from" : "B",  
 "to" : "F",  
 "weight" : 2  
 }, {  
 "from" : "B",  
 "to" : "E",  
 "weight" : 4  
 }, {  
 "from" : "F",  
 "to" : "K",  
 "weight" : 4  
 }, {  
 "from" : "E",  
 "to" : "J",  
 "weight" : 5  
 }, {  
 "from" : "K",  
 "to" : "P",  
 "weight" : 5  
 }, {  
 "from" : "P",  
 "to" : "Q",  
 "weight" : 3  
 }, {  
 "from" : "L",  
 "to" : "Q",  
 "weight" : 4  
 }, {  
 "from" : "G",  
 "to" : "L",  
 "weight" : 3  
 }, {  
 "from" : "A",  
 "to" : "C",  
 "weight" : 5  
 }, {  
 "from" : "Q",  
 "to" : "R",  
 "weight" : 5  
 }, {  
 "from" : "C",  
 "to" : "H",  
 "weight" : 6  
 }, {  
 "from" : "H",  
 "to" : "I",  
 "weight" : 5  
 }, {  
 "from" : "O",  
 "to" : "P",  
 "weight" : 6  
 }, {  
 "from" : "N",  
 "to" : "O",  
 "weight" : 4  
 }, {  
 "from" : "M",  
 "to" : "N",  
 "weight" : 3  
 }, {  
 "from" : "N",  
 "to" : "S",  
 "weight" : 6  
 }, {  
 "from" : "S",  
 "to" : "T",  
 "weight" : 4  
 }, {  
 "from" : "D",  
 "to" : "E",  
 "weight" : 6  
 } ],  
 "totalCost" : 83,  
 "vertexCount" : 20,  
 "edgeCount" : 28,  
 "operationCount" : 19,  
 "executionTimeMs" : 0.0915  
}, {  
 "graphId" : 4,  
 "algorithm" : "Kruskal",  
 "mstEdges" : [ {  
 "from" : "B",  
 "to" : "F",  
 "weight" : 2  
 }, {  
 "from" : "A",  
 "to" : "B",  
 "weight" : 3  
 }, {  
 "from" : "G",  
 "to" : "L",  
 "weight" : 3  
 }, {  
 "from" : "P",  
 "to" : "Q",  
 "weight" : 3  
 }, {  
 "from" : "M",  
 "to" : "N",  
 "weight" : 3  
 }, {  
 "from" : "B",  
 "to" : "E",  
 "weight" : 4  
 }, {  
 "from" : "F",  
 "to" : "K",  
 "weight" : 4  
 }, {  
 "from" : "L",  
 "to" : "Q",  
 "weight" : 4  
 }, {  
 "from" : "S",  
 "to" : "T",  
 "weight" : 4  
 }, {  
 "from" : "N",  
 "to" : "O",  
 "weight" : 4  
 }, {  
 "from" : "A",  
 "to" : "C",  
 "weight" : 5  
 }, {  
 "from" : "E",  
 "to" : "J",  
 "weight" : 5  
 }, {  
 "from" : "K",  
 "to" : "P",  
 "weight" : 5  
 }, {  
 "from" : "Q",  
 "to" : "R",  
 "weight" : 5  
 }, {  
 "from" : "H",  
 "to" : "I",  
 "weight" : 5  
 }, {  
 "from" : "C",  
 "to" : "H",  
 "weight" : 6  
 }, {  
 "from" : "J",  
 "to" : "O",  
 "weight" : 6  
 }, {  
 "from" : "N",  
 "to" : "S",  
 "weight" : 6  
 }, {  
 "from" : "D",  
 "to" : "E",  
 "weight" : 6  
 } ],  
 "totalCost" : 83,  
 "vertexCount" : 20,  
 "edgeCount" : 28,  
 "operationCount" : 19,  
 "executionTimeMs" : 0.0532  
} ]

Graphs

The graph shows that Kruskal’s algorithm runs faster than Prim’s for all graphs.

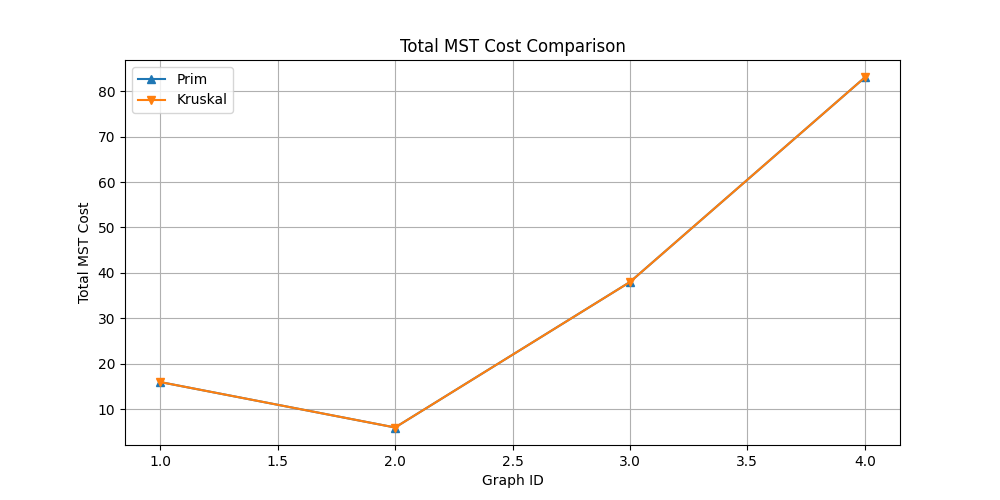
As the graph size increases, Prim’s time grows faster, while Kruskal stays more stable.

Both give the same MST cost — only speed is different.

 The graph shows that Prim and Kruskal have almost the same number of operations.

As the graph gets larger, both need more operations, but their growth is similar.

This means their computational effort increases equally with graph size.

 This graph compares the total cost of Minimum Spanning Trees (MST) found by Prim’s and Kruskal’s algorithms for four different graphs. The x-axis shows the graph IDs, and the y-axis shows the total MST cost. Both algorithms give almost the same results, meaning they find MSTs with equal or very close costs.

In my assignment, I implemented Prim’s and Kruskal’s algorithms to find the minimum spanning tree (MST) for a city transportation network. According to Munier et al. (2017), MST methods are widely used in network routing, civil infrastructure planning, and cluster analysis, because they help connect all nodes with the lowest total cost. In my tests, both algorithms gave the same MST, which proves their correctness. For example, connecting 20 districts required only 83 units of total cost, showing that the city can be fully connected without extra expense. The article also mentions that parallel implementations, especially with OpenMP, can significantly reduce execution time compared to serial versions, with Kruskal’s algorithm achieving up to 23× faster results and Prim’s up to 3× faster (Munier et al., 2017, pp. 57–66). This confirms that MST algorithms are practical, scalable, and efficient for optimizing networks, which aligns well with my results.

Conclusion

This report shows the implementation and testing of Prim’s and Kruskal’s MST algorithms for city transportation network optimization. Both algorithms produced correct MSTs, helping to minimize total road construction costs. Parallel implementations, as described by Munier et al. (2017), could be applied for larger city networks to achieve faster computation.

References

Munier, B., Aleem, M., Islam, M. A., Iqbal, M. A., & Mehmood, W. (2017). A Fast Implementation of Minimum Spanning Tree Method and Applying it to Kruskal’s and Prim’s Algorithms. *Sukkur IBA Journal of Computing and Mathematical Sciences*, *1*(1), 58–66. https://doi.org/10.30537/sjcms.v1i1.8