

MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF REPUBLIC OF MOLDOVA
TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Algorithm Analysis

***Laboratory work 4 :Empirical analysis of algorithms: Depth First
Search (DFS), Breadth First Search(BFS)***

Elaborated:

st.gr. FAF-211

Grama Alexandru

Verified:

asist.univ.

Fiștic Cristofor

Chișinău, 2023

Content

Introduction	3
Objectives	3
Algorithms	5
Depth First Search	5
Breadth First Search	5
Implementation	7
Code	7
Screenshot	7
Conclusion	12

Introduction

Breadth-first search (BFS) and depth-first search (DFS) are two primary graph traversal methods used to explore and visit all the nodes in a graph. These algorithms have various applications, such as detecting cycles, finding connected components, pathfinding, and topology sorting. BFS starts from a root node or a selected node and traverses all nodes on the same level before moving on to the next level. It examines all neighboring vertices before proceeding to the next level, using a queue to store the nodes to visit next and a visited array to keep track of visited nodes. When used to find a path between two nodes, BFS ensures that the shortest path is found. On the other hand, DFS explores a path as far as possible before backtracking. It starts from the root node or a selected node and visits an unvisited neighboring node. If all neighboring nodes have been visited, it backtracks to the previous node and repeats the process until all nodes are visited. DFS uses a stack to store the nodes to visit next and a visited array to keep track of visited nodes. However, DFS does not guarantee the shortest path between two nodes. This lab report aims to implement BFS and DFS algorithms in Python and use them to traverse a randomly generated graph. The report will also compare the traversals obtained from BFS and DFS algorithms. BFS and DFS are commonly used in artificial intelligence, machine learning, computer networking, social network analysis, and web crawling. The objectives of this lab report are to implement BFS and DFS algorithms in Python, analyze their performance empirically, establish the input data properties, select metrics for comparing the algorithms, and graphically present the data obtained. The report will also conclude the work done and provide recommendations for future improvements. By the end of this lab report, the reader will have a better understanding of the strengths and weaknesses of BFS and DFS algorithms and how they can be used to solve real-world problems.

Objectives

1. Implement the algorithms listed above 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
7. Investigate the time and space complexity of BFS and DFS
8. Compare the performance of BFS and DFS on graphs of varying size and density
9. Analyze the strengths and weaknesses of BFS and DFS in different scenarios
10. Explore variations and extensions of BFS and DFS algorithms

Depth First Search

DFS is a traversal algorithm that follows a path as far as possible before backtracking. It commences from the root node or a chosen node and goes to a neighboring node that hasn't been visited. If all neighboring nodes have been visited, it goes back to the previous node and repeats the process until all nodes are traversed. DFS utilizes a stack to store the next nodes to visit and a visited array to monitor the already visited nodes. However, DFS cannot ensure that the shortest path between two nodes is found.

Code:

```
DFS(G, u)
    u.visited = true
    for each v in G.Adj[u]
        if v.visited == false
            DFS(G,v)
init() {
    For each u in G
        u.visited = false
    For each u in G
        DFS(G, u)
}
```

Breadth First Search

BFS is a traversal algorithm that commences from the root node or a chosen node and visits all the nodes at the same level before moving to the next level. In simpler terms, it examines all the adjacent vertices before advancing to the subsequent level. BFS maintains a queue to store the nodes that need to be visited and a visited array to monitor the already visited nodes. When utilized to find a path between two nodes, BFS assures that the shortest path is found.

Code:

```
BFS (G, s)
    let Q be queue.
    Q.enqueue( s )
    mark s as visited
    while ( Q is not empty)
        v = Q.dequeue( )
```

