UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 12

Graph Searching Algorithm

Submitted by:
Balaoro, Judge Wayne 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

```
Graph Adjacency List:
  A: ['B', 'C']
  B: ['A', 'D', 'E']
  C: ['A', 'F']
  D: ['B']
  E: ['B', 'F']
  F: ['C', 'E']

DFS Recursive Traversal:
Visiting (Recursive): A
Visiting (Recursive): B
Visiting (Recursive): D
Visiting (Recursive): E
Visiting (Recursive): F
Visiting (Recursive): C
Full Path (Recursive): ['A', 'B', 'D', 'E', 'F', 'C']

DFS Iterative Traversal:
Visiting (Iterative): A
Visiting (Iterative): B
Visiting (Iterative): D
Visiting (Iterative): E
Visiting (Iterative): F
Visiting (Iterative): C
Full Path (Iterative): ['A', 'B', 'D', 'E', 'F', 'C']

BFS Traversal:
Visiting (BFS): A
Visiting (BFS): B
Visiting (BFS): C
Visiting (BFS): D
Visiting (BFS): E
Visiting (BFS): F
Full Path (BFS): ['A', 'B', 'C', 'D', 'E', 'F']
```

Figure 1: Program Output

1. I would prefer DFS over BFS, when I need to explore a single path as far as possible before trying another path just like how you solve a maze. I would use BFS when i want to find a shortest path just like how we see on our suggestions on friends that are the

"friends of our friends" on social networking apps like Facebook, because it checks all nearby nodes first, layer by layer.

2. The difference is that BFS can use a lot of memory for wide graphs because its queue will have to hold many nodes at the same level. While the DFS can use a lot of memory for deep graphs because it uses a stack to store the entire long path it's currently storing.

3. DFS uses a stack that uses the LIFO (Last-In, First-Out) to dive deeper which means it explores one branch completely before backtracking. On the other hand, BFS uses a queue that implements FIFO (First-In, First-Out) to explore in layers visiting all neighbors of a node before moving to the next level.

4. The recursive DFS can fail with Stack Overflow Error if the graph is so deep. This can happen because each of those functions adds to the system's limited stack. To avoid this problem we use iterative DFS by using its own stack data structure, which is stored on a much larger system and won't overflow just because a path is too long.

IV. Conclusion

This laboratory activity provided me a practical understanding of the two graph searching algorithms which are Depth-First Search (DFS) and Breadth-First Search (BFS). By implementing both a recursive and an iterative of DFS. I was able to compare their limitations and I learned how DFS uses a stack to dive deep into one branch which can lead to stack overflow error in its recursive form. On the other hand, BFS uses a queue to explore layer by layer making it ideal for finding the shortest path. This experience makes me think of where one is preferred over the other.

References

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

[2] *DFS vs BFS Algorithm (All Differences With Example)*. (n.d.). WsCube Tech. https://www.wscubetech.com/resources/dsa/dfs-vs-bfs

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