Assignment No: - 1

Name :- Isha Ghorpade

Enrollnment :- 22420020

Roll :- 381066

Problem Statement

Implement Depth First Search (DFS) and Breadth First Search (BFS) algorithms to solve the 8-puzzle problem, where the objective is to reach the goal state from a given initial configuration by moving the blank tile (0).

Objectives

- To understand the working of uninformed search strategies.
- To implement BFS and DFS for solving state-space search problems.
- To represent the 8-puzzle as a state-space problem.
- To evaluate the performance of BFS and DFS in terms of completeness, optimality, and efficiency.

S/W Packages and H/W apparatus used

- Operating System: Windows/Linux/MacOS
- Kernel: Python 3.x
- Tools: Jupyter Notebook, Anaconda, or Google Colab
- Hardware: CPU with minimum 4 GB RAM
- Libraries used: Collections (deque), basic Python lists

Theory

8-Puzzle Problem

The 8-puzzle problem consists of a 3×3 grid containing 8 numbered tiles and one blank space (represented as 0). The tiles can move up, down, left, or right into the blank space. The goal is to transform a given initial configuration into the goal state:

1 2 3

4 5 6

7 8 0

This is a classic state-space search problem in Artificial Intelligence.

State Space Representation

- State: Arrangement of tiles in the 3×3 grid.
- Initial State: The puzzle configuration given at the start.
- Goal State: The ordered configuration (as above).
- Operators: Move the blank tile (0) up, down, left, right.
- Path Cost: Number of moves made.

Breadth First Search (BFS)

- Type: Uninformed Search (Blind search).
- Explores all nodes at depth *d* before moving to depth *d+1*.
- Algorithm Steps:
    1. Start with the initial state in a queue.
    2. Expand nodes level by level.
    3. If goal is found → stop.
    4. Else, enqueue all unvisited neighboring states.
- Properties:
    - Complete (always finds solution if one exists).
    - Optimal (finds shortest solution path).
    - High memory usage.

Depth First Search (DFS)

- Type: Uninformed Search (Blind search).
- Explores nodes by going as deep as possible before backtracking.
- Algorithm Steps:
    1. Start with the initial state in a stack.
    2. Expand the deepest unexplored node.
    3. If goal is found → stop.
    4. Else, push unvisited neighbors into stack.

- **Properties:**
    - Not guaranteed to be complete (may go into infinite loop).
    - Not optimal (may not give shortest solution).
    - Low memory requirement compared to BFS.

Methodology

1. Represent puzzle states using a 3×3 matrix.
2. Define a function to locate the blank tile.
3. Define valid moves for the blank tile.
4. Implement BFS using a queue (FIFO).
5. Implement DFS using a stack (LIFO) with depth limit.
6. Check if the current state matches the goal state.
7. Stop when solution is found.

Advantages

- BFS: Guarantees shortest solution path.
- DFS: Requires less memory, easy to implement.

Limitations

- BFS: High memory and time consumption for large state spaces.
- DFS: May get stuck in deep branches, not guaranteed to find solution.

Applications

- Puzzle solving (8-puzzle, 15-puzzle).
- Pathfinding problems.
- AI planning.
- Game playing.

Algorithm

Breadth First Search (BFS):

1. Initialize a queue with initial state.
2. While queue is not empty:
    - Dequeue a state.
    - If goal → return solution.
    - Else enqueue neighbors.

Depth First Search (DFS):

1. Initialize a stack with initial state.
2. While stack is not empty:
    - Pop a state.
    - If goal → return solution.
    - Else push neighbors (within depth limit).

Diagram

```
     Initial State
      /    |    \
     /     |     \
  Next   Next   Next ...
  ↓      ↓      ↓
 Expanded states → Goal State
```

(BFS expands level by level, DFS expands path deeply first.)

Conclusion

The 8-puzzle problem demonstrates the working of uninformed search strategies.

- BFS guarantees the shortest solution but consumes more time and memory.
- DFS is faster and memory-efficient but may not always reach the goal or may provide a longer path. Both algorithms provide insights into solving real-world problems like pathfinding, game playing, and robotics using search techniques.