**Assignment 1**

**Title:**

Solving the 8-Puzzle Problem Using Depth-First Search (DFS) and Breadth-First Search (BFS)

**Aim:**

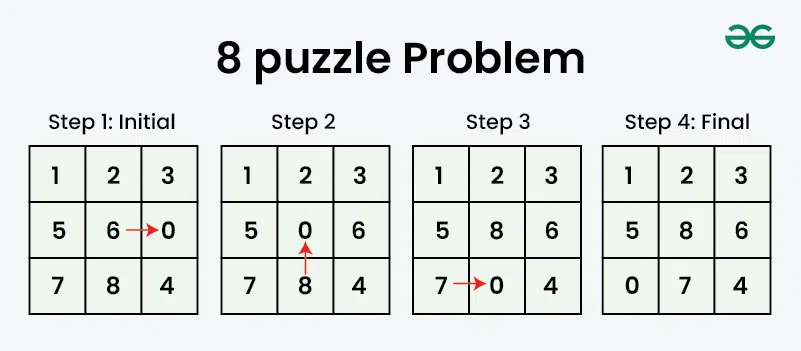
To solve the 8-puzzle problem using both Depth-First Search (DFS) and Breadth-First Search (BFS) algorithms and compare their efficiency in terms of time and memory usage.

**Objectives:**

1. To understand the 8-puzzle problem as a state-space search problem.
2. To implement DFS and BFS for finding solutions to the 8-puzzle problem.
3. To evaluate the performance of DFS and BFS algorithms in terms of time and memory.
4. To analyse the behaviour of the search algorithms when applied to this problem and identify their strengths and weaknesses.

**Theory:**

**The 8-Puzzle Problem:**

****

The 8-puzzle problem is a sliding puzzle that consists of a 3x3 grid with 8 numbered tiles and one empty space. The objective is to move the tiles in such a way that they achieve the goal state. The player can move any tile adjacent to the empty space into that space, effectively "sliding" the tile.

* **Initial State:** A random arrangement of the tiles.
* **Goal State:** The tiles are ordered as follows:

1 2 3

4 5 6

7 8 \_

(Where \_ denotes the empty space)

* **Possible Actions**: Up, Down, Left, Right (depending on the position of the empty space).
* State Representation: A 3x3 matrix, where each element is either a number from 1 to 8 or the empty space (\_).
* **Solution Path:** A sequence of tile moves leading from the initial state to the goal state.

**Depth-First Search (DFS):**

DFS explores as far as possible along each branch before backtracking. In the context of the 8-puzzle problem, DFS searches deeper into one branch of the state space until it finds a solution or reaches a dead end. DFS is not optimal (i.e., it may not always find the shortest path), and it may get stuck in infinite loops unless the search space is limited or visited states are tracked.

* **Advantage:** Low memory usage (linear with respect to depth of the search tree).
* **Disadvantage:** It is not guaranteed to find the shortest path and can get stuck in deep branches.

**Breadth-First Search (BFS):**

BFS explores all possible moves at the current depth level before moving on to the next depth level. In the 8-puzzle problem, BFS will explore all possible states reachable in one move, then two moves, and so on, until it finds the goal state. BFS is guaranteed to find the shortest path, but it requires significant memory to store all the nodes at the current level.

* **Advantage:** Guarantees the shortest path solution.
* **Disadvantage:** Requires more memory, as it must store all possible nodes at each level.

**Procedure:**

1. **Problem Definition and Setup:**

* Represent the 8-puzzle as a 3x3 matrix.
* Define the goal state as [[1, 2, 3], [4, 5, 6], [7, 8, 0]], where 0 represents the empty space.
* Define possible moves (Up, Down, Left, Right) based on the current position of the empty space.

1. **Implement Depth-First Search (DFS):**

* Start with the initial state.
* Implement DFS recursively (or using a stack) to explore all possible moves.
* Track visited states to avoid cycles and redundant calculations.
* Return the sequence of moves that leads to the goal state, or conclude that no solution exists.

1. **Implement Breadth-First Search (BFS):**

* Start with the initial state.
* Use a queue to explore all states at the current depth level before moving on to the next depth level.
* Keep track of visited states to prevent revisiting the same state multiple times.
* Return the sequence of moves that leads to the goal state, or conclude that no solution exist

1. **Evaluation and Comparison:**

* **For both DFS and BFS:**
* Count the number of nodes generated.
* Measure the time taken to find the solution.
* Track the memory usage (i.e., the number of states stored in memory during the search).
* Compare the results of DFS and BFS in terms of efficiency and memory usage.
* Analyze the path found by each algorithm and determine if DFS finds an optimal solution.

1. **Edge Cases:**

* Test the algorithms on edge cases such as:
* Already solved puzzle (Initial state = Goal state).
* Unsolvable puzzle (Check if a given puzzle configuration is unsolvable).
* Worst-case scenario where the goal state is farthest from the initial state.

**Expected Output:**

1. A sequence of moves leading from the initial state to the goal state.
2. For DFS:

* The path to the solution (not necessarily the shortest).
* The time taken to find the solution.
* The number of nodes generated and visited.

1. For BFS:

* The optimal (shortest) path to the solution.
* The time taken to find the solution.
* The number of nodes generated and visited.

1. Comparison of DFS and BFS performance (time and memory efficiency).

**Conclusion:**

In this lab, DFS and BFS were implemented to solve the 8-puzzle problem. BFS, as expected, provided the shortest path solution, whereas DFS might not find the optimal solution. However, DFS was more memory-efficient, while BFS consumed more memory. Both algorithms successfully solved the 8-puzzle problem, demonstrating their strengths and weaknesses in different aspects such as time complexity, memory usage, and solution optimality.