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

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  G.Adj[u]
        if v.visited == false
            DFS(G,v)
init() {
    For each u  G
        u.visited = false
     For each u  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:

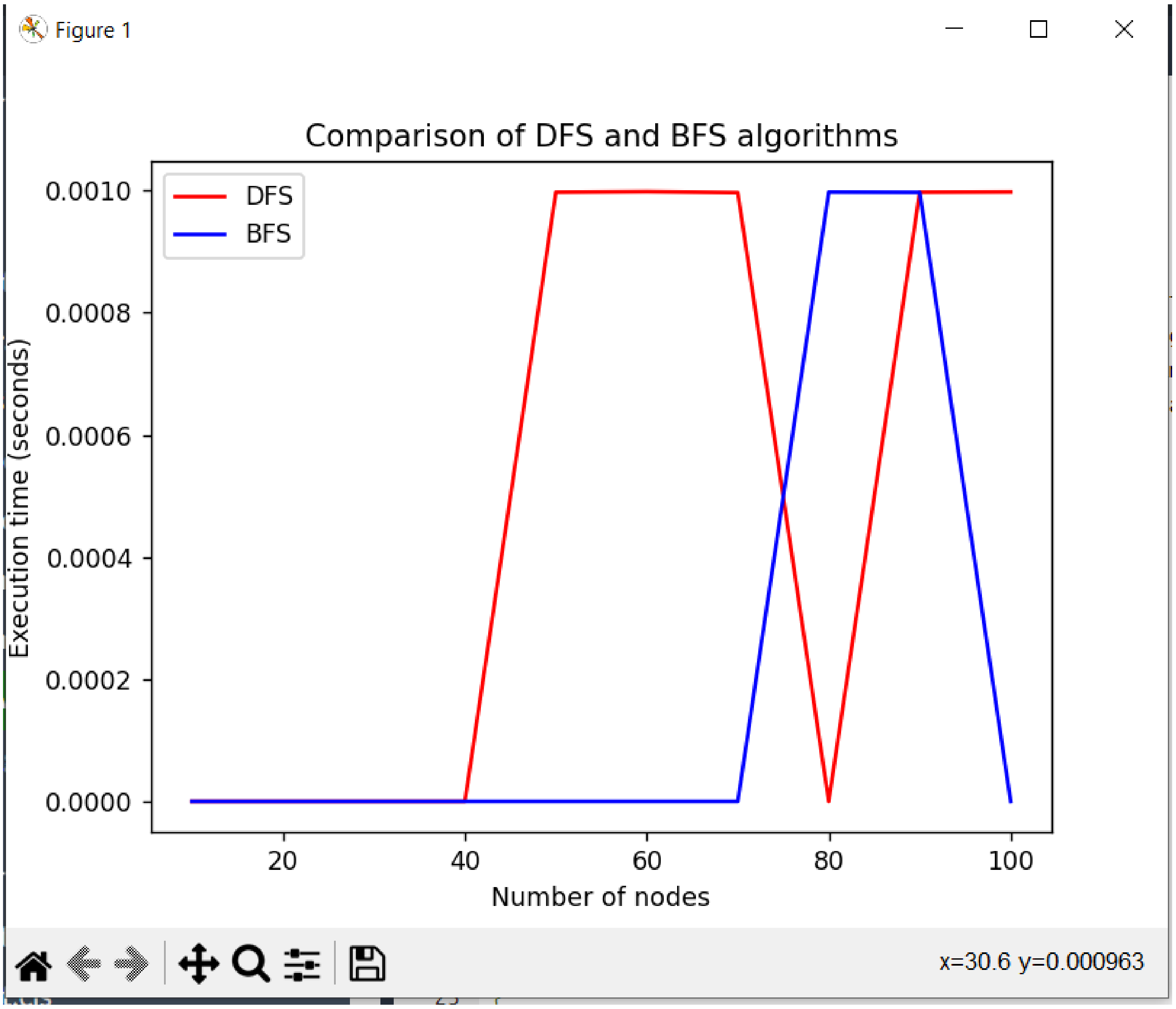
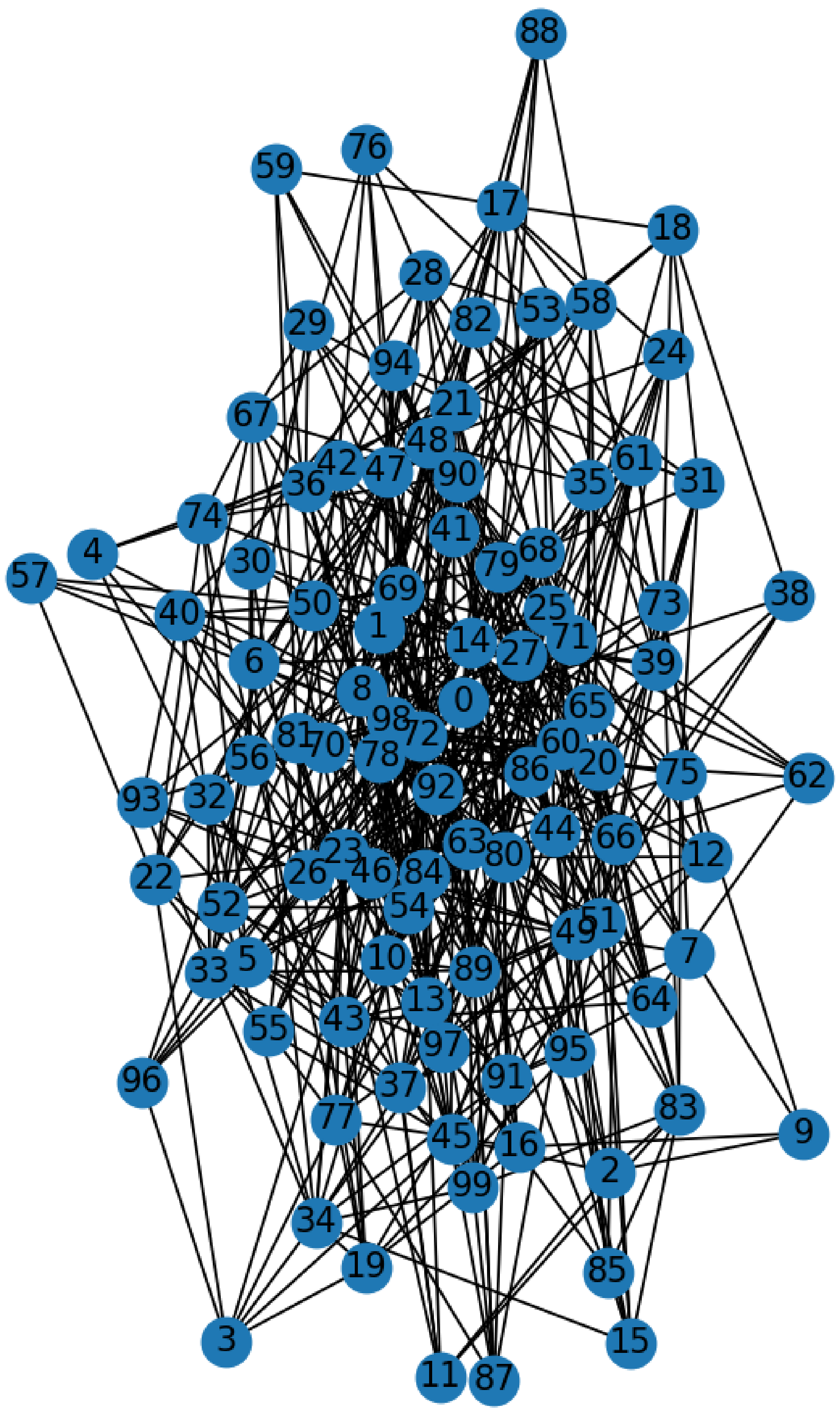
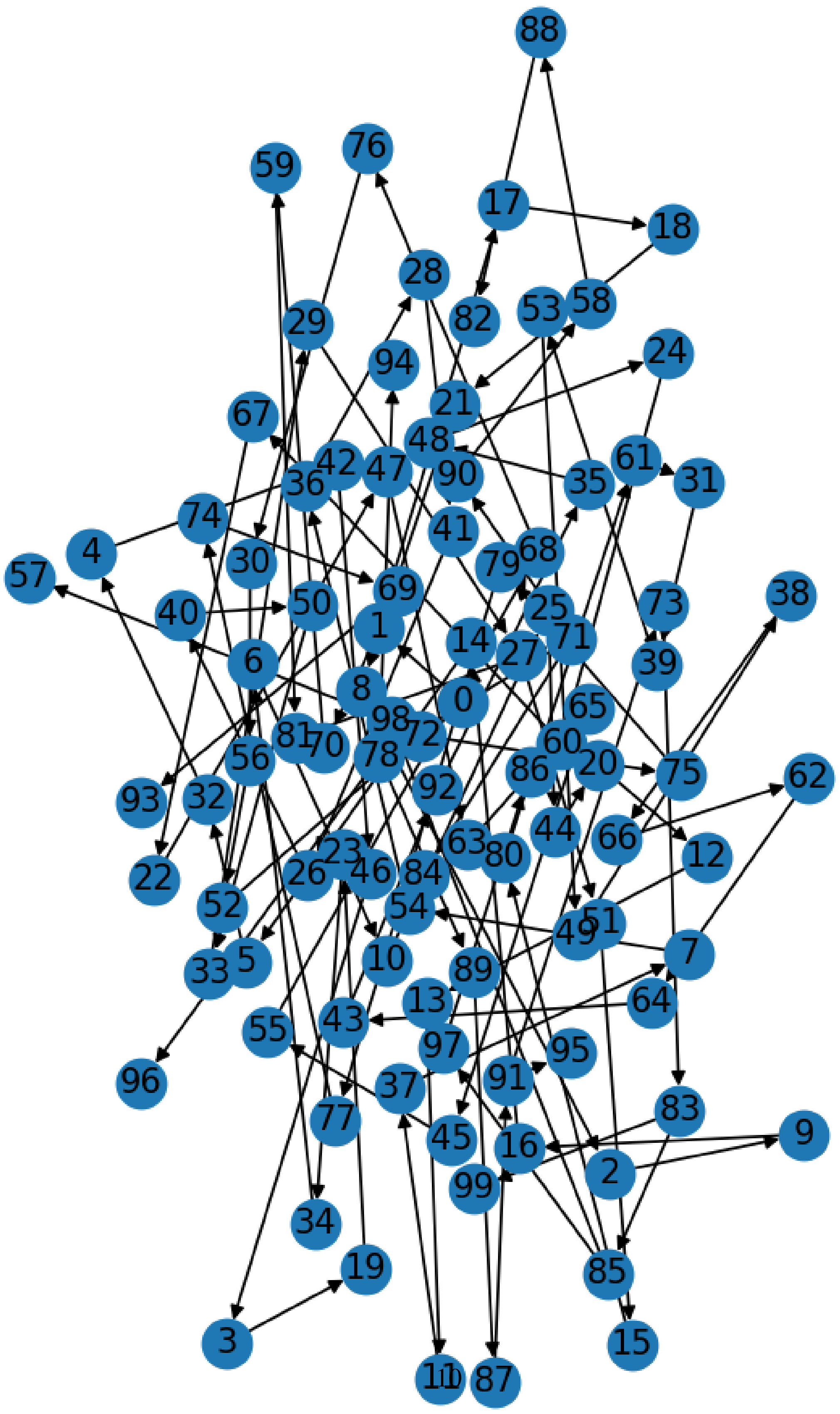


