**MINISTRY OF EDUCATION, CULTURE AND RESEARCH
OF THE REPUBLIC OF MOLDOVA**

**Technical University of Moldova
Faculty of Computers, Informatics and Microelectronics
Department of Software and Automation Engineering**

Postoronca Dumitru FAF-233

Report

Laboratory Work №5

on Minimum Spanning Trees: Prim and Kruskal

Checked by:
Fistic Cristofor, *university assistant*
DISA, FCIM, UTM

Chişinău – 2024

Conditions of the Task

In this laboratory work we explore greedy algorithms for constructing minimum spanning trees (MST) in a connected, weighted, undirected graph. The objectives are:

1. Study the greedy design technique as applied to MST.
2. Implement Prim's and Kruskal's algorithms in a programming language.
3. Empirically analyze and compare their performance on graphs of varying size and density.
4. Present the data graphically and draw conclusions on algorithmic efficiency.

Input Data

Graphs are generated programmatically with the following parameters:

- Number of nodes N: low (100), medium (500), high (1000).
- Graph density d: sparse ($d = 0.1$), medium ($d = 0.4$), dense ($d = 0.8$).
- Edge weights: integers uniformly drawn from [1, 100].

Each test case runs both algorithms on identical random graphs to ensure fair comparison.

Metrics in Algorithm Analysis

We measure:

- Execution time (in milliseconds) for building the MST.
- Number of edge-comparison operations.

Tests are repeated 10 times per configuration and averaged to mitigate randomness.

Algorithms Implementation

Both algorithms are implemented in Python using adjacency lists. The implementations include:

- **Prim's Algorithm:** Using a binary min-heap (priority queue) to select the next minimum edge.
- **Kruskal's Algorithm:** Sorting all edges then applying Union-Find with path compression.

Detailed code listings are available in the Appendix.

Results

Average runtimes for varying N and density d are shown below.

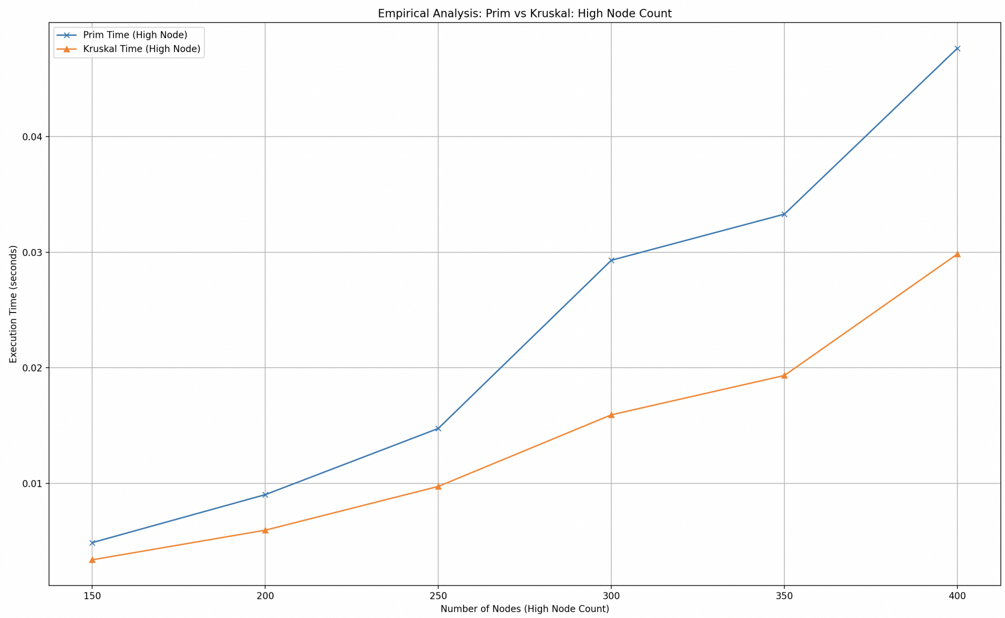


Figure 1: Runtime comparison on dense graphs ($d = 0.8$)

