UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 12

Graph Searching Algorithm

Submitted by:
Ruperto, April Anne A.

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. Depth-first search (DFS) is preferably used if we want to explore as deeply as possible into a graph before backtracking. Breadth-first search (BFS) is preferably used if we want to explore all neighbors at the present depth level before moving n to nodes at the next depth level.
2. DFS used less space because it doesn't need to store all nodes at once. For recursive DFS, the space is consumed by the stack. For iterative DFS, space complexity depends on the number of nodes at the deepest level. BFS can be more memory-efficient, but its queue might need more space to hold all nodes at a given level. It mainly stores the nodes in the queue.
3. DFS explores as deeply as possible before backtracking while BFS explores level by level, visiting all the nodes at the current level before moving on to the next level.
4. Recursive DFS may fail if the graph is extremely deep because the system's call stack can exceed the maximum limit. This is a common problem with deep recursions in languages like Python, where the default recursion depth limit can be reached. Iterative DFS used an explicit stack so it can handle deeper graphs without hitting a stack overflow.

IV. Conclusion

In conclusion, depth-first search (DFS) and breadth-first search (BFS) are fundamental graph transversal algorithms. DFS is typically preferred when exploring deep/large graphs, it excels in problems requiring exploration like solving puzzles, mazes. However, DFS struggles with stack overflow in recursive implementation if the graph is too deep. While BFS is used for problems where shortest path discovery or level-order transversal is required. It explores nodes layer by layer, but at the cost of potentially high memory usage due to the widespread at each level.

References

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