Data Structure and Algorithm

Laboratory Activity No. 12

Graph Searching Algorithm

|  |  |
| --- | --- |
| *Submitted by:* | *Instructor:* |
| Nerio, Hannah Grace A. | Engr. Maria Rizette H. Sayo |

October 25, 2025

# 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

# 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

1. 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?

# Results

Questions:

1. When would you prefer DFS over BFS and vice versa?
   * Depth-First Search (DFS) is preferred when you want to explore as far as possible along each branch before backtracking. It is especially useful in applications like maze solving, topological sorting, and detecting cycles because it follows a single path deeply before exploring others. DFS is also suitable when memory is limited, and the graph is not too wide since it only needs to remember nodes along the current path. On the other hand, Breadth-First Search (BFS) is ideal for finding the shortest path in an unweighted graph because it explores all neighboring nodes first before moving to deeper levels. BFS is often used in routing, peer-to-peer networks, and social network analysis where proximity and shortest connections matter. According to Codecademy (2024), BFS guarantees the shortest route between two points, while DFS is more efficient for exploring or searching through deep structures.
2. What is the space complexity difference between DFS and BFS?
   * Although both DFS and BFS have the same time complexity of O (V + E), their space usage differs depending on the graph’s structure. BFS uses a queue that may hold all nodes at the current depth level, which can grow significantly in wide graphs. Therefore, its worst-case space complexity is O(V). DFS, on the other hand, uses either recursion or a stack to keep track of visited vertices, with a space complexity that depends on the maximum depth of the search. In most cases, DFS consumes less memory than BFS in wide graphs but can use more memory if the graph is extremely deep. As explained by GeeksforGeeks (2025), BFS tends to require more memory when the branching factor is large because it stores many nodes at once, while DFS is generally more space-efficient when dealing with narrow or sparse graphs.
3. How does the traversal order differ between DFS and BFS?
   * The main difference between DFS and BFS lies in the order they visit vertices. BFS explores nodes level by level, starting from the source node and visiting all its neighbors before moving to the next layer. This ensures that nodes closer to the start are processed first, making it suitable for finding the shortest path. Meanwhile, DFS explores one branch deeply before backtracking to explore other paths, creating a depth-first visiting pattern. In your code, the bfs () function prints nodes in breadth order using a queue, while the dfs\_recursive () and dfs\_iterative () functions show a depth-first order using a recursive call or a stack. According to FreeCodeCamp (2024), BFS moves outward from the starting point like waves, whereas DFS dives deeper along one direction before resurfacing to explore other branches
4. When does DFS recursive fail compared to DFS iterative?
   * DFS recursive can fail when the graph is extremely deep or complex because it relies on the system’s call stack. In Python, the recursion limit is around 1000 calls, so if the graph depth exceeds this limit, a Recursion Error may occur. This limitation makes DFS recursive less reliable for very large or deeply nested graphs. On the other hand, DFS iterative uses an explicit stack data structure, which allows it to handle larger graphs without running into recursion limits. It is also easier to manage memory and control flow in the iterative version. As stated by Tutorialspoint (2024), recursive DFS is simpler to write but may crash in deep graphs, while iterative DFS is more stable and suitable for larger datasets

# Conclusion

In this laboratory experiment on Graph Searching Algorithms, both Breadth-First Search (BFS) and Depth-First Search (DFS) were successfully implemented and analyzed. The activity demonstrated how both algorithms can traverse a graph completely but differ in their approach—BFS explores level by level while DFS goes deeper before backtracking. Although they share the same time complexity of O (V + E), BFS often uses more memory, and DFS may encounter recursion limits depending on graph depth. This experiment enhanced our understanding of graph traversal and highlighted how choosing the appropriate algorithm depends on the problem’s requirements, such as finding the shortest path or performing a deep exploration of nodes.

**References**

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

[2] Codecademy, “BFS vs DFS,” Codecademy Computer Science Articles, 2024. [Online]. Available: [**https://www.codecademy.com/article/bfs-vs-dfs**](https://www.codecademy.com/article/bfs-vs-dfs)

[3] GeeksforGeeks, “Time and Space Complexity of DFS and BFS Algorithm,” GeeksforGeeks, 2025. [Online]. Available: [**https://www.geeksforgeeks.org/dsa/time-and-space-complexity-of-dfs-and-bfs-algorithm/**](https://www.geeksforgeeks.org/dsa/time-and-space-complexity-of-dfs-and-bfs-algorithm/)

[4] FreeCodeCamp, “Depth First Search vs Breadth First Search Explained,” FreeCodeCamp Tutorials, 2024. [Online]. Available: [**https://www.freecodecamp.org/news/**](https://www.freecodecamp.org/news/)

[5] Tutorialspoint, “Difference between BFS and DFS,” Tutorialspoint Computer Science, 2024. [Online]. Available: [**https://www.tutorialspoint.com/difference-between-bfs-and-dfs**](https://www.tutorialspoint.com/difference-between-bfs-and-dfs)