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 Greedy Algorithms

1.1 Objective

The objective of this work is to conduct an empirical analysis of greedy algorithms, specifically focusing on Prim's Algorithm and Kruskal's Algorithm. The study aims to implement these algorithms, analyze their efficiency based on different input conditions, and compare their performance using relevant metrics. Through graphical representations of the results, we seek to evaluate their practical efficiency in various scenarios.

1.2 Tasks

To achieve the outlined objective, the following tasks will be performed: 1. Study the greedy algorithm design technique and its core principles. 2. Implement Prim's and Kruskal's algorithms in a chosen programming language. 3. Define the characteristics of the input data for analysis. 4. Select appropriate metrics for comparing the algorithms, such as execution time, memory usage, and edge selection count. 5. Conduct empirical experiments by increasing the number of nodes in the graph and analyzing the impact on performance. 6. Present the obtained results in graphical form to highlight performance differences. 7. Compile the findings into a comprehensive report discussing the strengths and weaknesses of each algorithm.

1.3 Theoretical Notes

Greedy algorithms form a fundamental class of optimization techniques in computer science. These algorithms make locally optimal choices at each step with the hope of finding a globally optimal solution. Although they do not always guarantee the best solution, they are widely used due to their efficiency and simplicity.

Prim's Algorithm is a greedy algorithm used to find the Minimum Spanning Tree (MST) of a weighted graph. It starts from a single node and continuously adds the smallest edge that connects a new vertex to the tree. The time complexity of Prim's Algorithm is $O(E \log V)$ when implemented with a priority queue, where V is the number of vertices and E is the number of edges.

Kruskal's Algorithm is another greedy approach for finding the MST. Instead of growing the tree from a starting vertex, it sorts all edges by weight and adds them to the MST in increasing order while avoiding cycles. Kruskal's Algorithm typically has a time complexity of $O(E \log E)$, making it efficient for graphs with fewer edges.

Key properties influencing algorithm performance include: - Graph density (sparse vs. dense graphs) - Graph structure (connected vs. disconnected, directed vs. undirected) - Graph size (number of nodes and edges) - Memory usage and sorting complexity

Empirical analysis allows evaluation of these properties by measuring execution time, memory consumption, and other relevant factors under different input conditions.

1.4 Introduction

Greedy algorithms provide efficient solutions for a variety of optimization problems by making locally optimal choices at each step. These algorithms work in a top-down manner and do not backtrack once a decision is made. While they do not always guarantee the global optimal solution, they are often used for problems where an optimal substructure and greedy choice property exist.

Prim's and Kruskal's algorithms are two widely used greedy algorithms for finding the Minimum Spanning Tree (MST) of a graph. While Prim's Algorithm builds the MST incrementally by adding the smallest edge that expands the tree, Kruskal's Algorithm sorts all edges and selects the smallest ones while ensuring no cycles are formed. The performance of these algorithms varies depending on the number of nodes and edges in the graph.

By conducting empirical experiments, this study aims to analyze how increasing the number of nodes in a graph affects the efficiency of Prim's and Kruskal's algorithms. The results will be presented graphically to compare their performance in different scenarios, highlighting their advantages and limitations.

2 Implementation

These two algorithms, Prim's and Kruskal's, 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 Prim's Algorithm

2.1.1 Introduction

Prim's algorithm is a greedy algorithm used for finding the Minimum Spanning Tree (MST) of a weighted, connected, and undirected graph. It starts with a single vertex and iteratively adds the lowest-weight edge that expands the MST.

2.1.2 Working Principle

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

1. Initialize all vertices as not included in MST.
2. Set the weight of the starting vertex to zero and all others to infinity.
3. Use a priority queue to extract the vertex with the minimum key value.
4. Update the key values of adjacent vertices if a smaller weight is found.
5. Repeat until all vertices are included in the MST.

2.1.3 Advantages

Prim's algorithm offers several advantages:

- Works efficiently for dense graphs with many edges.

