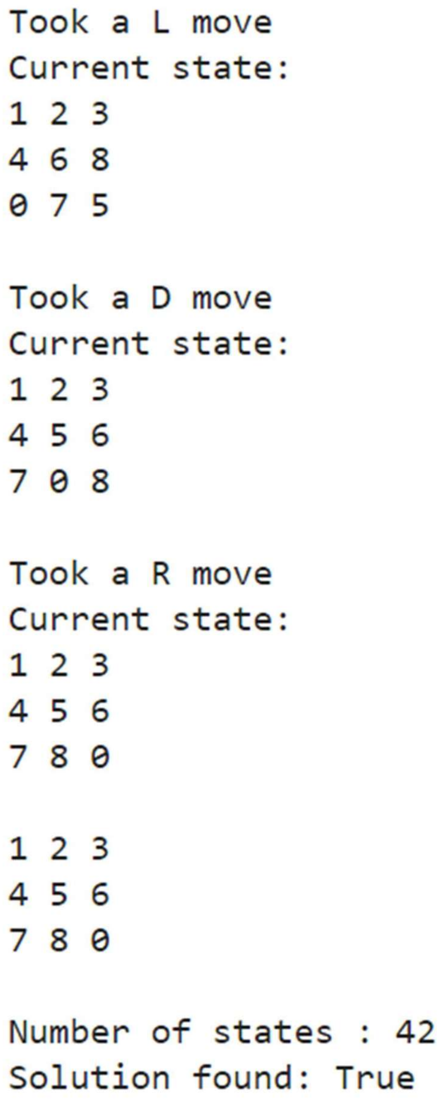
1BM22CS241

Lab-3: 8 puzzle

1. DFS

```
cnt = 0;
def print_state(in_array):
    global cnt
    cnt += 1
    for row in in_array:
        print(' '.join(str(num) for num in row))
    print()  # Print a blank line for better readability

def helper(goal, in_array, row, col, vis):
    # Mark the current position as visited
    vis[row][col] = 1
    drow = [-1, 0, 1, 0]  # Directions for row movements: up, right, down, left
    dcol = [0, 1, 0, -1]  # Directions for column movements
    dchange = ['U', 'R', 'D', 'L']

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

    # Check if the current state is the goal state
    if in_array == goal:
        print_state(in_array)
        print(f"Number of states : {cnt}")
        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
    returningvis[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:

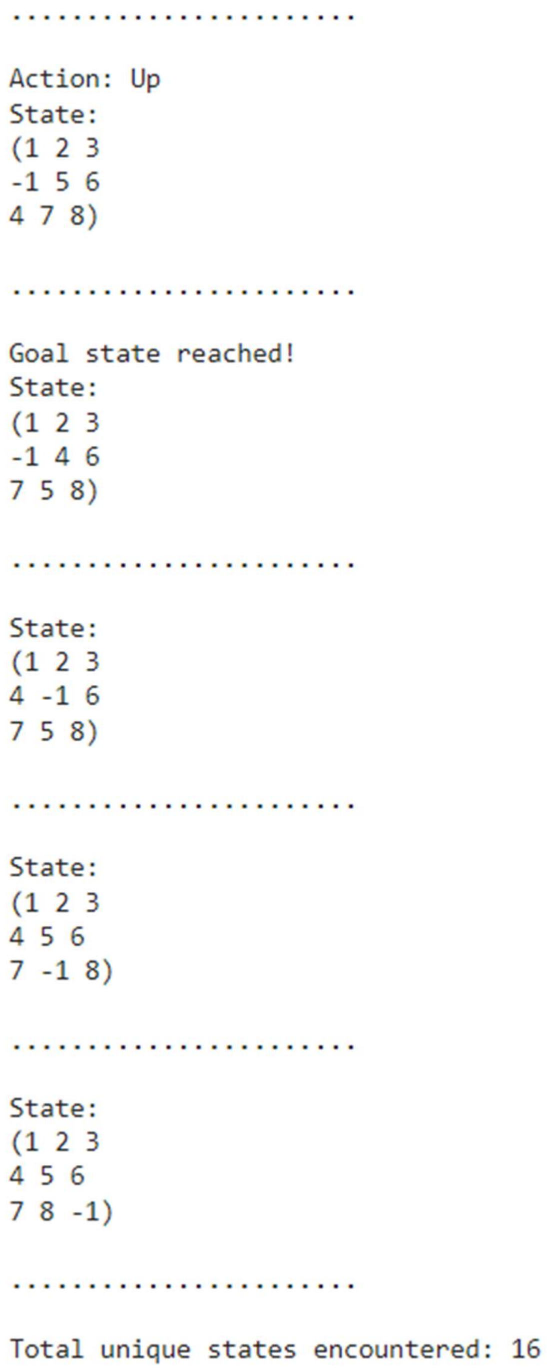