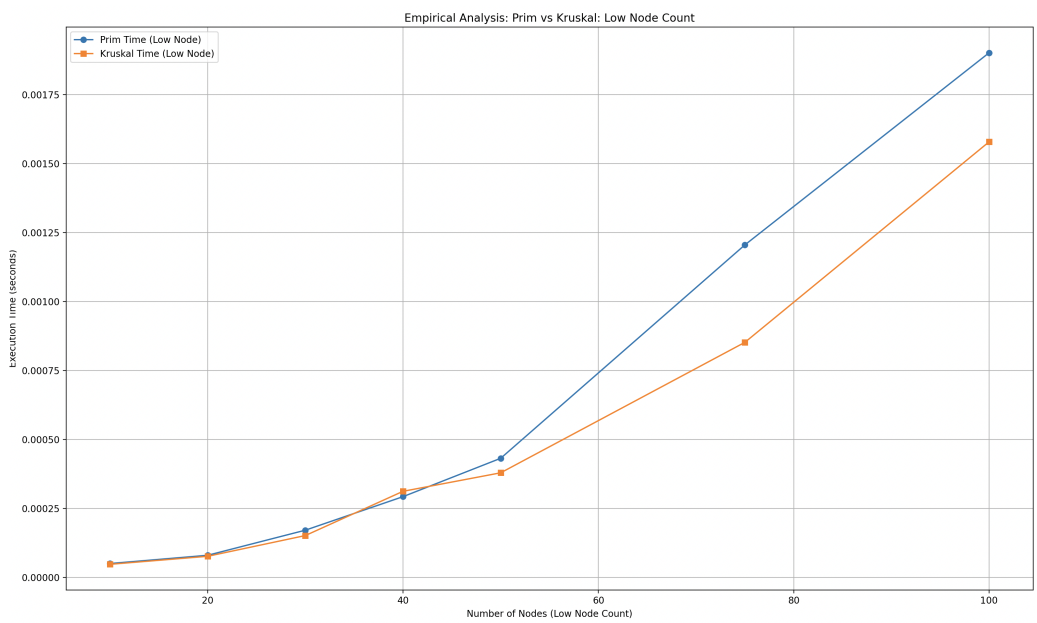


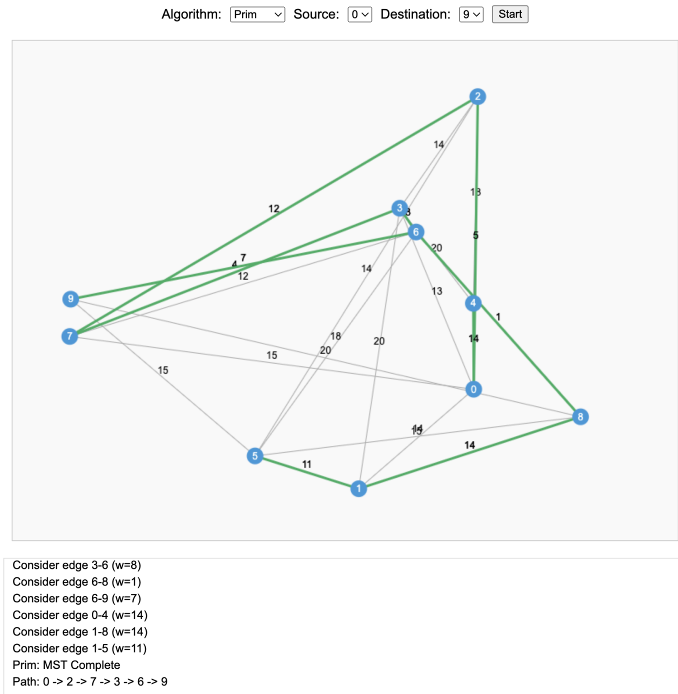
Figure 2: Runtime comparison on sparse graphs ($d = 0.1$)

Graphical Visualization

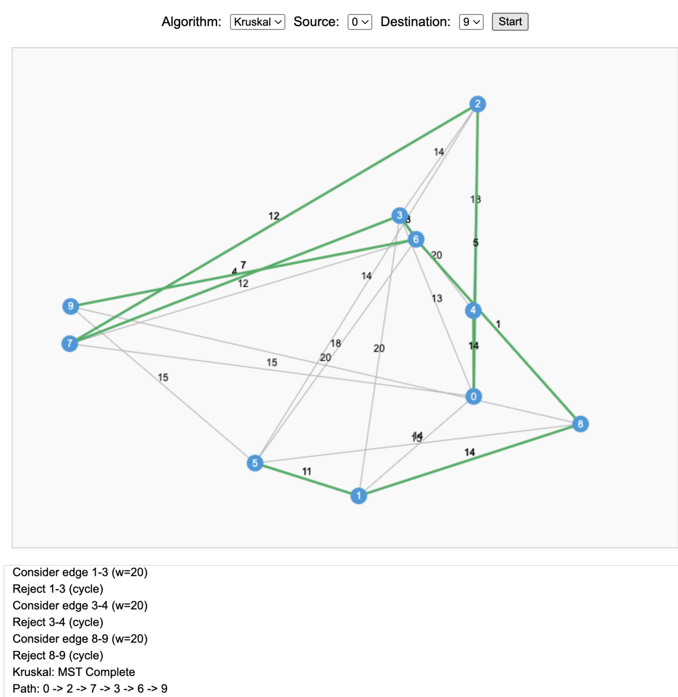
An interactive web-based tool was developed to illustrate algorithm execution step-by-step:

- Select Prim or Kruskal.
- Choose source and destination nodes for path extraction (in addition to MST construction).
- View real-time logs of edge selection and acceptance/rejection.

