Project Report: Maze Pathfinding Using Breadth-First Search and Depth-First Search

Author: عمرو عبد الحليم محمد النجار (ID: 2023147) **Date:** December 2025

1. Introduction

Search algorithms are a **fundamental component of Artificial Intelligence**. They are used to explore a problem space in order to find a sequence of actions that leads from an initial state to a goal state. The efficiency and effectiveness of an AI system often depend on the choice of the underlying search strategy.

In this project, two **uninformed search algorithms**—Breadth-First Search (BFS) and Depth-First Search (DFS)—are implemented and applied to solve a classic **Maze Pathfinding Problem**. The primary goal is to compare their behavior, efficiency, and performance using clear evaluation metrics such as path cost, completeness, and memory usage.

2. Problem Description

The Maze Pathfinding problem is defined on a two-dimensional grid. This grid represents the environment, and the agent's task is to navigate from a starting point to a target.

Each cell in the maze can be one of four types:

Cell Type	Symbol	Description
Start	`S`	The initial position of the agent.
Goal	`G` or `E`	The target position that the agent must reach.
Free Cell	`.`	A traversable cell that the agent can move through.
Wall	`#`	A blocked cell that cannot be crossed.

The agent is restricted to movement in **four cardinal directions** only: Up, Down, Left, and Right. The objective is to find a valid sequence of moves from the start cell to the goal cell.

---

3. Problem Formulation

The maze problem is formally formulated as a search problem using the following components:

Component	Definition
State Representation	A state is represented as a pair (r, c) indicating the current position (row, column) in the maze.
Initial State	The position of the start cell S.
Goal State	The position of the goal cell G (or E).
Actions	Move up, down, left, or right to an adjacent valid cell.
Transition Model	Movement from one cell to another is possible only if the target cell is inside the maze boundaries and is not a wall (`#`).
Path Cost	Each valid move has a **uniform cost of 1**.

---

4. Implemented Algorithms

4.1 Breadth-First Search (BFS)

Breadth-First Search explores the state space **level by level** using a **queue** (First-In, First-Out) data structure. It expands all nodes at a given depth before moving to the next depth level.

Because all moves in the maze have equal cost, BFS is **complete** (it will find a solution if one exists) and, more importantly, it **guarantees finding the shortest path** from the start to the goal.

4.2 Depth-First Search (DFS)

Depth-First Search explores the state space by going **as deep as possible** along one path before backtracking. It uses a **stack** (Last-In, First-Out) data structure.

DFS is also complete for finite state spaces like the maze, but it **does not guarantee the shortest path**. Its primary advantage lies in its **memory efficiency**, as it only needs to store the nodes along the current path.

---

5. Experimental Setup

To ensure a fair comparison, a fixed maze layout and starting conditions were used for all experiments.

5.1 Implementation Details

- **Language:** Both algorithms were implemented in **C++**.
- **Problem:** A single, fixed maze layout was used for all experiments.
- **Starting Conditions:** The start cell S and the goal cell E were fixed.

5.2 Recorded Metrics

The following metrics were recorded for a direct comparison of the algorithms:

1. **Path Length:** The number of moves required to reach the goal (Path Cost).
2. **Number of Expanded Nodes:** The total count of cells visited by the algorithm (a proxy for time complexity).
3. **Execution Time:** The time taken to find the solution (in microseconds).

---

6. Complexity Analysis

The theoretical complexity of both algorithms is often expressed in terms of V (the number of states or maze cells) and E (the number of possible transitions between states).

6.1 Breadth-First Search (BFS)

Metric	Complexity	Notes
Time Complexity	$O(V + E)$	Linear in the size of the graph.
Space Complexity	$O(V)$	BFS may require significant memory because it stores all frontier nodes at each level, which can be the entire graph in the worst case.

6.2 Depth-First Search (DFS)

Metric	Complexity	Notes
Time Complexity	$O(V + E)$	Linear in the size of the graph.
Space Complexity	$O(V)$ (Worst Case)	DFS generally uses less memory than BFS, as it only stores the current path, but the worst-case space is still proportional to the number of vertices.

---

7. Comparison Between BFS and DFS

The table below summarizes the key differences between the two uninformed search strategies:

Criterion	Breadth-First Search (BFS)	Depth-First Search (DFS)
Search Strategy	Level by level (Shallow first)	Depth first (Deep first)
Data Structure	**Queue** (FIFO)	**Stack** (LIFO)
Completeness	Complete	Complete (for finite graphs)
Optimality	**Optimal** (Guarantees shortest path)	**Not Optimal** (May find a longer path)
Memory Usage	Higher	Lower
Path Quality	Shortest path	May be a much longer path

---

8. Results and Discussion

The experimental results demonstrate the trade-off between optimality and search efficiency.

- **BFS** consistently finds the **shortest path** in the maze due to its level-wise exploration. This is crucial for applications where path cost is the primary concern.
- **DFS**, on the other hand, may reach the goal faster in terms of nodes expanded or time taken in certain maze layouts, but it often produces a longer, **non-optimal path**. Its performance depends heavily on the order in which neighboring nodes are explored and whether the goal lies deep down a path it explores early.

The choice between BFS and DFS is therefore dictated by the problem's requirements: **BFS** for guaranteed optimal solutions, and **DFS** for memory-constrained environments or when finding *any* solution quickly is the priority.

---

9. Conclusion

This project successfully demonstrated the fundamental differences between Breadth-First Search and Depth-First Search when applied to a maze pathfinding problem.

BFS is more suitable when optimal solutions are required, while **DFS is useful in scenarios where memory efficiency or fast exploration is more important**. The comparison highlights how different search strategies can significantly affect performance, even when solving the same problem.

---

10. References

[1] Artificial Intelligence Course Project Proposal