Ministry of Education of Republic of Moldova
Technical University of Moldova
Faculty of Computers, Informatics and Microelectronics

Laboratory No.2 Study and empirical analysis of sorting algorithms. Analysis of quickSort, mergeSort, heapSort, one of choice.

Verified:
Fistic Cristofor asist. univ.

Titerez Vladislav FAF-233

Moldova, April 2025

Contents

1 Algorithm Analysis

1.1 Objective

The objective of this work is to conduct an empirical analysis of Dijkstra's Algorithm and Floyd–Warshall Algorithm. The study aims to implement these algorithms, analyze their efficiency based on various input conditions, and compare them using relevant performance metrics. Through graphical representations of the results, we seek to draw conclusions about their practical efficiency in different scenarios.

1.2 Tasks

To achieve the outlined objective, the following tasks will be performed: 1. Implement the Dijkstra's and Floyd–Warshall algorithms in a chosen programming language. 2. Define the characteristics of the input data to be used in the analysis. 3. Select appropriate metrics for comparing the algorithms, such as execution time, memory usage, and node traversal count. 4. Conduct empirical experiments to measure and compare the performance of the algorithms. 5. Present the obtained results in graphical form to highlight performance differences. 6. Draw conclusions based on the empirical findings, discussing the strengths and weaknesses of each algorithm.

1.3 Theoretical Notes

Graph algorithms play a fundamental role in computer science, particularly in network routing, shortest path problems, and graph theory. The two algorithms in focus, Dijkstra's Algorithm and the Floyd–Warshall Algorithm, are both used for finding the shortest paths in a graph but differ in their approach and use cases.

Dijkstra's Algorithm is a greedy algorithm used to find the shortest path from a source node to all other nodes in a weighted graph with non-negative edge weights. Dijkstra's Algorithm is commonly used in applications such as GPS navigation, network routing, and AI pathfinding. Its time complexity is $O((V + E) \log V)$ when using

a priority queue, where V is the number of vertices and E is the number of edges.

Floyd–Warshall Algorithm is a dynamic programming approach used to find the shortest paths between all pairs of vertices in a weighted graph. Unlike Dijkstra's, which computes single-source shortest paths, Floyd-Warshall computes the shortest paths between every pair of vertices. This algorithm has a time complexity of $O(V^3)$, making it less efficient for large graphs, but useful for dense graphs and small to medium-sized datasets.

Key properties influencing algorithm performance include: - Graph density (sparse vs. dense graphs) - Graph structure (directed vs. undirected, cyclic vs. acyclic) - Graph size (number of nodes and edges) - Memory usage (Dijkstra's Algorithm requires less memory than Floyd–Warshall in most cases)

Empirical analysis helps in evaluating these properties by measuring execution time, memory consumption, and other relevant factors under different input conditions.

1.4 Introduction

Shortest path algorithms are critical in a variety of fields including network routing, transportation, and artificial intelligence. Two of the most widely used shortest path algorithms are Dijkstra's Algorithm and the Floyd–Warshall Algorithm. While both algorithms are designed to find shortest paths, their methods and performance characteristics vary significantly.

Dijkstra's Algorithm uses a greedy approach, incrementally exploring the shortest path from the source node to all other nodes. It is highly efficient for sparse graphs and is used in real-world applications like GPS navigation and network routing. Dijkstra's Algorithm can be implemented using a priority queue, allowing for efficient pathfinding in large graphs with relatively low complexity.

The Floyd–Warshall Algorithm, on the other hand, uses dynamic programming to compute the shortest paths between all pairs of nodes in a graph. It is particularly effective for dense graphs, but its cubic time complexity makes it less suitable for large graphs. The Floyd–Warshall algorithm is useful in applications like routing tables and network analysis where all-pairs shortest path information is needed.

Despite their similarities, Dijkstra's and Floyd-Warshall's performance varies greatly depending on factors such as the size and structure of the graph, the number of nodes and edges, and the specific problem requirements. Through empirical analysis, this study will compare these algorithms, present the results graphically, and draw conclusions about their practical efficiency in different scenarios.

2 Implementation

These two algorithms, Dijkstra and Floyd–Warshall, will be implemented in their native form in JavaScript and analyzed empirically based on the time required for their completion. While the general trend of the results may be similar to other experimental observations, the particular efficiency in relation to input will vary depending on the memory of the device used.
The error margin determined will constitute 0.0005 seconds as per experimental measurement.