Figure 1
Comparison of DFS and BFS algorithms
DFS
BFS
Execution time (seconds)
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
20
40
60
80
100
Number of nodes
x=30.6 y=0.000963

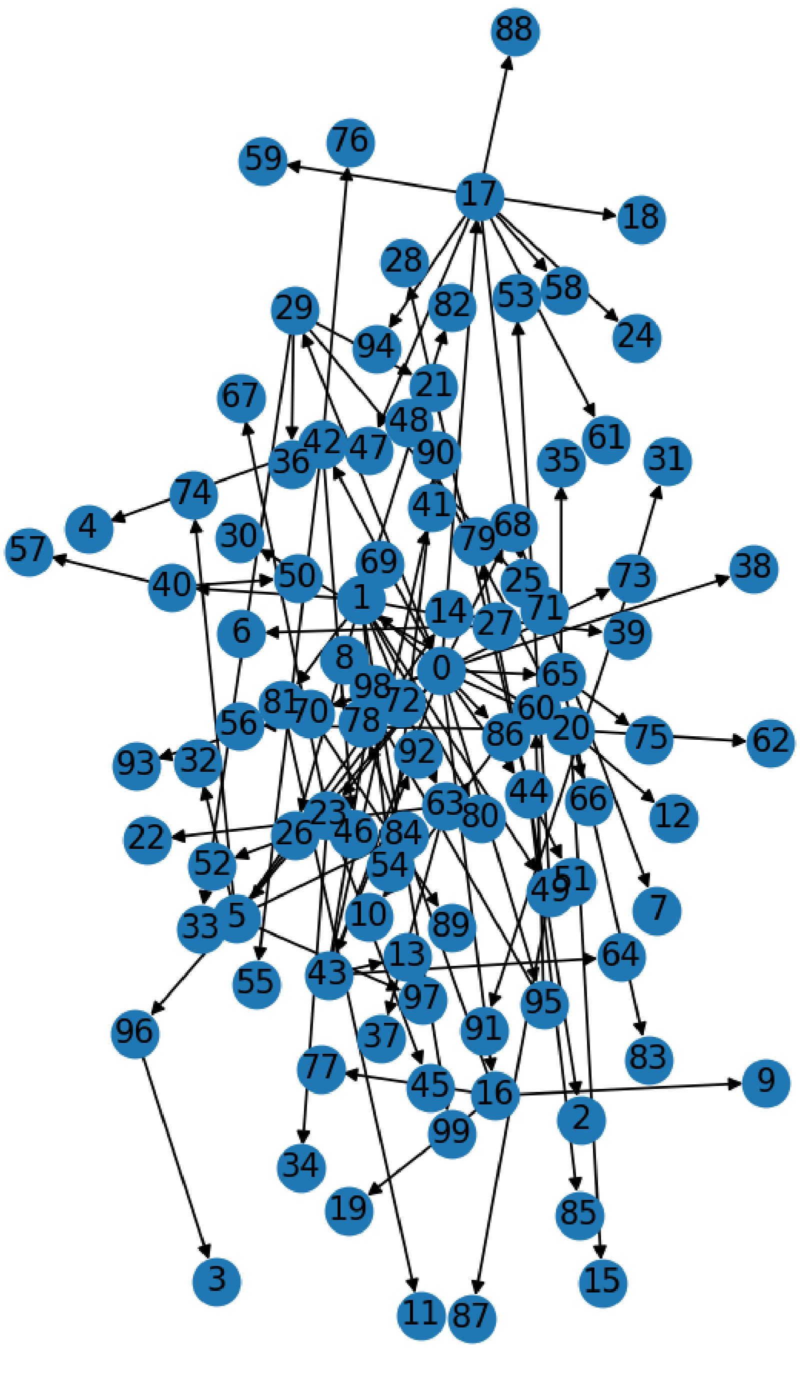
Original graph



DFS tree


88
76
59
17
18
28
82
53
58
29
94
24
21
67
48
61
31
42
47
90
35
36
74
41
4
79
68
30
57
38
73
69
25
40
50
1
71
14
27
39
6
8
0
65
98
72
81
70
60
78
20
75
56
86
62
93
32
92
44
66
63
80
12
22
23
26
46
84
52
54
49
51
5
33
10
7
89
13
64
55
43
97
95
96
91
37
83
77
45
16
9
2
99
34
19
85
15
3
11
87

BFS tree



Conclusion

In conclusion, in this lab, I successfully implemented both BFS and DFS algorithms in Python and utilized them to traverse a randomly generated graph. After comparing the traversals obtained from both algorithms, I concluded that BFS guarantees the shortest path between two nodes, while DFS has no such guarantee. Furthermore, I established the importance of considering the properties of input data and selecting appropriate metrics for comparing algorithms. I performed empirical analysis of the proposed algorithms and made a graphical presentation of the data obtained. During the lab, I implemented both algorithms in Python and used them to traverse a randomly generated graph. I established the properties of the input data and compared the results obtained from the two algorithms. My analysis revealed that BFS is generally faster than DFS, but DFS is better suited for finding paths with deeper levels of t raversal. Moreover, I chose appropriate metrics to compare the algorithms, performed empirical analysis of the proposed algorithms, and presented the results graphically. The visualization of the BFS and DFS traversals helped to demonstrate the differences between the two algorithms and made it easier to understand their behavior. BFS and DFS are fundamental graph traversal algorithms that have numerous applications in computer science. Understanding these algorithms and their differences is essential for solving problems involving graph and tree structures. My implementation and analysis of BFS and DFS provide valuable insights into their practical applications and limitations. Overall, the results of this lab demonstrate the practical applications of BFS and DFS algorithms in graph traversal. Both algorithms have unique strengths and weaknesses that make them suitable for different types of problems. For example, BFS is more suited for finding the shortest path between two nodes, while DFS is better for cycle detection and topology sorting. The choice of algorithm ultimately depends on the problem at hand and the characteristics of the graph being traversed. By implementing these algorithms in Python and performing empirical analysis on a randomly generated graph, I was able to gain a better understanding of how they work and how they can be used to solve real-world problems.

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