AI
LAB 2

Solve 8 puzzle problems
Implement Iterative deepening search algorithm

```
# 8-Puzzle Problem using Iterative Deepening Search (IDS)
# Date: 5 November 2024
# College: BMSCE

from copy import deepcopy

# ----- Define goal state -----
goal_state = [[1, 2, 3],
              [4, 5, 6],
              [7, 8, 0]]  # 0 represents the blank tile


# ----- Helper Functions -----
def find_blank(puzzle):
    """Find the position of the blank (0)."""
    for i in range(3):
        for j in range(3):
            if puzzle[i][j] == 0:
                return i, j


def is_goal(puzzle):
    """Check if puzzle is goal state."""
    return puzzle == goal_state


def print_puzzle(puzzle):
    """Print puzzle in readable format."""
    for row in puzzle:
        print(row)
    print()


# ----- Generate possible moves -----
def get_neighbors(puzzle):
    """Return all possible states reachable from current puzzle."""
    neighbors = []
    x, y = find_blank(puzzle)

    moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # Up, Down, Left, Right
    for dx, dy in moves:
```

```
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_puzzle = deepcopy(puzzle)
            new_puzzle[x][y], new_puzzle[nx][ny] = new_puzzle[nx][ny],
new_puzzle[x][y]
            neighbors.append(new_puzzle)
    return neighbors


# ----- Depth Limited Search (DLS) -----
def depth_limited_search(puzzle, depth_limit, path, visited):
    """Recursive DFS up to depth limit."""
    if is_goal(puzzle):
        return path

    if depth_limit == 0:
        return None

    visited.add(tuple(tuple(row) for row in puzzle))

    for neighbor in get_neighbors(puzzle):
        n_tuple = tuple(tuple(row) for row in neighbor)
        if n_tuple not in visited:
            result = depth_limited_search(neighbor, depth_limit - 1,
path + [neighbor], visited)
            if result is not None:
                return result
    return None


# ----- Iterative Deepening Search (IDS) -----
def iterative_deepening_search(start):
    """Perform IDS by increasing depth limit iteratively."""
    depth = 0
    while True:
        visited = set()
        result = depth_limited_search(start, depth, [start], visited)
        if result is not None:
            return result
        depth += 1


# ----- Example Input -----
initial_state = [[1, 2, 3],
                 [4, 0, 6],
                 [7, 5, 8]]

print("Initial State:")
print_puzzle(initial_state)
```

```
print("Goal State:")
print_puzzle(goal_state)

# ----- Run IDS -----
solution_path = iterative_deepening_search(initial_state)

# ----- Display Solution -----
print("✅ Solution Found using Iterative Deepening Search!")
print(f"Number of moves: {len(solution_path) - 1}\n")

for step, state in enumerate(solution_path):
    print(f"Step {step}:")
    print_puzzle(state)
```

OUTPUT:

```
Initial State:
[1, 2, 3]
[4, 0, 6]
[7, 5, 8]

Goal State:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]

✅ Solution Found using Iterative Deepening Search!
Number of moves: 2

Step 0:
[1, 2, 3]
[4, 0, 6]
[7, 5, 8]

Step 1:
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]

Step 2:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
```