Ministerul Educației și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work 3:

Empirical analysis of algorithms: Depth First Search (DFS), Breadth First Search (BFS)

Elaborated:
st. gr. FAF-231
Dârzu Cătălin

Verified:
asist. univ.
Fiştic Cristofor

Chişinău - 2025

Contents

1 Algorithm Analysis

1.1 Objective

Study and empirical analysis of graph traversal algorithms. Analysis of Depth First Search (DFS) and Breadth First Search (BFS).

1.2 Tasks

As per the Lab 3 requirements:

1. Implement the algorithms listed above (DFS and BFS) in a programming language.
2. Establish the properties of the input data against which the analysis is performed.
3. Choose metrics for comparing algorithms.
4. Perform empirical analysis of the proposed algorithms.
5. Make a graphical presentation of the data obtained.
6. Make a conclusion on the work done.

2 Graph Traversal Algorithms

Graph traversal is the process of visiting each vertex in a graph. Traversal algorithms may visit vertices in a specific order, or they may explore the graph based on certain properties of its structure. Two fundamental traversal algorithms are Depth First Search (DFS) and Breadth First Search (BFS).

2.1 Depth First Search (DFS)

1. Introduction
Depth First Search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at a selected root node (in the case of a graph, any arbitrary node can be chosen as the root) and explores as far as possible along each branch before backtracking. (Based on information from [Depth-First Search (DFS) — Brilliant Math Science Wiki](https://brilliant.org/wiki/depth-first-search-dfs/) and [Depth-first search - Wikipedia](https://en.wikipedi
first$_search))$

2. Algorithm Description
2.1 Concept DFS explores a path to its end before backtracking and exploring other paths. It uses a stack (implicitly through recursion or explicitly) to keep track of vertices to visit.

- Start from a given source vertex.
- Mark the current vertex as visited.
- For each adjacent unvisited vertex, recursively call DFS.

- If all neighbors are visited, backtrack to the previous vertex and explore other unvisited paths. (Based on information from [What Is DFS (Depth-First Search): Types, Complexity More — Simplilearn](https://www.simplilearn.com/tutorials/data-structure-tutorial/dfs-algorithm))

2.2 Standard Implementation (Recursive Pseudocode)

```
DFS(graph, vertex, visited_set):
    mark vertex as visited
    add vertex to visited_set
    // Process vertex (e.g., print)

    for each neighbor of vertex in graph:
        if neighbor is not in visited_set:
            DFS(graph, neighbor, visited_set)

// Initial call for a graph G starting at vertex S:
// visited = new Set()
// DFS(G, S, visited)
// For disconnected graphs, iterate through all vertices
// and call DFS if not visited.
```

(Conceptual, based on typical DFS implementations discussed in [Depth-first search - Wikipedia](https://en.wi
first$_{s}earch))$
2.3 Iterative Implementation (Conceptual Outline) An iterative version uses an explicit stack:

1. Initialize an empty stack and push the starting vertex onto it.
2. Initialize a set or boolean array to keep track of visited vertices.
3. While the stack is not empty:
   (a) Pop a vertex from the stack.
   (b) If the vertex has not been visited:
      i. Mark it as visited and process it.
      ii. For each unvisited neighbor, push it onto the stack.

(Based on information from [Depth-first search - Wikipedia](https://en.wikipedia.org/wiki/Depth-first$_{s}earch)and[Depth-FirstSearchinPython : TraversingGraphsandTrees|DataCamp](https : //www.datacamp.com/tutorial/depth-first-search-in-python))$
3. Complexity

- **Time Complexity**: $O(V + E)$, where V is the number of vertices and E is the number of edges, because DFS visits every vertex and every edge once in the worst case. (From [Depth-First Search (DFS) — Brilliant Math Science Wiki](https://brilliant.org/wiki/depth-first-search-dfs/), [What Is DFS (Depth-First Search): Types, Complexity More — Simplilearn](https://www.simplilearn.com/tutorials/data-structure-tutorial/dfs-algorithm))

- **Space Complexity**: $O(V)$ in the worst case (e.g., for a path graph) due to the recursion stack depth or the explicit stack size. (From [Depth-first search - Wikipedia](https://en.wikipedia.org/wiki/Depth-first$_search))$

4. Applications

- Detecting cycles in a graph. (From [What Is DFS (Depth-First Search): Types, Complexity More — Simplilearn](https://www.simplilearn.com/tutorials/data-structure-tutorial/dfs-algorithm))
- Topological sorting for Directed Acyclic Graphs (DAGs). (From [Depth-First Search (DFS) — Brilliant Math Science Wiki](https://brilliant.org/wiki/depth-first-search-dfs/))
- Finding connected components in an undirected graph.
- Solving puzzles with a single solution, like mazes. (From [Depth-First Search (DFS) — Brilliant Math Science Wiki](https://brilliant.org/wiki/depth-first-search-dfs/))
- Pathfinding (though not guaranteed to be the shortest path). (From [Depth-First Search in Python: Traversing Graphs and Trees — DataCamp](https://www.datacamp.com/tutorial/depth-first-search-in-python))

