**Developing Search Algorithms**

David Ibegbu,

ARTIFICIAL INTELLIGENCE

February 5th, 2024

**Introduction**

Pathfinding is a fundamental problem in artificial intelligence and computer science, where an agent must navigate a space while avoiding obstacles to reach a specified goal. One of the most efficient algorithms for solving this problem in an unweighted grid environment is Breadth-First Search (BFS). BFS is a systematic search algorithm that explores nodes level by level, ensuring that the shortest path is found in an unweighted graph.

This paper explores the implementation of BFS in an 8×8 grid-based map, where certain cells are obstacles, preventing traversal. The BFS algorithm ensures that the shortest path is identified between a start position (S) and a goal position (G) while avoiding blocked areas (X).

**Overview of Breadth-First Search (BFS)**

BFS functions using the first in, first out queue, which allows us to guarantee that nodes are explored in the exact order they were found. The algorithm follows these steps: Initialization, Exploring, and Stopping. In the Initial phase, a queue is built to store nodes to be explored, and a boolean array is implemented to track visited nodes and prevent time-wasting through the process. In the exploration phase, the algorithm starts at the beginning point, which is where (S) is located in the code and enquires it. It analyzes each node in the queue by checking the vertices around it (up, right, down, left). If a neighbor is valid (no obstacles, within range, and not previously visited) it is marked as visited, added to the queue, and linked to the parent node. Finally, we have the stopping condition the algorithm continues this level-by-level analysis exploration process until it reaches the finishing point which in my code is (G). If (G) is found, BFS puts together the shortest path using the parent references stored for each node. If the end node is not found, then the queue becomes empty, and the code concludes that no path was found.

