UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 12

Graph Searching Algorithm

Submitted by:
Barbas, Steven Jade P.

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

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

- I would prefer BFS over DFS because BFS is perfect when you're looking for the shortest route between two points in a graph. It explores all neighbors first, guaranteeing it finds the minimal number of steps.

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

- The space complexity difference between those two is BFS must store all nodes at the current depth level before moving deeper, which can require a lot of memory for wide graphs. In contrast, DFS only needs to remember the nodes along the single path it is currently exploring, using much less memory.

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

- The traversal order differs fundamentally in how they explore nodes. **DFS** plunges deep down one branch to the end before backtracking, following a path as far as possible. In contrast, **BFS** expands radially, visiting all neighbors at the present depth level before moving to the next level.

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

- DFS recursive crashes if the graph is too deep because it runs out of memory. DFS iterative works fine for deep graphs because it uses regular memory instead. Always use iterative for very large graphs to avoid crashes.

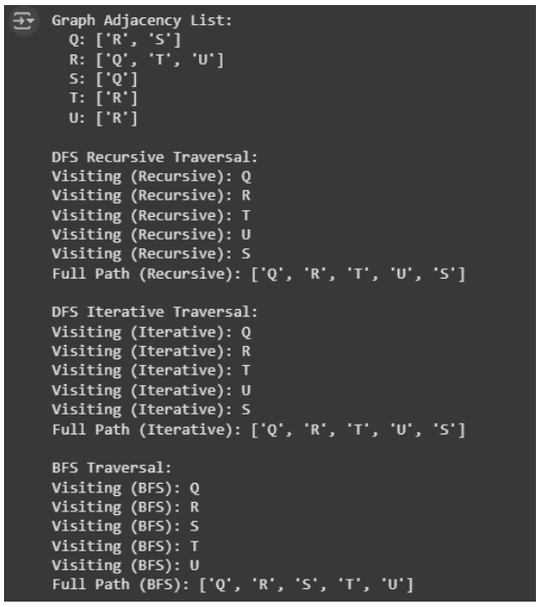

Figure 1 Screenshot of output of the program

IV. Conclusion

In conclusion, this lab successfully implemented and tested graph search algorithm using DFS and BFS. We learned that BFS is better for finding shortest paths and exploring level by level, while DFS is more memory-efficient for deep graphs. The main difference is that BFS spreads out widely, while DFS goes deep first. For very large graphs, iterative DFS is safer than recursive DFS to avoid crashes.

References

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

[2] GeeksforGeeks. (2025, July 23). *When to use DFS or BFS to solve a Graph problem?* GeeksforGeeks. https://www.geeksforgeeks.org/dsa/when-to-use-dfs-or-bfs-to-solve-a-graph-problem/

[3] GeeksforGeeks. (2025a, July 11). *Difference between BFS and DFS*. GeeksforGeeks. https://www.geeksforgeeks.org/dsa/difference-between-bfs-and-dfs/

[4] Sauce, T. (2024, November 5). *Time Complexity and Space Complexity of DFS and BFS Algorithms*. Medium. https://techsauce.medium.com/time-complexity-and-space-complexity-of-dfs-and-bfs-algorithms-671217e43d58

[5] *DFS Recursive vs DFS Iterative*. (2014, November 20). Stack Overflow. https://stackoverflow.com/questions/27033429/dfs-recursive-vs-dfs-iterative