2.1 Dijkstra's Algorithm

2.1.1 Introduction

Dijkstra's algorithm is a dynamic programming-based algorithm used for finding the shortest path from a single source node to all other nodes in a weighted graph with non-negative edge weights. It is commonly used in network routing protocols, mapping services, and AI pathfinding.

2.1.2 Working Principle

The working principle of Dijkstra's algorithm follows a **greedy** approach using a priority queue:

1. Initialize distances to all vertices as infinite, except for the source vertex, which is set to zero.
2. Use a priority queue to repeatedly extract the vertex with the minimum distance.
3. Update the distance values of adjacent vertices if a shorter path is found.
4. Repeat until all vertices are processed.

2.1.3 Advantages

Dijkstra's algorithm offers several advantages:

- Guarantees the shortest path in graphs with non-negative weights.
- Efficient for graphs with sparse connections using a priority queue.
- Used in real-world applications like GPS navigation and network routing.

2.1.4 JS Implementation

```
function dijkstra(graph, source) {
  let distances = {};
  let visited = {};
  let pq = new MinPriorityQueue();

  distances[source] = 0;
  pq.enqueue(source, 0);

  while (!pq.isEmpty()) {
    let { element: node } = pq.dequeue();
    if (visited[node]) continue;
    visited[node] = true;
    for (let neighbor in graph[node]) {
      let newDist = distances[node] + graph[node][neighbor];
      if (newDist < (distances[neighbor] || Infinity)) {
        distances[neighbor] = newDist;
        pq.enqueue(neighbor, newDist);
      }
    }
  }
  return distances;
}
```