5. Graphical Representation

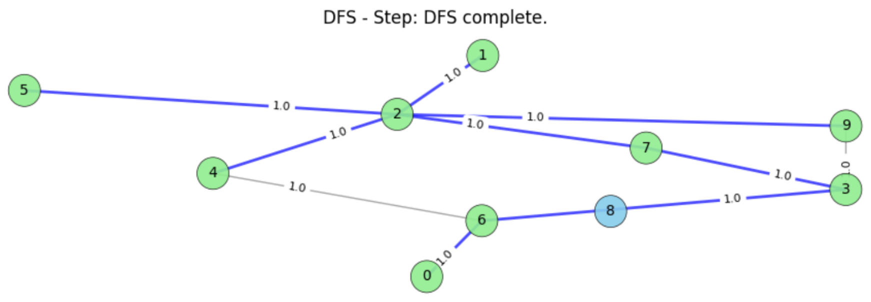

Figure 1: Graphical presentation of Depth First Search (Replace with actual image)

2.2 Breadth First Search (BFS)

1. Introduction Breadth First Search (BFS) is another fundamental algorithm for traversing or searching tree or graph data structures. It starts at a selected root node and explores all the neighbor nodes at the present depth level prior to moving on to the nodes at the next depth level. (Based on [Breadth-first search - Wikipedia](https://en.wikipedia.org/wiki/Breadth-first$_search)and[WhatIsB$ $FinalRoundAI](https://www.finalroundai.com/blog/what-is-breadth-first-search-a-clear-overview-of-the-algorithm))$

2. Algorithm Description

2.1 Concept BFS explores the graph layer by layer. It uses a queue to keep track of vertices to visit.

- Start from a given source vertex and add it to a queue. Mark it as visited.
- While the queue is not empty:

    - Dequeue a vertex from the front of the queue.
    - Process the dequeued vertex.
    - For each unvisited neighbor of the dequeued vertex, mark it as visited and enqueue it.

(Based on [Breadth First Search or BFS for a Graph — GeeksforGeeks](https://www.geeksforgeeks.org/breadt first-search-or-bfs-for-a-graph/))

2.2 Standard Implementation (Pseudocode)

```
BFS(graph, start_vertex):
    queue = new Queue()
    visited_set = new Set()

    queue.enqueue(start_vertex)
    visited_set.add(start_vertex)

    while queue is not empty:
        current_vertex = queue.dequeue()
        // Process current_vertex (e.g., print)

        for each neighbor of current_vertex in graph:
            if neighbor is not in visited_set:
                visited_set.add(neighbor)
                queue.enqueue(neighbor)

// For disconnected graphs, iterate through all vertices
// and call BFS if not visited.
```

(Conceptual, based on typical BFS implementations described in [Breadth-first search - Wikipedia](https://en.w first$_{s}earch$))

3. Complexity

- **Time Complexity**: $O(V+E)$, where V is the number of vertices and E is the number of edges, as each vertex and each edge is explored once in the worst case. (From [Time and Space Complexity of Breadth First Search (BFS) - GeeksforGeeks](https://www.geeksforgeeks.org/time- and-space-complexity-of-breadth-first-search-bfs/))

- **Space Complexity**: $O(V)$ in the worst case (e.g., for a graph where all nodes are at the same level from the source, like a star graph, or when the queue holds all vertices). (From [Time and Space Complexity of Breadth First Search (BFS) - GeeksforGeeks](https://www.geeksforgeek and-space-complexity-of-breadth-first-search-bfs/))

4. Applications

- Finding the shortest path in an unweighted graph. (From [Breadth-First Search (BFS) - CelerData](https://celerdata.com/glossary/breadth-first-search-bfs))

- Cycle detection in an undirected graph (can also be used for directed).

- Finding connected components. (From [Breadth-First Search (BFS) - CelerData](https://celerdata.com/g first-search-bfs))

- Web crawlers to explore web pages level by level. (From [Breadth-First Search (BFS) - CelerData](https://celerdata.com/glossary/breadth-first-search-bfs))
- Used in algorithms like Prim's or Dijkstra's (with modifications for weighted graphs). (Mentioned in [Breadth First Search or BFS for a Graph — GeeksforGeeks](https://www.geeksforgeeks. first-search-or-bfs-for-a-graph/))
- Checking if a graph is bipartite. (From [Breadth-First Search (BFS) - CelerData](https://celerdata.com/gl first-search-bfs))

