**Ministerul Educaţiei și Cercetării al Republicii Moldova**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

**REPORT**

Laboratory Work nr.5

*at Algorithms Analysis*

Elaborated by:

st. gr. FAF-211 Hariton Dan

Verified by:

asist. univ. Fiștic Cristofor

Chişinău - 2023

**ALGORITHM ANALYSIS**

**Subject:** Empirical analysis of algorithms: Disjktra and Floyd-Warshall.

**Tasks:**

1. Implement the algorithms listed below in a programming language
2. Establish the properties of the input data against which the analysis is performed
3. Choose metrics for comparing algorithms
4. Perform empirical analysis of the proposed algorithms
5. Make a graphical presentation of the data obtained
6. Make a conclusion on the work done.

**Establish Comaparation:**

We will compare these 2 algorithms we used on a sparce graph, and an dense graph.

**Comparison Metric:**

The comparison metric for this laboratory work will be considered the time of execution of each algorithm (T(n))

**IMPLEMENTATION**

**Dijkstra:**

The algorithm maintains a set of visited vertices and a set of unvisited vertices. It starts by assigning a tentative distance to each unvisited vertex, with the distance to the starting vertex set to zero. It then selects the vertex with the smallest tentative distance and marks it as visited. For each unvisited neighbor of the visited vertex, it updates their tentative distance if it is smaller than the current tentative distance. This process continues until all vertices have been visited, or until the destination vertex has been reached.

def dijkstra(graph, start):  
 n = len(graph)  
 dist = [float('inf')] \* n  
 visited = [False] \* n  
 dist[start] = 0  
 for \_ in range(n):  
 u = min(filter(lambda x: not visited[x], range(n)), key=dist.\_\_getitem\_\_)  
 visited[u] = True  
 for v in range(n):  
 if graph[u][v] != 0 and not visited[v]:  
 alt = dist[u] + graph[u][v]  
 if alt < dist[v]:  
 dist[v] = alt  
 return dist

## Floyd-Warshall:

## Floyd-Warshall is an algorithm that finds the shortest path between all pairs of vertices in a weighted graph. It works for graphs with positive, negative, or zero edge weights. The algorithm uses a matrix to store the shortest path between every pair of vertices and has a time complexity of O(|V|^3), where |V| is the number of vertices in the graph. It is useful for small graphs or when the graph is not too dense.

def floyd\_warshall(graph):  
 n = len(graph)  
 dist = graph.copy()  
 for k in range(n):  
 for i in range(n):  
 for j in range(n):  
 dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])  
 return dist