```
Took a L move
Current state:
1 2 3
4 6 8
0 7 5

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

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

1 2 3
4 5 6
7 8 0

Number of states : 42
Solution found: True
```

2. BFS

```
from collections import deque

class PuzzleState:
    def __init__(self, board, empty_tile_pos, moves):
        self.board = board
        self.empty_tile_pos = empty_tile_pos
        self.moves = moves  # Record the path of moves

    def __str__(self):
        return '\n'.join([' '.join(map(str, row)) for row in self.board])

    def is_goal(self):
        return self.board == [[1, 2, 3], [4, 5, 6], [7, 8, -1]]

    def get_possible_moves(self):
        row, col = self.empty_tile_pos
        moves = []
        if row > 0: moves.append((-1, 0))  # Up
        if row < 2: moves.append((1, 0))   # Down
        if col > 0: moves.append((0, -1))  # Left
        if col < 2: moves.append((0, 1))   # Right
        return moves

    def move(self, direction):
        row, col = self.empty_tile_pos
        new_row, new_col = row + direction[0], col + direction[1]
        new_board = [r[:] for r in self.board]  # Deep copy the board
        new_board[row][col], new_board[new_row][new_col] = new_board[new_row][new_col], new_board[row][col]
        return PuzzleState(new_board, (new_row, new_col), self.moves + [new_board])

    def to_string(self):
        return ''.join(map(str, [num for row in self.board for num in row]))

def bfs(initial_state):
    queue = deque([initial_state])
    visited = set()
    unique_states_count = 0  # Counter for unique states

    actions = {(-1, 0): "Up", (1, 0): "Down", (0, -1): "Left", (0, 1): "Right"}

    while queue:
        current_state = queue.popleft()

        if current_state.is_goal():
            print("Goal state reached!")
            for step in current_state.moves:
                print(f"State:\n({PuzzleState(step, current_state.empty_tile_pos, []).__str__()})\n")
                print("......................\n")
            break  # Exit the loop once the goal is found

        state_string = current_state.to_string()
        if state_string not in visited:
            visited.add(state_string)
            unique_states_count += 1  # Increment the unique states counter

            for move in current_state.get_possible_moves():
                new_state = current_state.move(move)
                if new_state.to_string() not in visited:
                    action_taken = actions[move]
                    print(f"Action: {action_taken}")
                    print(f"State:\n({new_state})\n")
                    print(".......................\n")
                    queue.append(new_state)

    print(f"Total unique states encountered: {unique_states_count}")


def main():
    initial_state_input = input("Enter the initial state (e.g. '1 2 3 4 5 6 7 8 -'): ")
    initial_state_list = []

    for value in initial_state_input.split():
        if value == '-':
            initial_state_list.append(-1)  # Use -1 for the empty tile
        else:
            initial_state_list.append(int(value))

    initial_state_board = [initial_state_list[i:i+3] for i in range(0, 9, 3)]
    empty_tile_pos = [(i, row.index(-1)) for i, row in enumerate(initial_state_board) if -1 in row][0]

    initial_state = PuzzleState(initial_state_board, empty_tile_pos, [initial_state_board])

    bfs(initial_state)


if __name__ == "__main__":
    main()
```

<u>Output:</u>



```
.......................

Action: Up
State:
(1 2 3
-1 5 6
4 7 8)

.......................

Goal state reached!
State:
(1 2 3
-1 4 6
7 5 8)

.......................

State:
(1 2 3
4 -1 6
7 5 8)

.......................

State:
(1 2 3
4 5 6
7 -1 8)

.......................

State:
(1 2 3
4 5 6
7 8 -1)

.......................

Total unique states encountered: 16
```