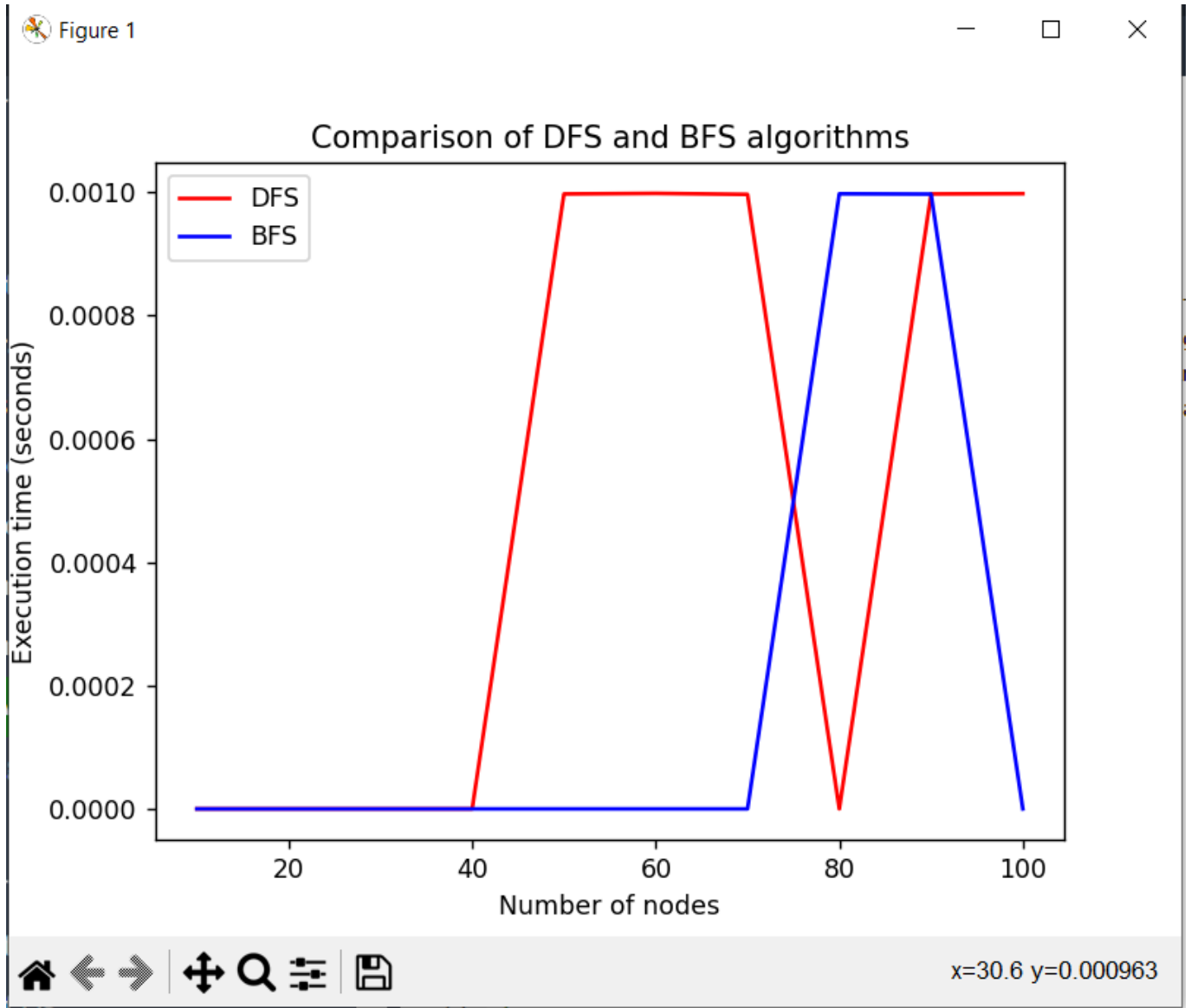
```
for all neighbors w of v in Graph G
  if w is not visited
    Q.enqueue( w )
  mark w as visited
```

