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 №4

of AA

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

Chișinău – 2024

Conditions of the Task

Dynamic programming is defined as a computer programming technique where an algorithmic problem is first broken down into sub-problems, the results are saved, and then the sub-problems are optimized to find the overall solution — which usually has to do with finding the minimum distance in the shortest path problem.

1. Implement in a programming language the algorithms of Dijkstra and Floyd–Warshall using dynamic programming.
2. Perform empirical analysis of these algorithms on both sparse and dense graphs.
3. Increase the number of nodes in graphs and analyze how this influences algorithm performance.
4. Make a graphical presentation of the data obtained.
5. Draw conclusions based on the analysis performed.

Input data

To analyze and compare the pathfinding algorithms, we define the following input data conditions:

1. Number of edges
    - Sparse graph
    - Dense graph
2. Edge Cases:
    - Negative weights (Floyd–Warshall supports them; Dijkstra does not)
    - Cyclic graphs

Input data was generated by a special function written in JS that takes as input the wideness of the graph (number of direct children each node will have) and its depth. Total number of nodes is calculated using the formula:

$$N_{nodes} = width^{depth}$$

Metrics in Algorithm analysis

In this laboratory work I test how each pathfinding algorithm performs depending on the number of edges

For this purpose I run the same algorithm to find the shortest path from the root node to a leaf node at various depths. If the depth of the generated graph is 5, I measure metrics for: depth 1 through depth 5.

Algorithms Implementation

Implementation of the algorithm was done using python and visual representation is done using matplotlib.

Results

After running the tests, the following results were obtained:

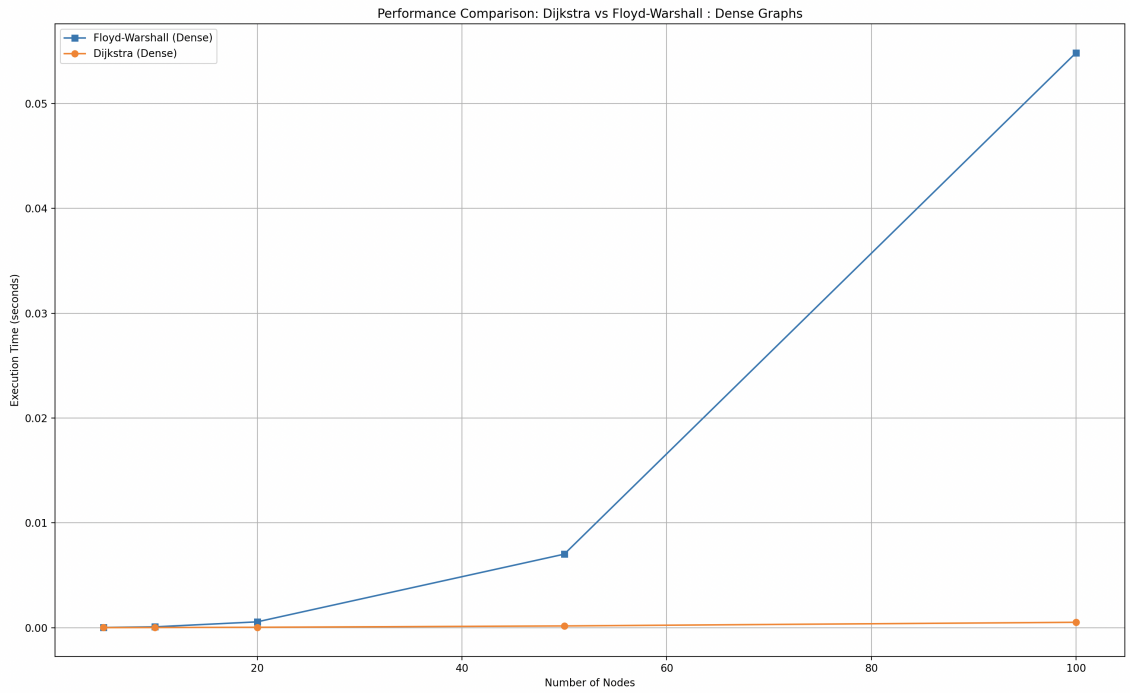

Figure 1: Pathfinding comparison for a dense graph

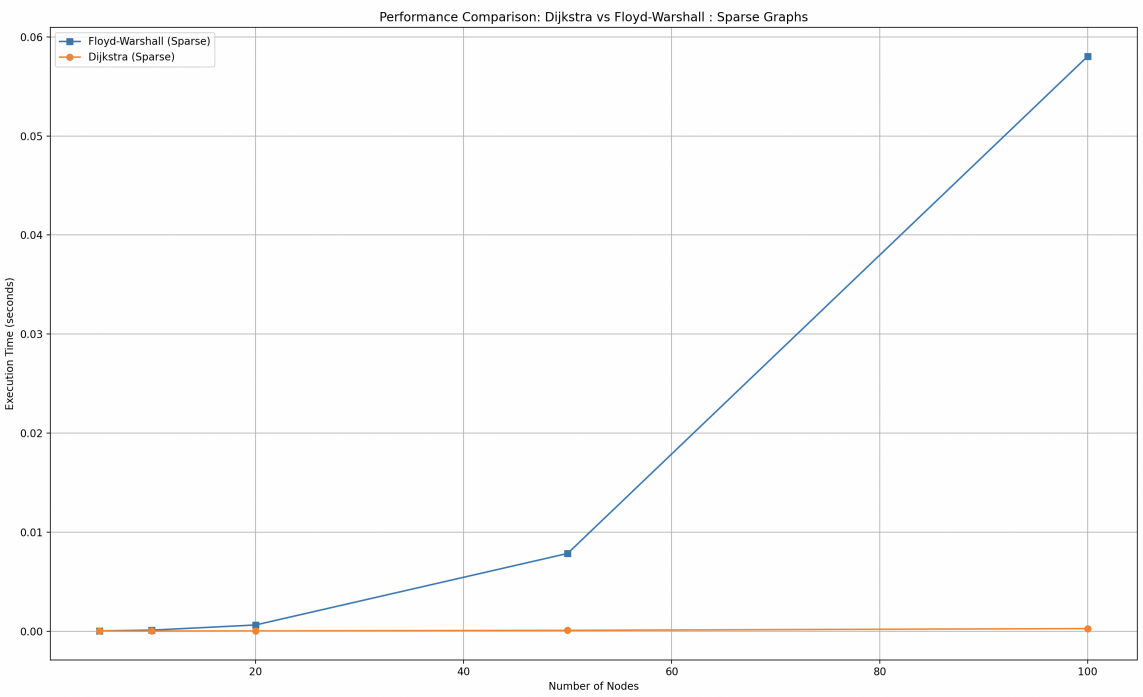

Figure 2: Pathfinding comparison for a sparse graph

As observed in the results, the Dijkstra algorithm performs efficiently for graphs without negative weights, providing accurate shortest paths and generally being faster on sparse graphs due to reduced edge exploration.

Floyd–Warshall, on the other hand, calculates all-pairs shortest paths and is more suited for dense graphs. It gracefully handles negative weights but has higher time complexity.

The time complexities are:

- **Dijkstra:** $O((V + E) \log V)$ with a priority queue
- **Floyd–Warshall:** $O(V^3)$

Despite its simplicity and power, Floyd–Warshall becomes impractical for large graphs due to its cubic time complexity.

Graphical Representation

A web-based visualization was developed to demonstrate how Dijkstra and Floyd–Warshall algorithms operate. It can be accessed online[3].

The user interface provides the following features:

- Recieve different combinations of the edges and weights at each reload
- Switch between Dijkstra and Floyd–Warshall
- View execution logs for analysis

Conclusion

In conclusion I can say that Dijkstra is a very powerful path finding algorithm due to its low time complexity and performs efficiently in finding the shortest path between 2 points.

But Floyd-Warshall algorithm is also a valid variant of finding the shortes path in cases when we need a full information about each of the nodes on a graph. Besides that Floyd-Warshall can handle negative weights and graphs with negative cycles, which Dijkstra just cann't do.