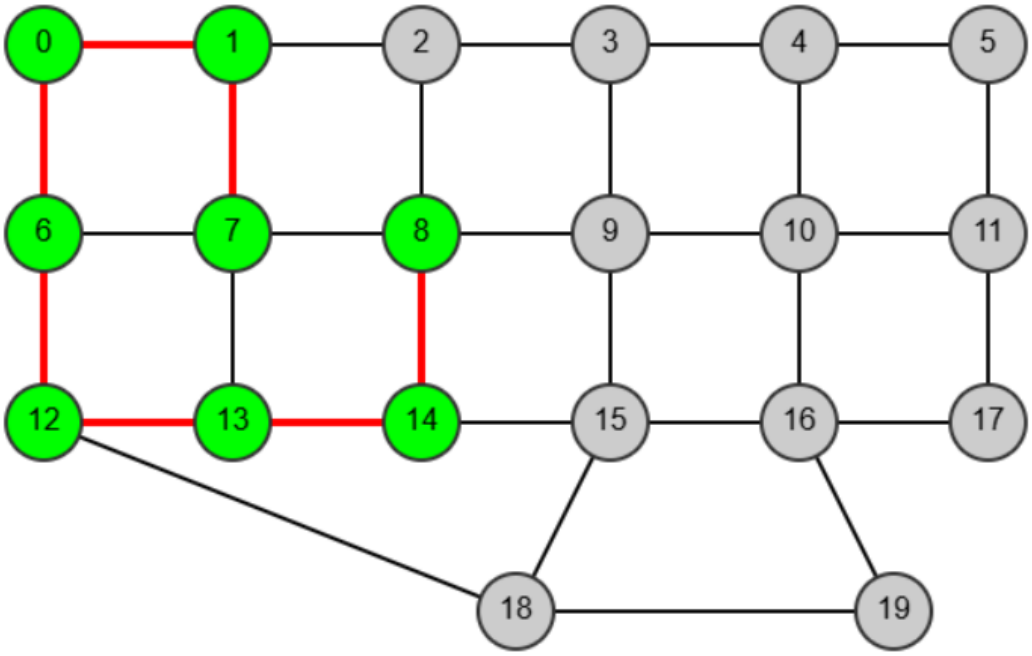

Figure 2.1: *Dijkstra's Algorithm graph*

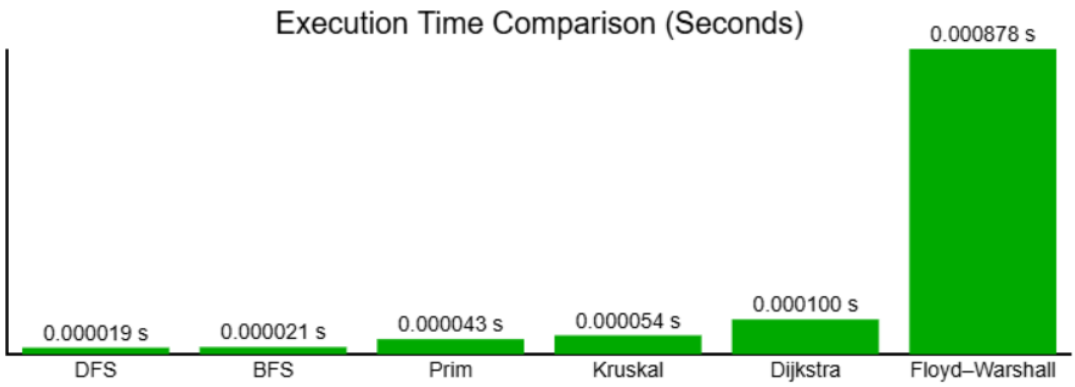

Figure 2.2: *Dijkstra's Algorithm performance*

2.1.5 Time Complexity

- Using a priority queue: $O((V + E) \log V)$
- Without a priority queue: $O(V^2)$

2.1.6 Auxiliary Space

The auxiliary space complexity of Dijkstra's algorithm is $O(V + E)$ when using an adjacency list representation with a priority queue.

2.2 Floyd–Warshall Algorithm

2.2.1 Introduction

The Floyd–Warshall algorithm is a dynamic programming approach to find the shortest paths between all pairs of vertices in a weighted graph. It is particularly useful for dense graphs and small to medium-sized datasets.

2.2.2 Working Principle

The algorithm iteratively updates the shortest paths using a three-level nested loop:

1. Initialize a distance matrix with direct edge weights, setting diagonal elements to zero.
2. Iterate through each vertex as an intermediate node.
3. Update distances between all pairs using the intermediate node.

2.2.3 Advantages

Floyd–Warshall provides:

- A simple, elegant solution for all-pairs shortest path problems.
- Applicability in network analysis and routing tables.
- A guaranteed solution even for dense graphs.

2.2.4 JS Implementation

```
function floydWarshall(graph) {
  let dist = JSON.parse(JSON.stringify(graph));
  let V = graph.length;
  for (let k = 0; k < V; k++) {
    for (let i = 0; i < V; i++) {
      for (let j = 0; j < V; j++) {
        if (dist[i][k] + dist[k][j] < dist[i][j]) {
          dist[i][j] = dist[i][k] + dist[k][j];
        }
      }
    }
  }
  return dist;
}
```