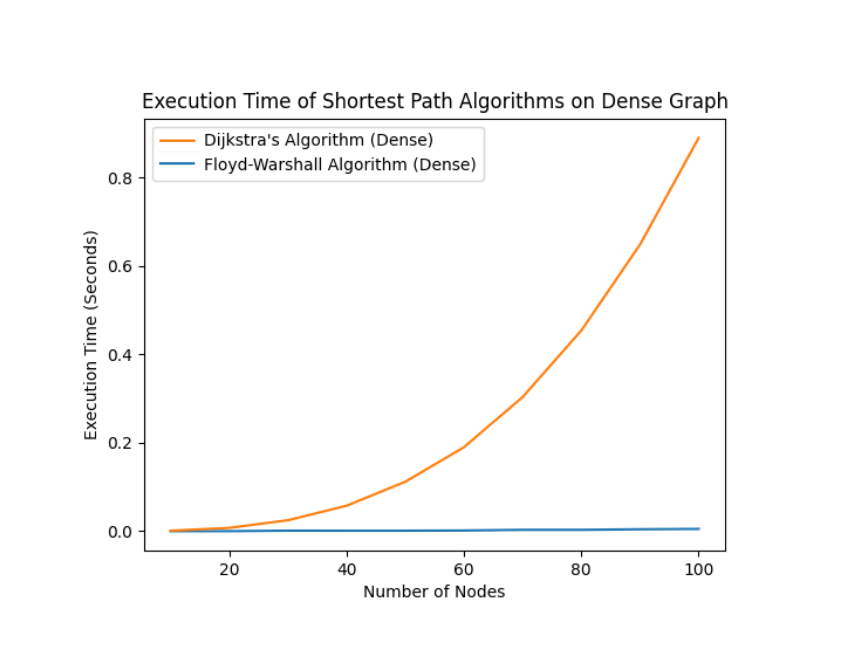
## Here are the graphs used

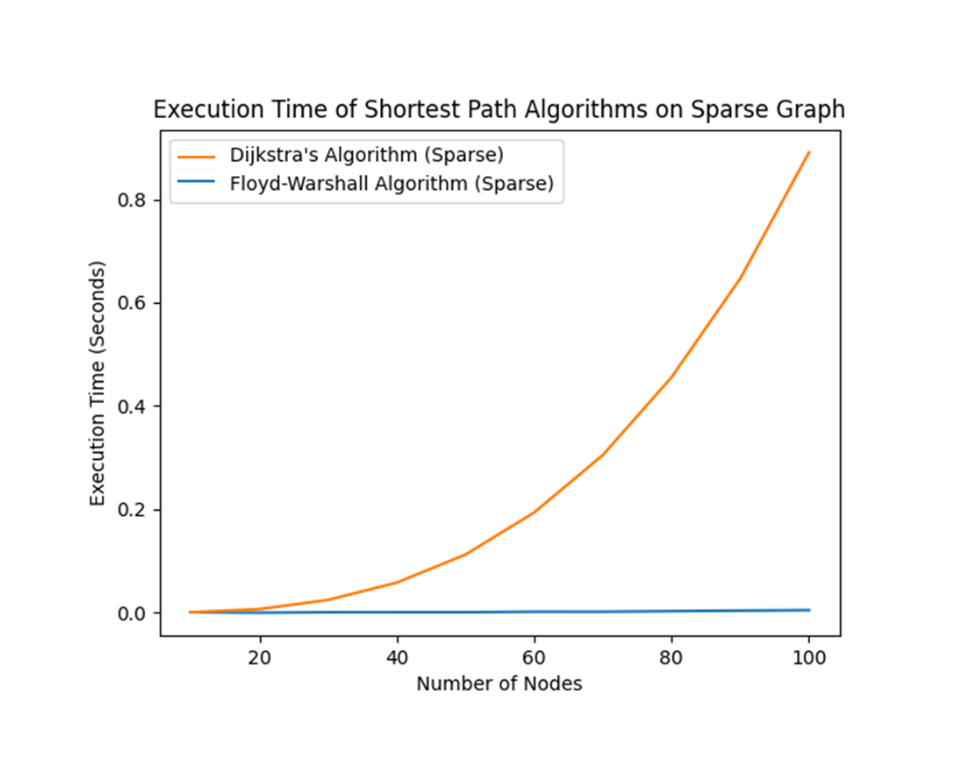
## Dense:

## Sparse:

## 

**Results:**

****

****

**Conclusion:**

The proposed work aimed to analyze the time-based and theoretical complexities of the Floyd-Warshall and Dijkstra's algorithms. We used two different algorithms to determine their temporal complexity and highlight the most efficient algorithm that will display the desired result using the least time that passed by. From the analysis, it can be concluded that the Dijkstra's algorithm is more efficient than the Floyd-Warshall algorithm in finding the shortest path between a single source node and all other nodes in a graph. This is due to the fact that Dijkstra's algorithm is a single-source algorithm and only processes the nodes that are needed to find the shortest path, while the Floyd-Warshall algorithm processes all pairs of nodes in the graph. On the other hand, the Floyd-Warshall algorithm is more efficient in finding the shortest path between all pairs of nodes in a graph. This is due to the fact that it computes all possible paths in the graph in a single run, while Dijkstra's algorithm would need to be run for each source node separately. In conclusion, both algorithms are very efficient and the choice of algorithm to use depends on the specific problem at hand. If the goal is to find the shortest path between a single source node and all other nodes in the graph, then Dijkstra's algorithm is the better choice. If the goal is to find the shortest path between all pairs of nodes in the graph, then the Floyd-Warshall algorithm is the better choice.

**Link to GitHub**: <https://github.com/haritondan/AA-Labs>