UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 12

Graph Searching Algorithm

Submitted by:
Talagtag Mark Angel T.

Instructor:
Engr. Maria Rizette H. Sayo

October 25, 2025

I. Objectives

Introduction

Depth-First Search (DFS)

- Explores as far as possible along each branch before backtracking
- Uses stack data structure (either explicitly or via recursion)
- Time Complexity: O(V + E)
- Space Complexity: O(V)

Breadth-First Search (BFS)

- Explores all neighbors at current depth before moving deeper
- Uses queue data structure
- Time Complexity: O(V + E)
- Space Complexity: O(V)

This laboratory activity aims to implement the principles and techniques in:

- Understand and implement Depth-First Search (DFS) and Breadth-First Search (BFS) algorithms
- Compare the traversal order and behavior of both algorithms
- Analyze time and space complexity differences

II. Methods

- Copy and run the Python source codes.
- If there is an algorithm error/s, debug the source codes.
- Save these source codes to your GitHub.
- Show the output

1. Graph Implementation

```
from collections import deque
import time

class Graph:
    def __init__(self):
        self.adj_list = {}

    def add_vertex(self, vertex):
        if vertex not in self.adj_list:
            self.adj_list[vertex] = []

    def add_edge(self, vertex1, vertex2, directed=False):
        self.add_vertex(vertex1)
        self.add_vertex(vertex2)

        self.adj_list[vertex1].append(vertex2)
        if not directed:
            self.adj_list[vertex2].append(vertex1)
```

```
    def display(self):
        for vertex, neighbors in self.adj_list.items():
            print(f"{vertex}: {neighbors}")
```

2. DFS Implementation

```
def dfs_recursive(graph, start, visited=None, path=None):
    if visited is None:
        visited = set()
    if path is None:
        path = []

    visited.add(start)
    path.append(start)
    print(f"Visiting: {start}")

    for neighbor in graph.adj_list[start]:
        if neighbor not in visited:
            dfs_recursive(graph, neighbor, visited, path)

    return path

def dfs_iterative(graph, start):
    visited = set()
    stack = [start]
    path = []

    print("DFS Iterative Traversal:")
    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            visited.add(vertex)
            path.append(vertex)
            print(f"Visiting: {vertex}")

            # Add neighbors in reverse order for same behavior as recursive
            for neighbor in reversed(graph.adj_list[vertex]):
                if neighbor not in visited:
                    stack.append(neighbor)
    return path
```

3. BFS Implementation

```
def bfs(graph, start):
    visited = set()
    queue = deque([start])
    path = []

    print("BFS Traversal:")
    while queue:
        vertex = queue.popleft()
        if vertex not in visited:
            visited.add(vertex)
            path.append(vertex)
            print(f"Visiting: {vertex}")
```

```
            for neighbor in graph.adj_list[vertex]:
                if neighbor not in visited:
                    queue.append(neighbor)

        return path
```

Questions:

1. When would you prefer DFS over BFS and vice versa?
2. What is the space complexity difference between DFS and BFS?
3. How does the traversal order differ between DFS and BFS?
4. When does DFS recursive fail compared to DFS iterative?

III. Results


```
C:\Users\compu\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\compu\PycharmProjects\pythonProject\soundboard.py
Graph Adjacency List:
A: ['B', 'C']
B: ['A', 'D']
C: ['A', 'E']
D: ['B', 'E']
E: ['C', 'D']

DFS Recursive Traversal:
Visiting: A
Visiting: B
Visiting: D
Visiting: E
Visiting: C
DFS Recursive Path: ['A', 'B', 'D', 'E', 'C']

DFS Iterative Traversal:
Visiting: A
Visiting: B
Visiting: D
Visiting: E
Visiting: C
DFS Iterative Path: ['A', 'B', 'D', 'E', 'C']
```

```
C:\Users\compu\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\compu\PycharmProjects\pythonProject\soundboard.py
Graph Adjacency List:
A: ['B', 'C']
B: ['A', 'D']
C: ['A', 'E']
D: ['B', 'E']
E: ['C', 'D']

DFS Recursive Traversal:
Visiting: A
Visiting: B
Visiting: D
Visiting: E
Visiting: C
DFS Recursive Path: ['A', 'B', 'D', 'E', 'C']

DFS Iterative Traversal:
Visiting: A
Visiting: B
Visiting: D
Visiting: E
Visiting: C
DFS Iterative Path: ['A', 'B', 'D', 'E', 'C']

BFS Traversal:
Visiting: A
Visiting: B
Visiting: C
Visiting: D
Visiting: E
BFS Path: ['A', 'B', 'C', 'D', 'E']

Process finished with exit code 0
```

1. When would you prefer DFS over BFS and vice versa?

You would use Depth-First Search (DFS) when you want to search as deep as you can in a graph before you backtrack. This can be helpful in problems such as cycle detection, topological sorting, visiting connected components, or solving mazes and puzzles in which there can be only one solution path.

Conversely, you would use Breadth-First Search (BFS) if you want to visit all the neighboring nodes level by level. BFS is especially efficient for determining the shortest path in an unweighted graph, doing level-order traversals in trees, or finding all nodes in a given distance from the beginning.

In general, DFS performs better on deep and narrow graphs, whereas BFS performs better for shallow and wide graphs.

2. What is the space complexity difference between DFS and BFS?

The space complexity of DFS is a function of the depth of the graph. If the depth of the graph is h, the recursive or iterative stack used in DFS will take O(h) space. In the worst scenario, like the graph that consists of a long linear chain, the space usage can reach O(V) where V is the number of vertices.

By comparison, BFS takes O(V) space in the worst scenario since it has to store all the nodes at the frontier level currently in a queue. This would imply that for very wide graphs, BFS would use much more memory than DFS.

Hence, if memory is constrained and the graph is deep, DFS is in general space-efficient, while BFS may become memory-bloated on wide, big graphs.

3. How does the traversal order differ between DFS and BFS?

The fundamental distinction in traversal order between DFS and BFS is the manner in which they traverse the graph.

DFS searches depth-first, i.e., it goes along one branch of the graph to the very end before backtracking to look at other branches. This results in a path-based exploration order, e.g., visiting the nodes in the sequence A → B → C →. before reversing direction.

BFS, on the other hand, searches breadth-first, i.e., it visits all neighbors of a node before visiting nodes in the next level. This gives a level-by-level ordering like visiting A → B → C (all immediate neighbors) before visiting their neighbors.

In plain English, DFS goes deep inside the graph, whereas BFS traverses it layer by layer.

4. When does DFS recursive fail compared to DFS iterative?

The recursive form of DFS will break down if the graph is extremely deep, since each recursive call pushes another frame onto the system's call stack. With languages like Python, the recursion limit is usually close to 1000, and if the graph contains a deep chain of nodes beyond that, RecursionError is raised.

The iterative implementation of DFS, on the other hand, does not rely on the call stack of the system. It employs an explicit stack data structure of the program and is thus able to work with very deep graphs without encountering recursion limit faults.

Hence, DFS recursive can fail by stack overflow, whereas DFS iterative can keep working for the same graph.

Conclusion

In short, both Depth-First Search (DFS) and Breadth-First Search (BFS) are basic graph traversal algorithms with different behaviors and applications. DFS travels deeper into a graph before it backtracks, thus being appropriate for use in pathfinding, topological sorting, and cycle detection. BFS, by contrast, visits all neighbor nodes level by level, hence being perfect for the shortest path search in unweighted graphs and for applications involving layer-wise exploration.

As far as efficiency is concerned, both algorithms share the same time complexities of O(V + E), but they vary in space complexities. DFS tends to consume less memory compared to BFS, particularly in broad graphs, although recursive DFS can fail in excessively deep graphs because of recursion depth restrictions. The iterative DFS overcomes this deficiency by maintaining its own stack.

Finally, the decision between DFS and BFS is made according to the graph structure, the purpose of the traversal, and available system resources. It is possible to more effectively solve problems and better optimize implementation in computer engineering tasks knowing the distinction between these algorithms.

References

[1] Co Arthur O.. “University of Caloocan City Computer Engineering Department Honor Code,” UCC-CpE Departmental Policies, 2020.

[2] A. O. Co, “University of Caloocan City Computer Engineering Department Honor Code,” *UCC-CpE Departmental Policies*, 2020.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.

[4] M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, *Data Structures and Algorithms in Python.* Hoboken, NJ: Wiley, 2014.

[5] “Difference Between BFS and DFS,” *GeeksforGeeks*, [Online]. Available: https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/. [Accessed: Oct. 25, 2025].

[6] “sys — System-specific parameters and functions,” *Python Software Foundation Documentation*, [Online]. Available: https://docs.python.org/3/library/sys.html#sys.getrecursionlimit. [Accessed: Oct. 25, 2025].

[7] “BFS vs DFS Algorithms in Graph Theory,” *Baeldung on Computer Science*, 2021. [Online]. Available: https://www.baeldung.com/cs/bfs-vs-dfs. [Accessed: Oct. 25, 2025].