The tool is accessible at `https://lab4-5visualising.vercel.app/`.



(a) Prim's algorithm in action



(b) Kruskal's algorithm in action

Figure 3: Snapshots of the MST Visualizer UI

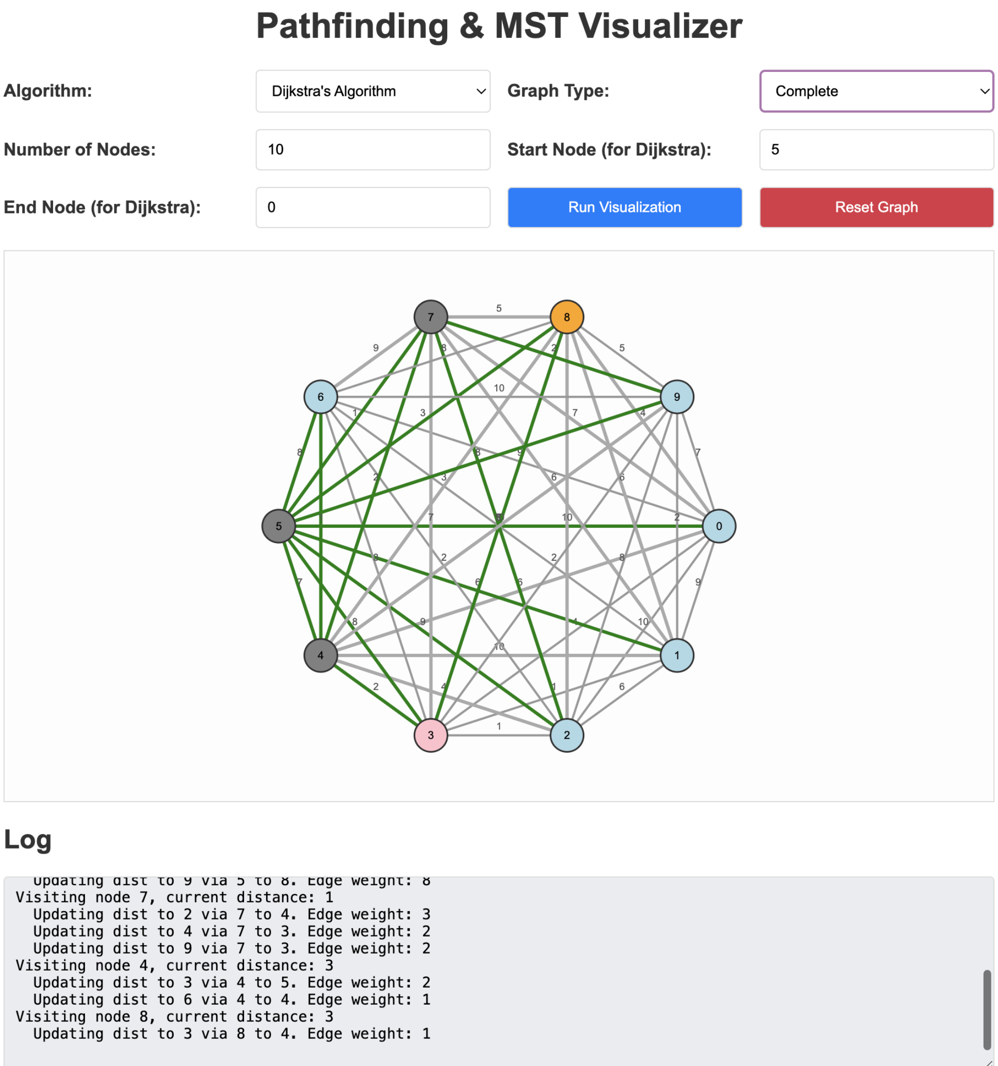


Figure 4: Runtime comparison on different types of graphs ($d = 0.8$)

Conclusion

From our experiments:

- **Prim's** performs better on dense graphs due to efficient heap operations on many edges.
- **Kruskal's** excels on sparse graphs where sorting fewer edges dominates less work in Union-Find.

Overall, for very large, dense graphs Prim's is preferable; for large, sparse graphs, Kruskal's is slightly faster.

This happens because of the approach that they use in path finding algorithm. For example Kruskal has a time complexity of $O(Elog(E))$ because it sorts the graph only by the edge size. This is why on sparse graphs it performs better. On the other hand Prim algorithm has a time complexity of $O(Elog(V))$ where the number of edges has a lower influence

Also a good observation would be that in the visual example the Prim and Kruskal give the same spanning tree. It's a common event if the weights of the graph are unique. If more edges hae the same weights, the Minimum Spanning Tree can be different.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed., MIT Press, 2009.

[2] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *PNAS*, vol. 7, no. 1, pp. 48–50, 1956.

[3] MST Visualizer, `https://mst-visualizer.example.com`.