LAB 3

```
from collections import deque

GOAL_STATE = (1, 2, 3, 4, 5, 6, 7, 8, 0)

def find_empty(state):
    return state.index(0)

def get_neighbors(state):
    neighbors = []
    empty_index = find_empty(state)
    row, col = divmod(empty_index, 3)
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    for dr, dc in directions:
        new_row, new_col = row + dr, col + dc
        if 0 <= new_row < 3 and 0 <= new_col < 3:
            new_index = new_row * 3 + new_col
            new_state = list(state)
            new_state[empty_index], new_state[new_index] = new_state[new_index], new_state[empty_index]
            neighbors.append(tuple(new_state))
    return neighbors

def bfs(initial_state):
    queue = deque([(initial_state, [])])
    visited = set()
    visited.add(initial_state)
    visited_count = 1  # Initialize visited count
    while queue:
        current_state, path = queue.popleft()
        if current_state == GOAL_STATE:
            return path, visited_count  # Return path and count
        for neighbor in get_neighbors(current_state):
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append((neighbor, path + [neighbor]))
                visited_count += 1  # Increment visited count
    return None, visited_count  # Return count if no solution found

def input_start_state():
    print("Enter the starting state as 9 numbers (0 for the empty space):")
    input_state = input("Format: 1 2 3 4 5 6 7 8 0\n")
    numbers = list(map(int, input_state.split()))
    if len(numbers) != 9 or set(numbers) != set(range(9)):
        print("Invalid input. Please enter numbers from 0 to 8 with no duplicates.")
        return input_start_state()
    return tuple(numbers)

def print_matrix(state):
    for i in range(0, 9, 3):
        print(state[i:i+3])

if __name__ == "__main__":
    initial_state = input_start_state()
    print("Initial state:")
    print_matrix(initial_state)
    solution, visited_count = bfs(initial_state)
    print(f"Number of states visited: {visited_count}")
    if solution:
        print("\nSolution found with the following steps:")
        for step in solution:
            print_matrix(step)
            print()
    else:
        print("No solution found.")
```

```
Enter the starting state as 9 numbers (0 for the empty space):
Format: 1 2 3 4 5 6 7 8 0
1 2 3 0 4 6 7 5 8
Initial state:
(1, 2, 3)
(0, 4, 6)
(7, 5, 8)
Number of states visited: 30

Solution found with the following steps:
(1, 2, 3)
(4, 0, 6)
(7, 5, 8)
```

```
(1, 2, 3)
(4, 5, 6)
(7, 0, 8)

(1, 2, 3)
(4, 5, 6)
(7, 8, 0)
```

Algorithm for 8 puzzle

8-Puzzle Algorithm

1. **Define Goal State**:
   - Set the goal state as `(1, 2, 3, 4, 5, 6, 7, 8, 0)`.
2. **Input Starting State**:
   - Prompt user for the initial configuration of the puzzle as 9 numbers (0-8).
3. **Initialize BFS**:
   - Create a queue and add the starting state with an empty path.
   - Create a set to track visited states.
4. **BFS Loop**:
   - While the queue is not empty:
     - Dequeue the current state.
     - If the current state matches the goal state, return the path.
     - Generate all valid neighboring states by sliding tiles into the empty space:
       - For each direction (up, down, left, right):
         - If the move is valid, create a new state and add it to the queue if not visited.
         - Mark the new state as visited and update the path.
5. **Output Result**:
   - If the goal state is found, print the sequence of moves.
   - If the queue is empty and the goal is not reached, indicate no solution exists.

```
from collections import deque

GOAL_STATE = (1, 2, 3, 4, 5, 6, 7, 8, 0)

def find_empty(state):
    return state.index(0)

def get_neighbors(state):
    neighbors = []
    empty_index = find_empty(state)
    row, col = divmod(empty_index, 3)
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    for dr, dc in directions:
        new_row, new_col = row + dr, col + dc
        if 0 <= new_row < 3 and 0 <= new_col < 3:
            new_index = new_row * 3 + new_col
            new_state = list(state)
            new_state[empty_index], new_state[new_index] = new_state[new_index], new_state[empty_index]
            neighbors.append(tuple(new_state))
    return neighbors

def dfs(initial_state):
    stack = [(initial_state, [])]
    visited = set()
    visited.add(initial_state)
    visited_count = 1  # Initialize visited count
    while stack:
        current_state, path = stack.pop()
        if current_state == GOAL_STATE:
            return path, visited_count  # Return path and count
```

```
            for neighbor in get_neighbors(current_state):
                if neighbor not in visited:
                    visited.add(neighbor)
                    stack.append((neighbor, path + [neighbor]))
                    visited_count += 1  # Increment visited count
    return None, visited_count  # Return count if no solution found

def input_start_state():
    print("Enter the starting state as 9 numbers (0 for the empty space):")
    input_state = input("Format: 1 2 3 4 5 6 7 8 0\n")
    numbers = list(map(int, input_state.split()))
    if len(numbers) != 9 or set(numbers) != set(range(9)):
        print("Invalid input. Please enter numbers from 0 to 8 with no duplicates.")
        return input_start_state()
    return tuple(numbers)

def print_matrix(state):
    for i in range(0, 9, 3):
        print(state[i:i+3])

if __name__ == "__main__":
    initial_state = input_start_state()
    print("Initial state:")
    print_matrix(initial_state)
    solution, visited_count = dfs(initial_state)
    print(f"Number of states visited: {visited_count}")
    if solution:
        print("\nSolution found with the following steps:")
        for step in solution:
            print_matrix(step)
            print()
    else:
        print("No solution found.")
```

```
Enter the starting state as 9 numbers (0 for the empty space):
Format: 1 2 3 4 5 6 7 8 0
1 2 3 4 5 6 0 7 8
Initial state:
(1, 2, 3)
(4, 5, 6)
(0, 7, 8)
Number of states visited: 5

Solution found with the following steps:
(1, 2, 3)
(4, 5, 6)
(7, 0, 8)

(1, 2, 3)
(4, 5, 6)
(7, 8, 0)
```

8-Puzzle DFS Algorithm

1. **Define Goal State**:
   - Set the goal state as `(1, 2, 3, 4, 5, 6, 7, 8, 0)`.
2. **Input Starting State**:
   - Prompt the user for the initial configuration of the puzzle as 9 numbers (0-8).
3. **Initialize DFS**:
   - Create a stack and push the starting state with an empty path.
   - Create a set to track visited states.
   - Initialize a counter for the visited states.
4. **DFS Loop**:
   - While the stack is not empty:
     - Pop the top state and its associated path from the stack.
     - If the current state matches the goal state, return the path and the visited count.
     - Generate all valid neighboring states by sliding tiles into the empty space:
       - For each direction (up, down, left, right):
         - If the move is valid, create a new state.
         - If the new state has not been visited: