UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 12

Graph Searching Algorithm

Submitted by:
Sorellano, John Kenneth 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

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

- DFS (depth first search) is good for solving some puzzle like sorting issues and good for exploring and finding deep pathways of data. In vice versa BFS (breadth first search) scans the data level by level or layer by layer starting from the parent node it's also better if you want the shortest path in unweighted graphs.

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

- Depth first Search only stores nodes along the current path (O(h), where h is the depth), this usually utilizes and use less memory. Storing every node at the current level (O(w), where w is the maximum width), Breadth first search needs a lot more memory.

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

- Depth first search explores as far as possible along a branch of data before backtracking in each of it and diving deep into the graph. While the breadth first search explores all neighbors of a node before moving to the next level, visiting nodes layer by layer.

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

- The depth first search fails over a stack overflow from too many recursive calls, it can cause DFS recursive to fail. DFS iterative is safer for larger graphs or deep graphs since it can use an explicit stack to avoid this issue.

Conclusion

In conclusion, this graph traversal methods DFS and BFS have unique benefits, one is the best for determining the shortest path in unweighted graphs (BFS), and the (DFS) is better suited for deep exploration that requires backtracking. Although stack overflow might cause DFS recursion to fail with deep graphs, DFS's space complexity is typically lower than BFS's. The problem will show which of DFS and BFS is best, as each approach performs well in different situations. Choosing the best algorithm for the job at hand requires an understanding between its traversal orders, space complexity, and application domains.

References

- GeeksforGeeks. (n.d.). *Depth First Search (DFS) for a graph*. GeeksforGeeks. Retrieved October 25, 2025, from https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
- GeeksforGeeks. (n.d.). *Breadth First Search (BFS) for a graph*. GeeksforGeeks. Retrieved October 25, 2025, from https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
- Wikipedia contributors. (2021, May 19). *Comparison of depth-first and breadth-first search*. Wikipedia. Retrieved October 25, 2025, from https://en.wikipedia.org/wiki/Comparison_of_depth-first_and_breadth-first_search