- Ensures that a connected MST is built step-by-step.
- Can be optimized using a priority queue for improved performance.

2.1.4 JS Implementation

```javascript
function prim(graph) {
  let mstSet = new Set();
  let key = {};
  let parent = {};
  let pq = new MinPriorityQueue();

  for (let vertex in graph) {
    key[vertex] = Infinity;
    parent[vertex] = null;
  }

  let startVertex = Object.keys(graph)[0];
  key[startVertex] = 0;
  pq.enqueue(startVertex, 0);

  while (!pq.isEmpty()) {
    let { element: u } = pq.dequeue();
    mstSet.add(u);

    for (let v in graph[u]) {
      if (!mstSet.has(v) && graph[u][v] < key[v]) {
        key[v] = graph[u][v];
        parent[v] = u;
        pq.enqueue(v, key[v]);
      }
    }
  }
  return parent;
}
```

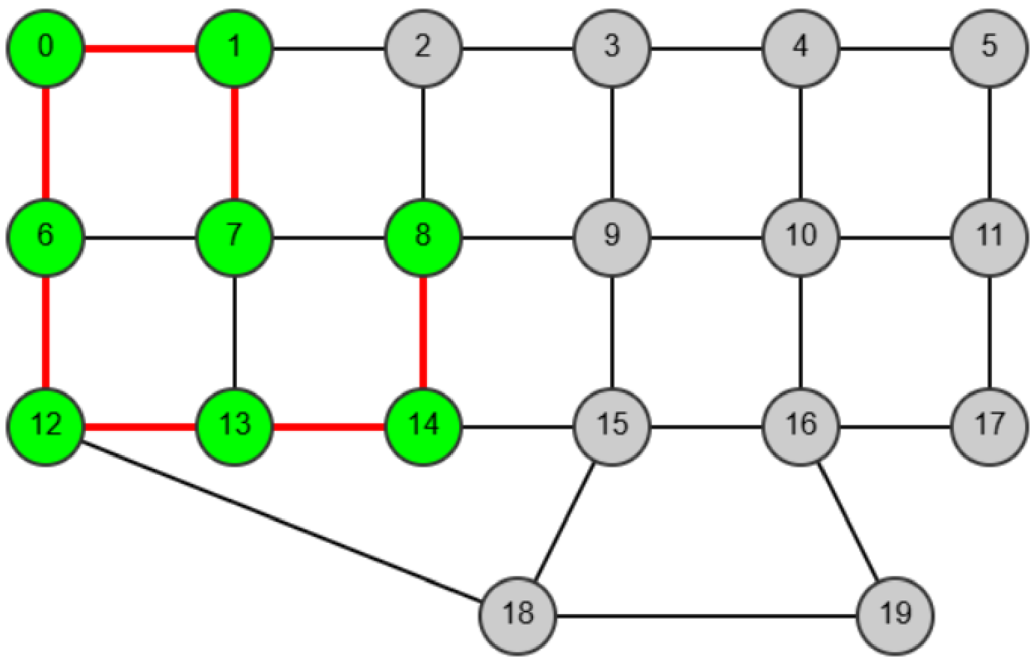

Figure 2.1: *Prim's Algorithm graph*

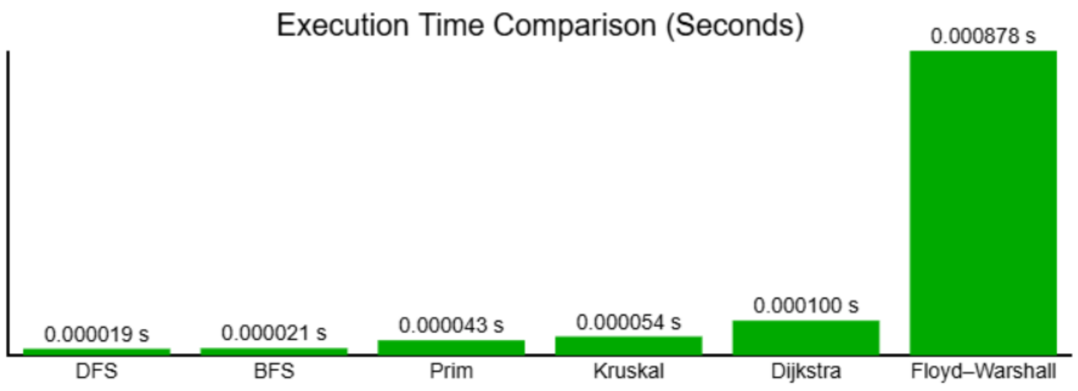

Figure 2.2: *Prim's Algorithm performance*

2.1.5 Time Complexity

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

2.1.6 Auxiliary Space

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

2.2 Kruskal's Algorithm

2.2.1 Introduction

Kruskal's algorithm is a greedy algorithm used for constructing the Minimum Spanning Tree by selecting edges in increasing order of weight while ensuring no cycles are formed.

2.2.2 Working Principle

The algorithm follows these steps:

1. Sort all edges in non-decreasing order of weight.
2. Initialize a disjoint-set data structure to manage connected components.
3. Iterate through the sorted edges and add an edge to the MST if it does not form a cycle.
4. Repeat until $V - 1$ edges are included in the MST.

2.2.3 Advantages

Kruskal's algorithm provides:

- An efficient solution for sparse graphs.
- A simple and intuitive approach to building an MST.
- The ability to work well with edge-list representations of graphs.

2.2.4 JS Implementation

```
function kruskal(graph) {
  let edges = [];
  let mst = [];
  let ds = new DisjointSet();

  for (let u in graph) {
    for (let v in graph[u]) {
      edges.push([u, v, graph[u][v]]);
    }
  }

  edges.sort((a, b) => a[2] - b[2]);

  for (let [u, v, weight] of edges) {
    if (ds.find(u) !== ds.find(v)) {
      ds.union(u, v);
      mst.push([u, v, weight]);
    }
  }
  return mst;
}
```