5. Graphical Representation

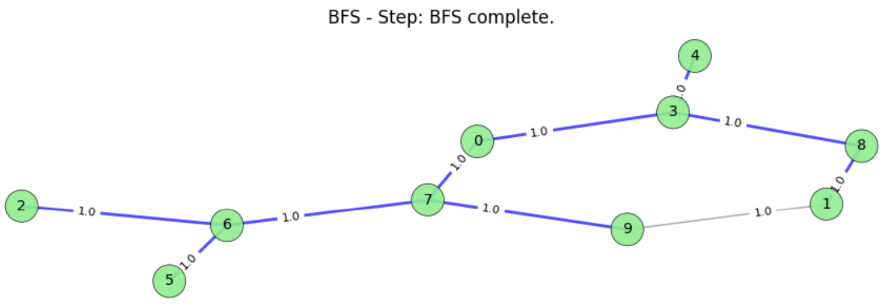

Figure 2: Graphical presentation of Breadth First Search (Replace with actual image)

3 Comparative Analysis of DFS and BFS

1. Key Characteristics

Characteristic	**DFS (Depth First Search)**	**BFS (Breadth First Search)**
Data Structure	Stack (or recursion)	Queue
Exploration	Goes deep first	Explores level by level
Pathfinding	Not guaranteed shortest	Finds shortest path (unweighted)
Completeness	Yes (for finite graphs)	Yes
Optimality	Not generally optimal	Optimal for shortest path (unweighted)
Memory (Worst Case)	$O(V)$ (depth of graph)	$O(V)$ (width of graph)
Time Complexity	$O(V + E)$	$O(V + E)$
Cycle Detection	Yes	Yes
Suited for	Topological sort, puzzles	Shortest path, network broadcast

(Table information synthesized from multiple sources including [DFS vs BFS Algorithm (All Differences With Example) - WsCube Tech](https://www.wscubetech.com/resources/dsa/dfs-vs-bfs) and [Difference between BFS and DFS - GeeksforGeeks](https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/))

2. Performance on Different Graph Structures

Graph Type	DFS Performance Notes	BFS Performance Notes
Deep, Narrow Graphs	Generally more memory efficient	Can be memory intensive
Wide, Shallow Graphs	Can be memory intensive (recursion)	Generally more memory efficie
Finding Solutions Deep in Graph	Might find solutions faster	Slower if solution is deep
Finding Solutions Close to Source	Slower if solution is shallow	Finds shallow solutions faster

(Table information synthesized from [8.4 Comparison of BFS and DFS - Intro To Algorithms - Fiveable](https://library.fiveable.me/introduction-algorithms/unit-8/comparison-bfs-dfs/study-guide/XtTDnw

3. Use Case Suitability

DFS is often preferred for:	BFS is often preferred for:
• Path existence checking • Topological sorting • Finding strongly connected components • Solving mazes or puzzles where path length doesn't matter • When memory is a constraint and graph is deep	• Finding the shortest path in unweighted graphs • Level-order traversal (e.g., in trees) • Finding all reachable nodes at a certain distance • Web crawling (to a certain depth) • Network broadcasting

(Information from [DFS vs BFS Algorithm (All Differences With Example) - WsCube Tech](https://www.wscu vs-bfs) and [Difference Between BFS and DFS - Scaler Blog](https://www.scaler.in/difference-between-bfs-and-dfs/))

Conclusion: Comparing Graph Traversal Algorithms

In this laboratory work, we studied two fundamental graph traversal algorithms: Depth First Search (DFS) and Breadth First Search (BFS). Both algorithms provide systematic ways to explore vertices and edges in a graph, but they differ significantly in their approach and suitability for various tasks.

Depth First Search (DFS)

DFS explores a graph by going as deep as possible along each branch before backtracking. It typically uses a stack (either explicitly or via recursion) to manage the order of visited nodes. DFS is well-suited for tasks such as detecting cycles, performing topological sorting in Directed Acyclic Graphs (DAGs), finding connected components, and solving pathfinding problems where the shortest path is not a requirement (e.g., maze solving). Its time complexity is $O(V + E)$, and its space complexity is $O(V)$ in the worst case, dependent on the maximum depth of the graph.

Breadth First Search (BFS)

BFS, on the other hand, explores the graph level by level. It utilizes a queue to manage the vertices to be visited. A key advantage of BFS is its ability to find the shortest path between two nodes in an unweighted graph, measured by the number of edges. It is also used for level-order traversals, finding all nodes within a certain distance, and as a component in more complex algorithms. Similar to DFS, its time complexity is $O(V+E)$, and its space complexity is $O(V)$ in the worst case, dependent on the maximum width of the graph.

Summary of Empirical Findings

(This section should be filled in based on your specific empirical analysis) Choosing between DFS and BFS depends heavily on the problem requirements and the structure of the graph.

- If the goal is to find the shortest path in an unweighted graph, BFS is the clear choice.
- If the problem involves exploring all possibilities down a certain path before trying others (e.g., puzzles, exhaustive search), DFS is often more natural.
- For very deep graphs, DFS might be more memory-efficient than BFS if implemented iteratively or if tail recursion is optimized. Conversely, for very wide graphs, BFS might require more memory due to the queue size.
- Both have the same theoretical time complexity, but practical performance can vary based on graph structure and implementation details.

This laboratory work provided practical experience in implementing and analyzing these essential graph algorithms, highlighting their distinct characteristics and application domains.

GitHub Repository Link: Dârzu Cătălin