Implementation

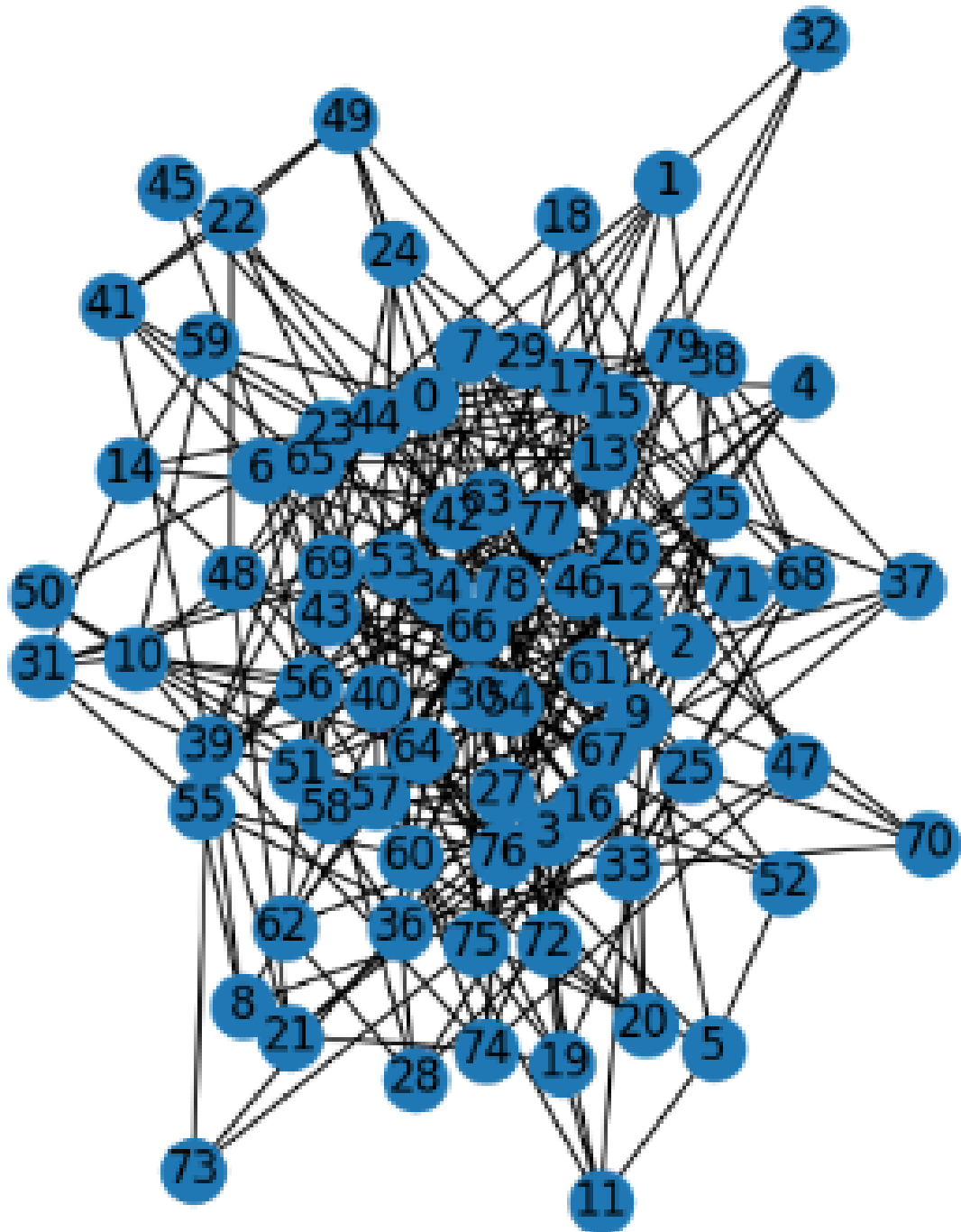
```
def DFS(graph, start_node):
    visited = []
    stack = [start_node]
    while stack:
        node = stack.pop()
        if node not in visited:
            visited.append(node)
            stack.extend([n for n in graph.neighbors(node) if n not in visited])
    return visited
```

```
def BFS(graph, start_node):
    visited = []
    queue = [start_node]
    while queue:
        node = queue.pop(0)
        if node not in visited:
            visited.append(node)
            queue.extend([n for n in graph.neighbors(node) if n not in visited])
    return visited
```