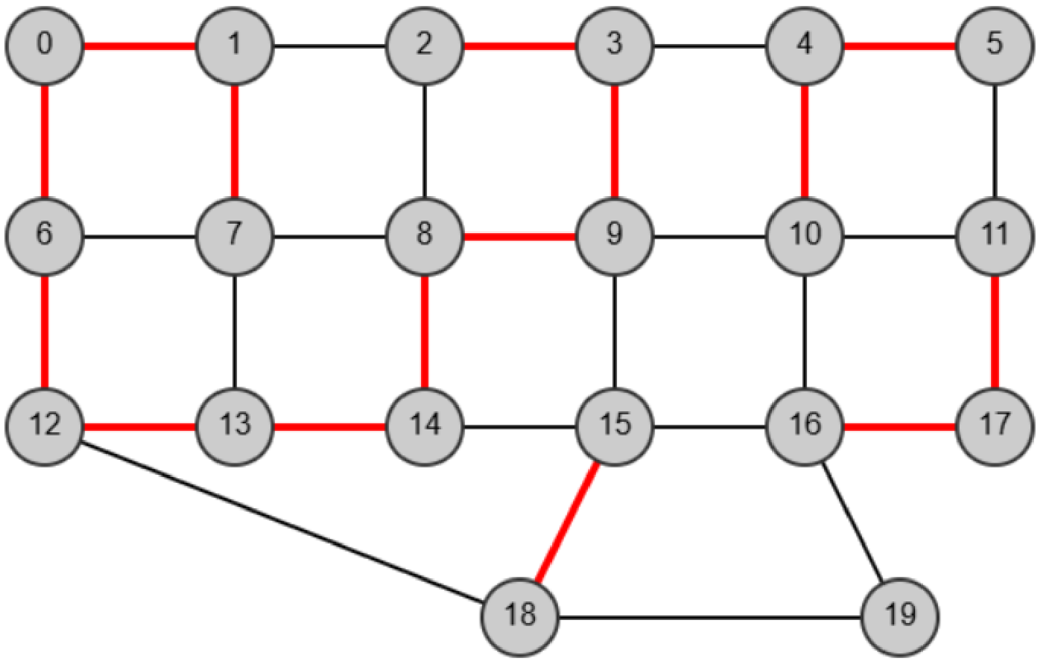

Figure 2.3: *Kruskal's Algorithm graph*

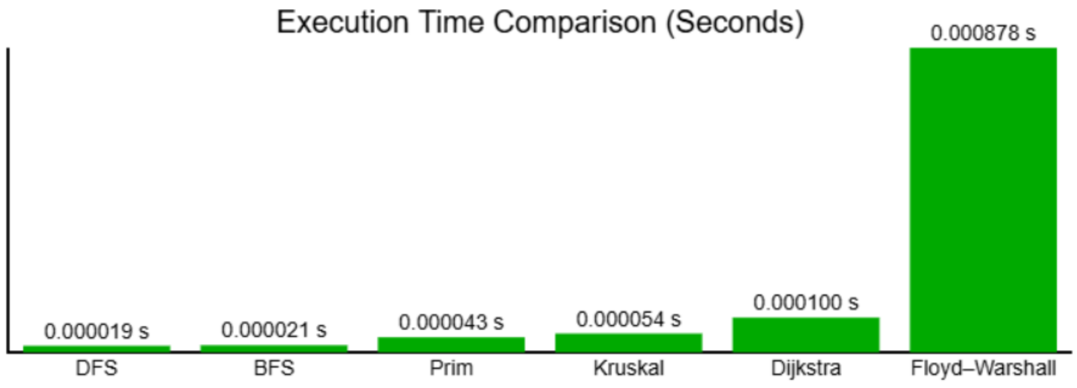

Figure 2.4: *Kruskal's Algorithm performance*

2.2.5 Time Complexity

- Using a disjoint-set: $O(E \log E)$
- Sorting edges: $O(E \log E)$

2.2.6 Auxiliary Space

The auxiliary space complexity of Kruskal's algorithm is $O(V + E)$ due to the disjoint-set operations.

3 Conclusion

This paper has examined the concept of greedy algorithms as a method for solving optimization problems by making locally optimal choices at each step with the goal of finding a globally optimal solution. Specifically, Prim's and Kruskal's algorithms were analyzed in the context of minimum spanning tree computation, highlighting their efficiency, time complexity, and practical applications.

Prim's Algorithm

Prim's algorithm is a greedy approach that incrementally constructs the minimum spanning tree (MST) by adding the smallest edge that connects a new vertex to the existing tree. With a time complexity of $O(E \log V)$ when implemented using a priority queue, it is highly efficient for dense graphs. Prim's algorithm is widely used in network design, circuit design, and clustering applications. However, its dependence on priority queue operations makes it less suitable for extremely sparse graphs when compared to Kruskal's algorithm.

Kruskal's Algorithm

Kruskal's algorithm sorts all edges and adds the smallest one to the MST while ensuring no cycles are formed. Its time complexity of $O(E \log E)$ makes it well-suited for sparse graphs. Kruskal's algorithm is commonly used in network connectivity analysis, transportation systems, and image segmentation. However, for dense graphs with many edges, the sorting step can become a limiting factor in performance.

Summary

Each algorithm has distinct advantages and trade-offs:

- **Prim's Algorithm**: Best suited for dense graphs where adjacency lists and priority queues allow for efficient MST construction.
- **Kruskal's Algorithm**: More effective for sparse graphs where sorting edges first reduces the number of necessary operations.

Future research could explore optimizations that combine aspects of both algorithms, such as using hybrid approaches for large-scale graphs or integrating parallel computing to improve execution time. Additionally, further empirical analyses could examine the impact of different graph structures and weight distributions on algorithm performance.

https://github.com/vvtttvv/AA/tree/main/lab3_4_5