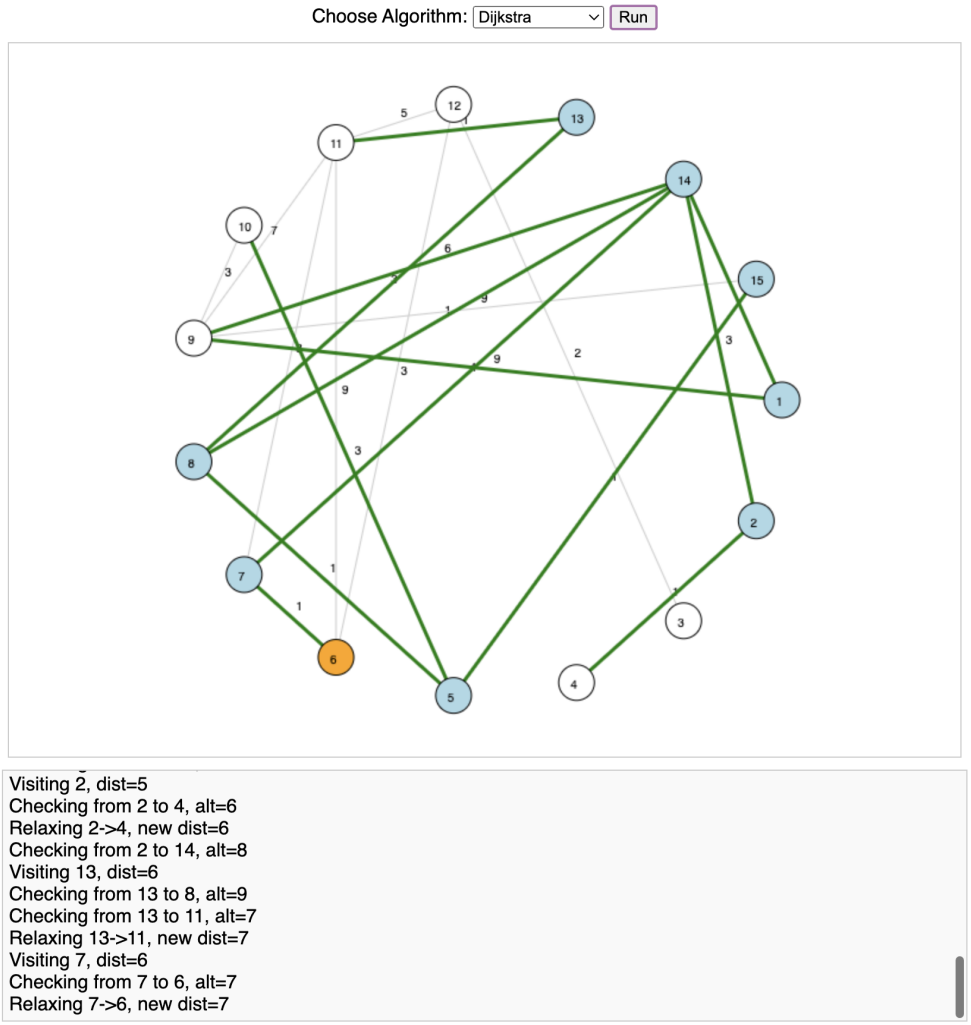

Figure 3: Graphical representation of Dijkstra and Floyd–Warshall algorithms

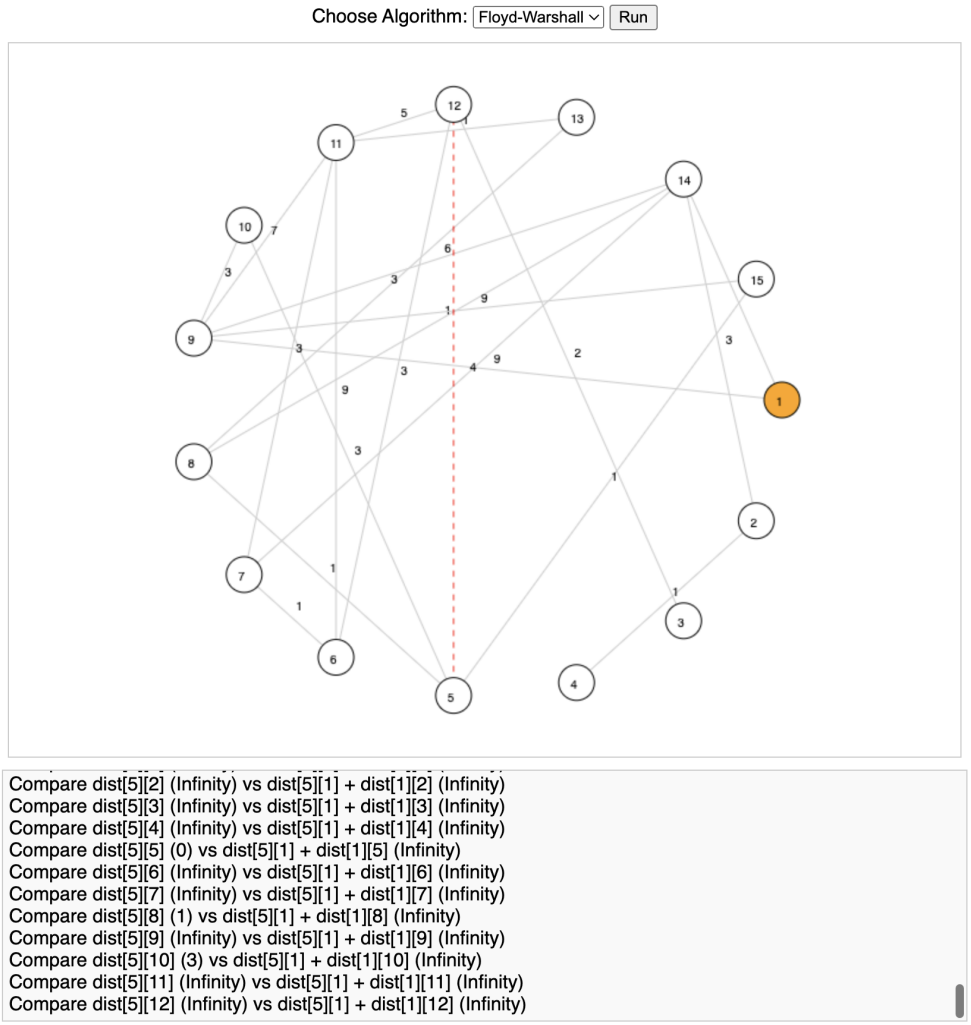

Figure 4: Graphical representation of Dijkstra and Floyd–Warshall algorithms

References

[1] Franceska Hinkelman (2017) - Understanding bytecode *Medium*

[2] GitHub repository of current laboratory work

[3] Graphical representation of Dijkstra and Floyd–Warshall