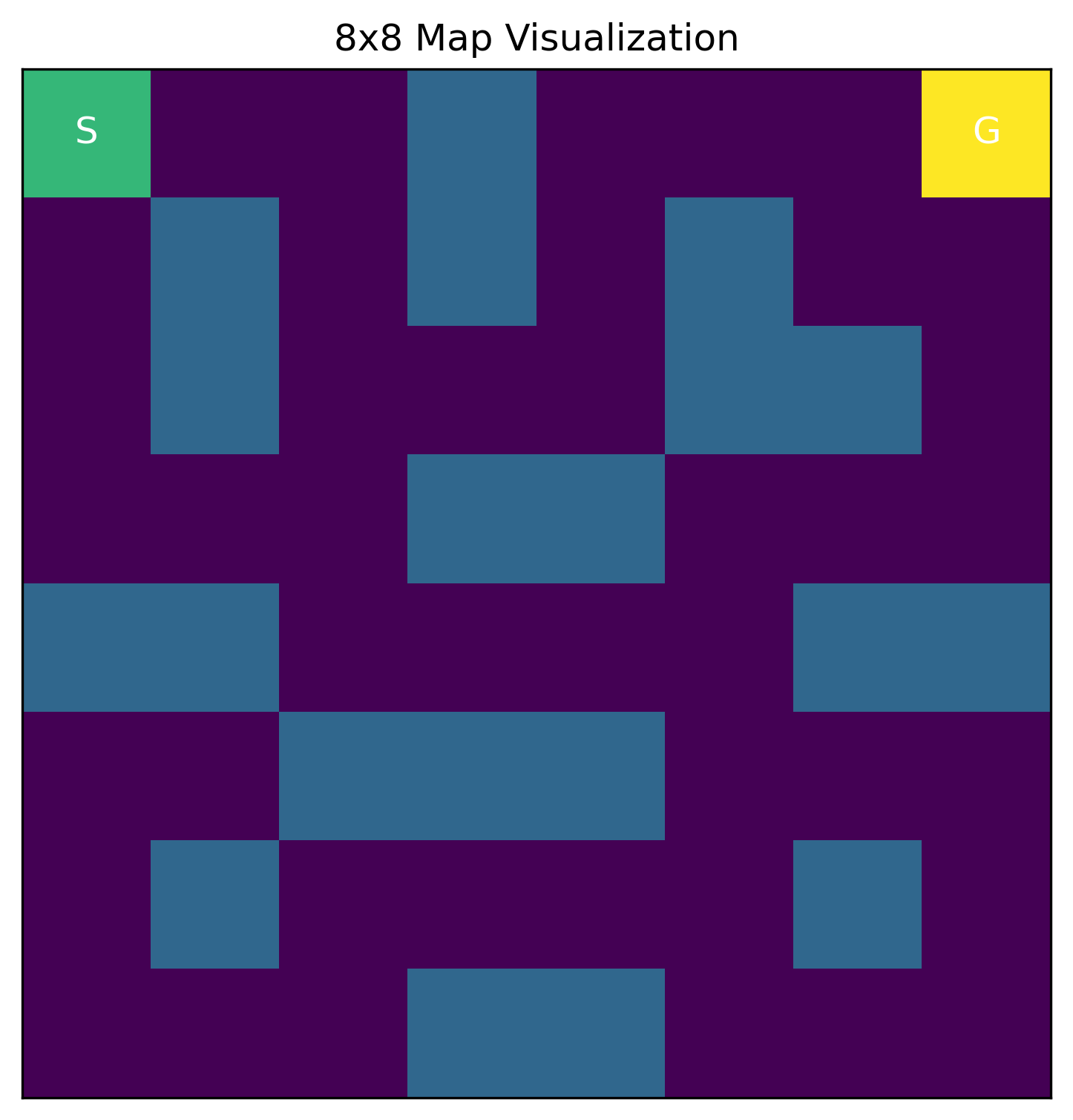
**BFS vs DFS**

Breadth-First Search (BFS) and Depth-First Search (DFS) are two fundamental graph traversal algorithms with distinct characteristics and applications. BFS explores nodes level by level, ensuring that all nodes at the current depth are visited before moving to nodes at the next depth. This approach guarantees the shortest path in unweighted graphs, making BFS suitable for pathfinding and shortest path problems. In contrast, DFS prioritizes exploring the most profound path first, often venturing as far as possible along a branch before backtracking. While DFS does not guarantee the shortest path, it is highly effective for tasks such as tree traversals and maze solving, where exploring all possible paths is more critical than finding the optimal one.

The data structures used by these algorithms also differ significantly. BFS employs a queue, which operates on a First-In-First-Out (FIFO) principle, ensuring that nodes are processed in the order they are discovered. This structure aligns with BFS's level-by-level exploration. On the other hand, DFS uses a stack that follows a Last-In-First-Out (LIFO) approach, allowing it to explore the most recently discovered nodes first, which is consistent with its depth-first strategy. These differences in traversal order and data structures make BFS and DFS suitable for different types of problems. BFS excels in scenarios where finding the shortest path is critical, while DFS is better suited for applications requiring exhaustive exploration of all possible paths, such as in tree traversals or solving mazes.

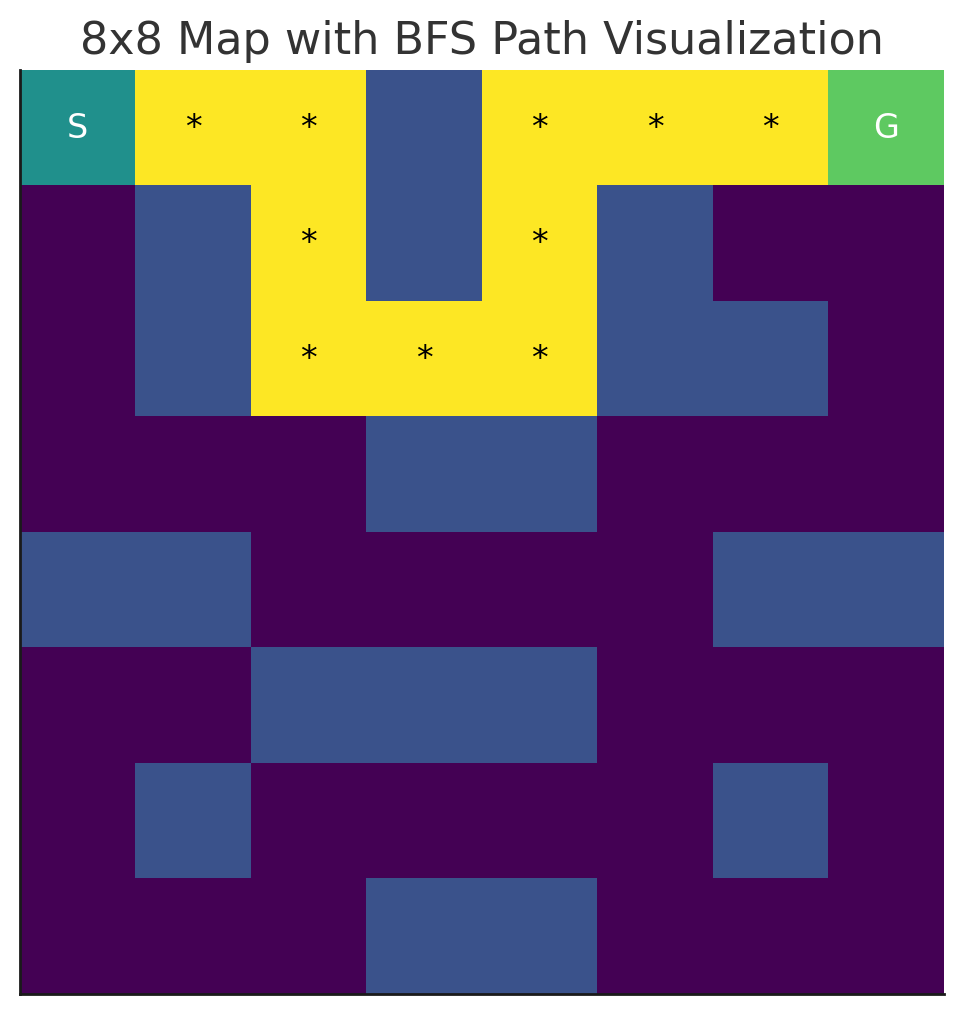
**Application of BFS to my Grid**

In the scenario of my code, the grid is represented as an 8x8 grid matrix with (S) acting as the staring position, (G) as the goal position, (-) acting as the open spaces, (X) acting as the Obstacles, and finally the (\*) the path taken from (S) to (G).



\*This is a map I generated using python to offer a visual representation of the map.\*

The BFS algorithm begins at the start position (S) and explores in four directions (Up, Down, Left, Right), ensuring that no obstacles (X) are traversed. Each valid move is added to a queue, allowing BFS to process all reachable cells level by level before advancing further. The algorithm continues this systematic exploration until it reaches the goal position (G), guaranteeing that the shortest path is found.



\*This is a map I generated using python to offer a visual representation of the path map.\*

The yellow-highlighted path (\*) shows the step-by-step traversal of the BFS algorithm, ensuring an efficient route without unnecessary detours. BFS successfully avoided the blue obstacles (X), demonstrating its ability to find an optimal solution in a structured grid environment. The total number of steps taken to reach the goal is 10 moves, making it a relatively direct path.

**Time Complexity**

The time Complexity of BFS depends on Number of node (V: grid cells) and the number of edges (E, connections between nodes). Since BFS explores all nodes in worst-case scenarios, the time complexity is given by: ***O(V + E)***. For a grid of size (N × N), BFS must traverse all cells once in the worst case. Each cell has at most 4 edges (Up, Down, Left, Right), so: ***O(N²+4N²)=O(N²)***. So in our best case ***O(1)*** resulting in the Goal being righ byv the start and the worst case being ***O(N²)*** which is the Goal being on the complete opposite end of the grid.

**Conclusion**

Breadth-First Search (BFS) is a powerful algorithm for grid-based pathfinding, ensuring the shortest path is found in an unweighted environment. Its key strengths include guaranteeing the shortest route, completeness, and simplicity, making it an ideal choice for applications such as robot navigation, game AI, and puzzle-solving. By systematically exploring all reachable nodes level by level, BFS ensures that if a solution exists, it will be found efficiently. However, BFS has notable limitations, particularly its high memory usage, as it must store all visited nodes, making it impractical for large-scale grids. Additionally, BFS is inefficient for weighted graphs, as it does not consider varying path costs, where alternative algorithms like A Search\* are preferred. Despite these limitations, BFS was successfully applied in this 8×8 grid-based implementation, demonstrating its ability to find an optimal path while effectively navigating obstacles. Future enhancements, such as incorporating A Search with heuristics\*, could further improve efficiency, particularly in larger and more complex environments.

**References**

AlgoMonster. (n.d.). Liteproblems/490: BFS Explanation. Retrieved from

<https://algo.monster/liteproblems/490>

GeeksforGeeks. (n.d.). Depth-First Search (DFS) for a Graph. Retrieved from

<https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>

GeeksforGeeks. (n.d.). Breadth-First Search (BFS) for a Graph. Retrieved from

<https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

YouTube. (n.d.). Breadth-First Search Algorithm Tutorial. Retrieved from

<https://www.youtube.com/watch?v=W9F8fDQj7Ok>