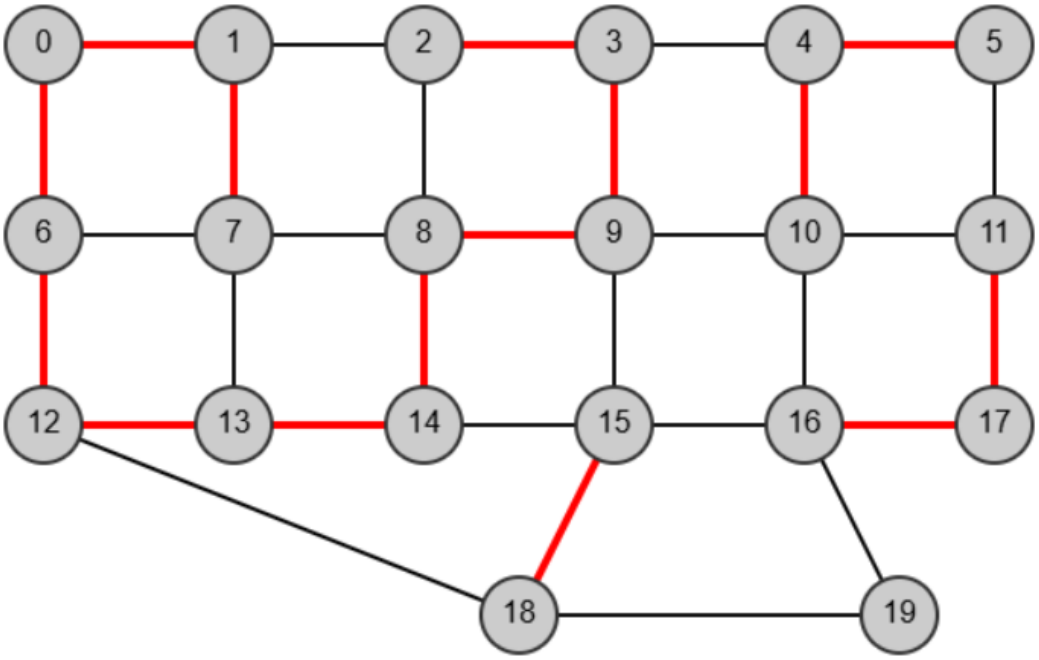

Figure 2.3: *Floyd–Warshall Algorithm graph*

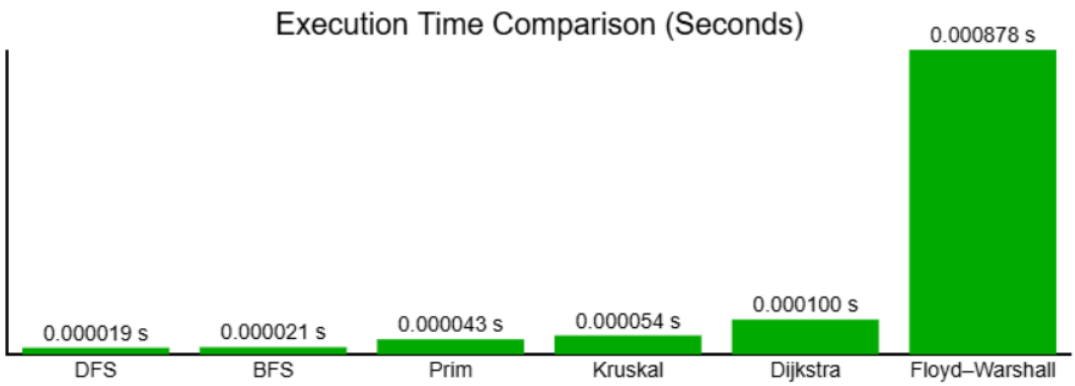

Figure 2.4: *Floyd–Warshall Algorithm performance*

2.2.5 Time Complexity

- Worst Case: $O(V^3)$
- Best Case: $O(V^2)$

2.2.6 Auxiliary Space

The auxiliary space complexity of Floyd–Warshall is $O(V^2)$ due to the need to store the distance matrix.

3 Conclusion

This paper has examined the concept of dynamic programming as a method for designing efficient algorithms by breaking problems into overlapping subproblems, storing intermediate results, and optimizing recursive solutions. Specifically, the Dijkstra and Floyd–Warshall algorithms were analyzed in the context of shortest path computation, highlighting their efficiency, time complexity, and practical applications.

Dijkstra's Algorithm

Dijkstra's algorithm is a classic example of dynamic programming used to find the shortest path from a single source vertex to all other vertices in a weighted graph. With a time complexity of $O(V^2)$ for an adjacency matrix implementation and $O((V + E) \log V)$ when using a priority queue, it is highly efficient for graphs with non-negative weights. Dijkstra's algorithm is widely used in network routing, GPS navigation, and AI pathfinding. However, its reliance on a priority queue makes it less effective for graphs with negative weights.

Floyd–Warshall Algorithm

The Floyd–Warshall algorithm is an all-pairs shortest path algorithm that applies dynamic programming principles by iteratively refining path costs between all vertex pairs. Its time complexity of $O(V^3)$ makes it more suitable for dense graphs where all-pairs computations are necessary. While highly effective for smaller graphs, its cubic complexity can be a limitation in large-scale applications. Floyd–Warshall is commonly used in network analysis, transitive closure computations, and routing optimization.

Summary

Each algorithm has distinct use cases and trade-offs:

- **Dijkstra's Algorithm**: Best suited for single-source shortest path problems in graphs with non-negative weights.
- **Floyd–Warshall Algorithm**: Optimal for all-pairs shortest path problems in dense graphs.

Future research could explore hybrid approaches that combine aspects of both algorithms, such as optimizing Dijkstra's algorithm for negative-weight graphs or improving the scalability of Floyd–Warshall using parallel computing techniques. Additionally, advancements in memory-efficient implementations could further enhance their real-world applicability. https://github.com/vvtttvv/AA/tree/main/lab3_4_5