LAB-3 8 PUZZLE PROBLEM

SHREE VARNA M

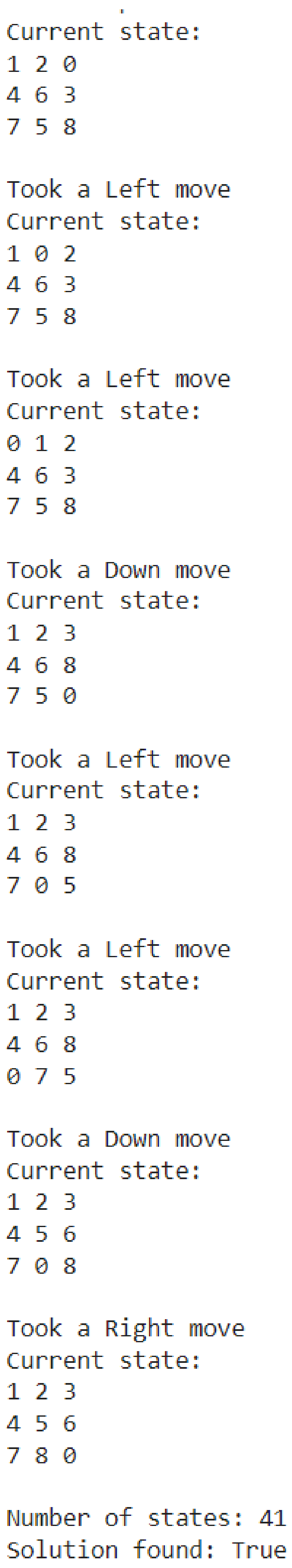
1BM22CS263

DFS :

```
count = 0
def print_state(in_array):
  global count
  count += 1
  for row in in_array:
    print(' '.join(str(num) for num in row))
  print()

def helper(goal, in_array, row, col, vis):
  # Marking current position as visited
  vis[row][col] = 1
  drow = [-1, 0, 1, 0]  # Dir for row: up, right, down, left
  dcol = [0, 1, 0, -1]  # Dir for column
  dchange = ['Up', 'Right', 'Down', 'Left']

  # Print current state
  print("Current state:")
  print_state(in_array)

  # Check if the current state is the goal state
  if in_array == goal:
    print(f"Number of states: {count}")
    return True

  # Explore all possible directions
  for i in range(4):
    nrow = row + drow[i]
    ncol = col + dcol[i]
```

```
        # Check if the new position is within bounds and not visited
        if 0 <= nrow < len(in_array) and 0 <= ncol < len(in_array[0]) and not vis[nrow][ncol]:
            # Make the move (swap the empty space with the adjacent tile)
            print(f"Took a {dchange[i]} move")
            in_array[row][col], in_array[nrow][ncol] = in_array[nrow][ncol], in_array[row][col]

            # Recursive call
            if helper(goal, in_array, nrow, ncol, vis):
                return True

            # Backtrack (undo the move)
            in_array[row][col], in_array[nrow][ncol] = in_array[nrow][ncol], in_array[row][col]

    # Mark the position as unvisited before returning
    vis[row][col] = 0
    return False

# Example usage
initial_state = [[1, 2, 3], [0, 4, 6], [7, 5, 8]]  # 0 represents the empty space
goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
visited = [[0] * 3 for _ in range(3)]  # 3x3 visited matrix
empty_row, empty_col = 1, 0  # Initial position of the empty space

found_solution = helper(goal_state, initial_state, empty_row, empty_col, visited)
print("Solution found:", found_solution)
```

OUTPUT:

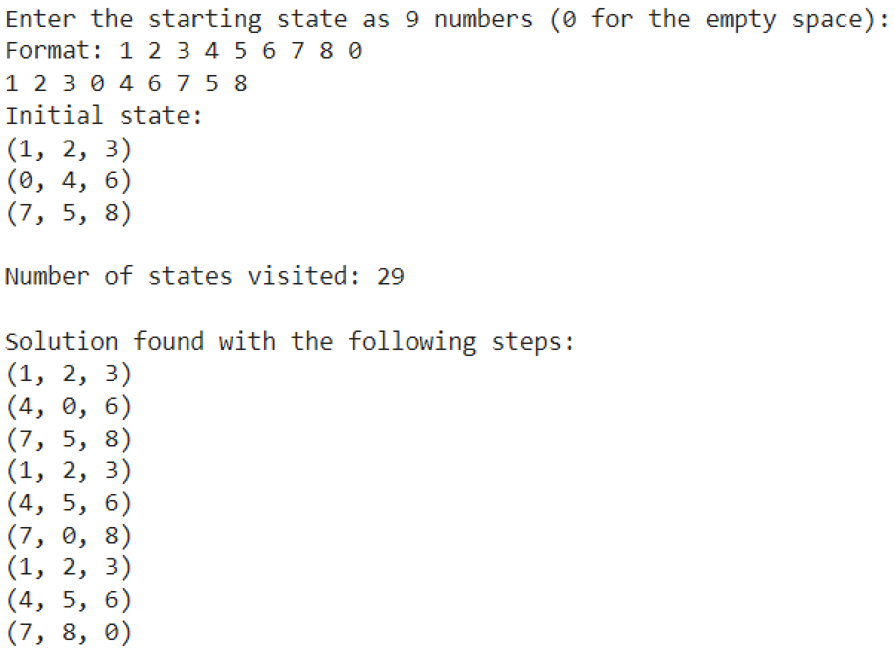


```
Current state:
1 2 0
4 6 3
7 5 8

Took a Left move
Current state:
1 0 2
4 6 3
7 5 8

Took a Left move
Current state:
0 1 2
4 6 3
7 5 8

Took a Down move
Current state:
1 2 3
4 6 8
7 5 0

Took a Left move
Current state:
1 2 3
4 6 8
7 0 5

Took a Left move
Current state:
1 2 3
4 6 8
0 7 5

Took a Down move
Current state:
1 2 3
4 5 6
7 0 8

Took a Right move
Current state:
1 2 3
4 5 6
7 8 0

Number of states: 41
Solution found: True
```

BFS:

```
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:
                queue.append((neighbor, path + [neighbor]))
                visited.add(neighbor)
                visited_count += 1  # Increment visited count
    return None, visited_count  # Return count if no solution found
def input_start_state():
    while True:
        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")

        try:
            numbers = list(map(int, input_state.split()))
            if len(numbers) != 9 or set(numbers) != set(range(9)):
                raise ValueError
            return tuple(numbers)
        except ValueError:
            print("Invalid input. Please enter numbers from 0 to 8 with no duplicates.")

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

if __name__ == "__main__":  # Corrected main check
    initial_state = input_start_state()
    print("Initial state:")
    print_matrix(initial_state)
    print()
    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)
    else:
        print("\nNo solution found.")
```

OUTPUT:



```
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: 29

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)
```