UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 12

Graph Searching Algorithm

Submitted by:
Delinia, Filjohn B.

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

ANSWERS:

1. DFS is preferred when you need to explore a graph deeply, such as in solving puzzles, topological sorting, or exploring all possible paths. It is suitable for deep but narrow graphs. BFS is preferred when you need the shortest path in an unweighted graph or level-order traversal. It works best for shallow or wide graphs where finding the closest solution is important.

2. DFS has a space complexity of **O(d + n)**, where **d** is the depth and **n** is the number of nodes, since it only stores nodes along the current path. BFS has a space complexity of **O(w + n)**, where **w** is the maximum width, because it stores all nodes at the current level in a queue. Thus, BFS generally uses more memory than DFS in wide graphs.

3. DFS explores as far as possible along one branch before backtracking, visiting nodes in depth order. BFS visits all neighbors of a node before moving to the next level, visiting nodes in breadth or level order. In short, DFS goes deep first, while BFS goes wide first.

4. Recursive DFS can fail in very deep graphs because it may exceed Python’s recursion depth limit, causing a stack overflow. Iterative DFS avoids this issue by using an explicit stack, making it more reliable for large or deep graphs.

IV. Conclusion

In summary, both Depth-First Search (DFS) and Breadth-First Search (BFS) are important algorithms used to explore and analyze graphs. They both visit all the vertices in a graph but do so in different ways. DFS goes as deep as possible along one path before backtracking, which makes it useful for problems like solving puzzles, finding connected components, or performing topological sorting. BFS, on the other hand, explores nodes level by level, which makes it ideal for finding the shortest path in an unweighted graph or for level-order traversal in trees.
Although both algorithms have the same time complexity of **O(V + E)**, they differ in space usage. DFS usually needs less memory since it only keeps track of nodes along the current path, while BFS can use more memory because it stores all nodes at the current level. However, recursive DFS can fail in very deep graphs due to Python’s recursion limit, which can be avoided by using an iterative version.

Overall, the choice between DFS and BFS depends on the problem. If you need to explore deeply or don’t care about the shortest path, DFS is a good option. If you need the shortest path or want to process nodes by levels, BFS is the better choice. Understanding both algorithms helps in choosing the most efficient method for different types of graph problems.

References

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

[2] *Graph Data Structure*. (n.d.). https://www.tutorialspoint.com/data_structures_algorithms/graph_data_structure.htm

[3] GeeksforGeeks. (2025, October 3). *Depth first search or DFS for a graph*. GeeksforGeeks. https://www.geeksforgeeks.org/dsa/depth-first-search-or-dfs-for-a-graph/