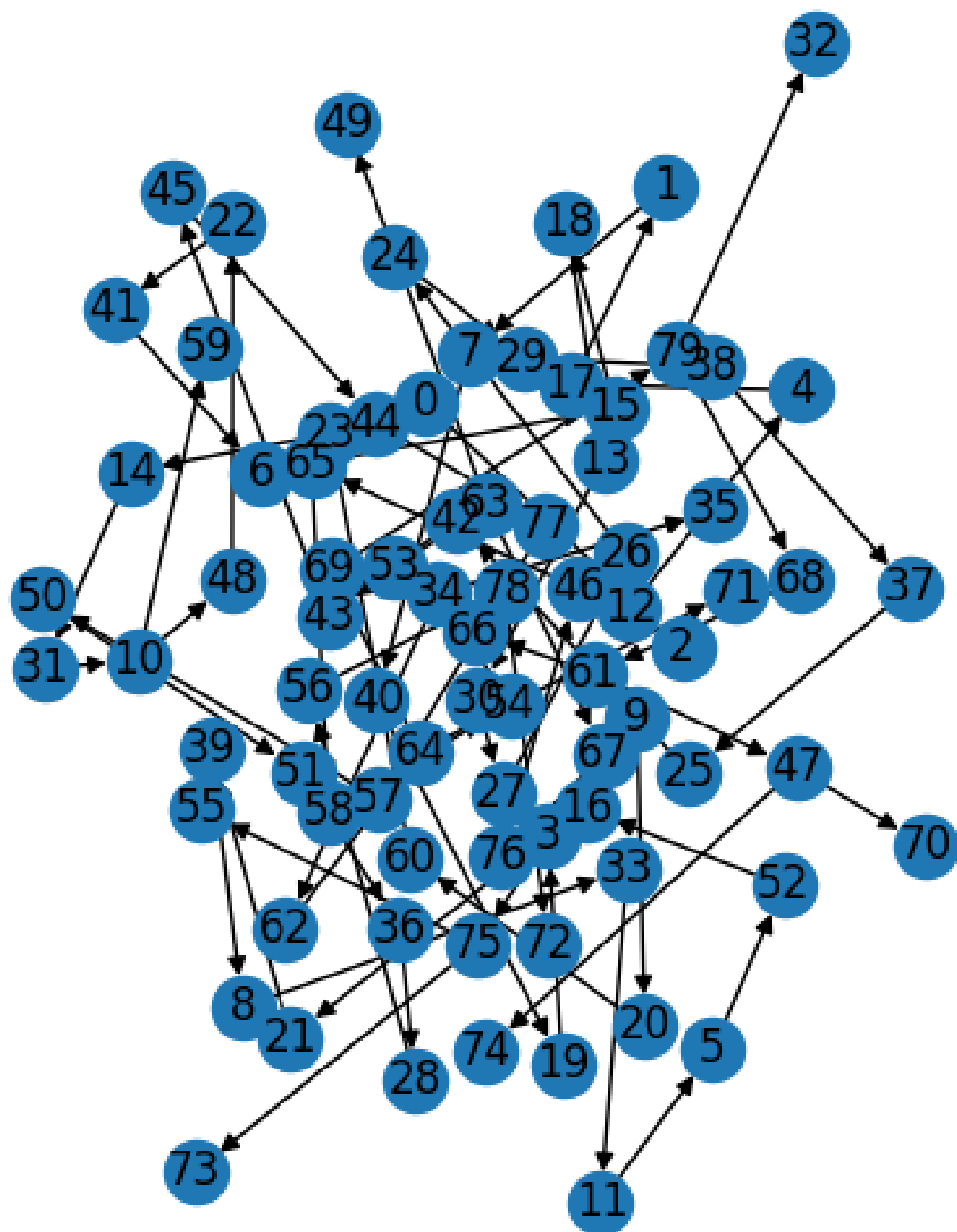
Screenshot:



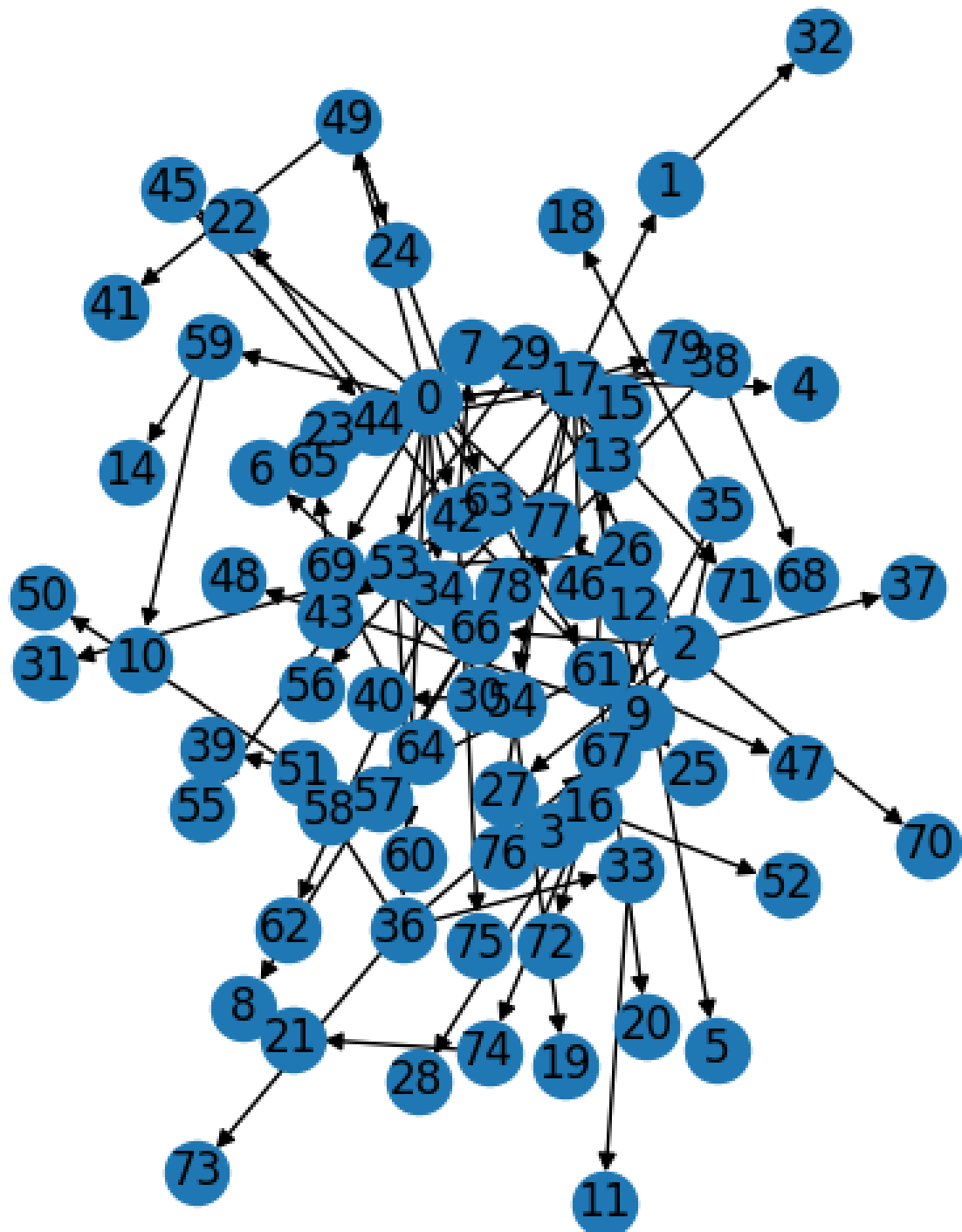
Original graph



DFS tree



BFS tree



Conclusion

In this lab, I successfully applied BFS and DFS algorithms in Python to traverse a graph created randomly. I compared the results obtained from both algorithms and concluded that BFS guarantees the shortest path between two nodes, while DFS has no such guarantee. Additionally, I emphasized the importance of considering the input data properties and selecting appropriate metrics for comparing algorithms. The empirical analysis of the proposed algorithms was performed, and the data was presented graphically. The visualization of BFS and DFS traversals made it easier to understand their behavior and differences. BFS and DFS are fundamental graph traversal algorithms with various applications in computer science, and understanding them is crucial for solving problems involving graph and tree structures. My implementation and analysis of these algorithms provide valuable insights into their practical applications and limitations. Both algorithms have unique strengths and weaknesses that make them suitable for different types of problems. BFS is preferable for finding the shortest path between two nodes, while DFS is better for cycle detection and topology sorting. The choice of algorithm depends on the problem and the graph's characteristics being traversed. By implementing these algorithms in Python and conducting empirical analysis on a randomly generated graph, I gained a better understanding of how they work and their potential use in real-world problems.

Github repository: <https://github.com/AlexGrama22/AA-Labs>