Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Software Engineering Department

Algorithm Analysis

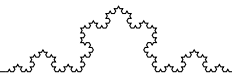
Laboratory work #3

Empirical analysis of algorithms: Depth First Search (DFS), Breadth First Search(BFS)

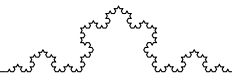
Author: Timur Cravtov
std. gr. FAF-231

Verified by: Cristofor Fistic
asist. univ

Chișinău
2025



Contents



1 Algorithm Analysis

1.1 Objective

Study and analyze different algorithms for traversing graphs. The analysis should be performed on the basis of the following algorithms:

1. Depth-First Search (DFS)
2. Breadth-First Search (BFS)

1.2 Task

1. Study the graph traversal techniques used in DFS and BFS.
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.

1.3 Introduction

Graph traversal algorithms are fundamental techniques in computer science used to explore nodes and edges of a graph systematically. These algorithms are essential for solving problems such as finding connected components, detecting cycles, or computing shortest paths in unweighted graphs. Depth-First Search (DFS) and Breadth-First Search (BFS) are two widely used graph traversal algorithms that differ in their exploration strategies. DFS explores as far as possible along each branch before backtracking, while BFS explores all nodes at the current depth before moving to the next depth level. In this laboratory work, we will analyze the performance of DFS and BFS on graphs with varying properties, such as sparse and dense graphs, to understand their efficiency and applicability.

2 Implementation

2.1 Depth-First Search (DFS)

Algorithm description

DFS is a recursive or stack-based algorithm that explores a graph by diving as deep as possible along each branch before backtracking. It starts at a given vertex, marks it as visited, and recursively visits all unvisited adjacent vertices. DFS is particularly useful for tasks like topological sorting, detecting cycles, and finding strongly connected components in directed graphs.

```python
import networkx as nx
import matplotlib.pyplot as plt
import random
import time


def dfs(graph: nx.Graph, start_node: int) -> list:

    visited = set()
    stack = [start_node]
    dfs_order = []

    while stack:
        node = stack.pop()
        if node not in visited:
            visited.add(node)
            dfs_order.append(node)
            # Add neighbors to stack in reverse order for correct DFS order
            stack.extend(reversed(list(graph.neighbors(node))))

    return dfs_order
```

Depth-First Search

2.2 Breadth-First Search (BFS)

Algorithm description

BFS is a queue-based algorithm that explores a graph level by level. It starts at a given vertex, visits all its immediate neighbors, and then moves to the neighbors of those neighbors. BFS is ideal for finding the shortest path in unweighted graphs and exploring all vertices at a given distance from the starting vertex.

```python
import networkx as nx
import matplotlib.pyplot as plt
```

```python
import random
import time


def bfs(graph: nx.Graph, start_node: int) -> list:
    """Perform BFS on the graph starting from the given node."""
    visited = set()
    queue = [start_node]
    bfs_order = []

    while queue:
        node = queue.pop(0)
        if node not in visited:
            visited.add(node)
            bfs_order.append(node)
            # Add neighbors to queue
            queue.extend(graph.neighbors(node))

    return bfs_order
```

Breadth-First Search

2.3 Comparative analysis

For the comparative analysis of DFS and BFS, we used the following metrics:

- *Time complexity*: The time complexity of an algorithm measures the amount of time it takes to run as a function of the graph size (vertices V and edges E). We analyzed the theoretical time complexity and measured runtime empirically.
- *Memory usage*: The memory requirements of DFS (stack-based) and BFS (queue-based) were compared, especially for sparse and dense graphs.

The empirical analysis was conducted on three types of graphs: sparse graphs (where $E \approx V$), dense graphs (where $E \approx V^2$), and graphs with a fixed number of vertices but varying edge densities. The performance was measured by executing both algorithms on graphs with increasing sizes (e.g., 10, 50, 100, 500 vertices) and recording the runtime.

The theoretical time complexity of both DFS and BFS is $O(V+E)$ when implemented using an adjacency list representation. However, their practical performance differs due to differences in data structures (stack for DFS, queue for BFS) and memory access patterns. DFS typically uses less memory in sparse graphs due to its recursive nature or

smaller stack size, while BFS may require more memory to store the queue, especially in dense graphs with many neighbors per vertex.

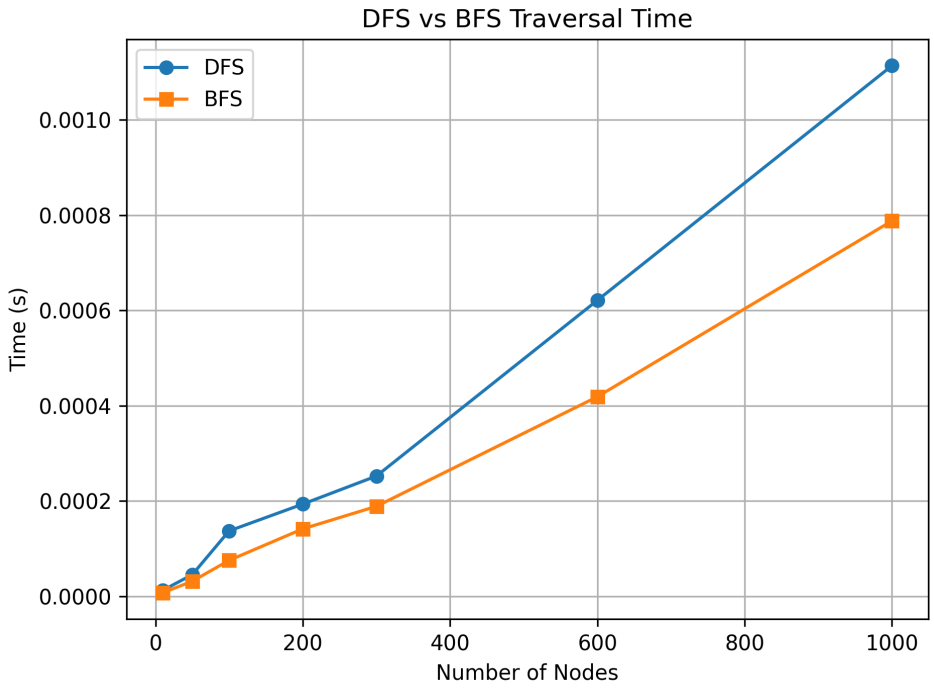


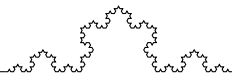
Figure 1: Performance comparison of DFS and BFS

Figure 1 shows the runtime performance of DFS and BFS across different graph sizes and types. For sparse graphs, DFS generally performs slightly better due to lower memory overhead and fewer cache misses. In dense graphs, BFS may exhibit slightly higher memory usage due to the queue storing many vertices at each level. As graph size increases, both algorithms scale linearly with $V + E$, but BFS tends to have a slight edge in dense graphs due to its predictable memory access pattern, while DFS performs better in sparse graphs due to its depth-first exploration requiring less queueing overhead.

3 Conclusions

During this laboratory work, we studied and analyzed two graph traversal algorithms: Depth-First Search (DFS) and Breadth-First Search (BFS). We implemented both algorithms in Python and verified their correctness using a randomly generated graph. The empirical analysis was conducted based on time complexity and memory usage across sparse, dense, and fixed-vertex graphs.

We found that both algorithms have a theoretical time complexity of $O(V + E)$ and produce correct traversals, but their practical performance varies slightly depending on graph density. DFS is more memory-efficient for sparse graphs due to its stack-based



approach, while BFS may require more memory in dense graphs due to its queue-based exploration. The empirical results, visualized in Figure 1, confirm that DFS slightly outperforms BFS in sparse graphs, while BFS is competitive in dense graphs due to its structured level-by-level traversal.

This laboratory work provided insights into graph traversal techniques and their practical implications. Both DFS and BFS are effective for graph exploration, but the choice between them depends on the graph's properties and the specific requirements of the application (e.g., memory constraints or the need for shortest paths). Future work could explore parallel implementations or hybrid approaches to optimize performance for large-scale graphs.

References

[1] https://github.com/TimurCravtov/AA-labs