Shuvam Rajbanshi
1BM22CS275
5-E

LAB-3 8 PUZZLE USING BFS & DFS

```
#BREADTH FIRST SEARCH

def bfs(src, target):
    queue = []
    queue.append(src)

    exp = []

    while len(queue) > 0:
        source = queue.pop(0)
        exp.append(source)

        print("Current State:")
        print_matrix(source)

        if source == target:
            print("Success!")
            return

        poss_moves_to_do = possible_moves(source, exp)

        for move in poss_moves_to_do:
            if move not in exp and move not in queue:
                queue.append(move)

def possible_moves(state, visited_states):
    # Index of empty spot
    b = state.index(0)

    # Directions array
    d = []

    # Add all possible directions
    if b not in [0, 1, 2]:
        d.append('u')
    if b not in [6, 7, 8]:
        d.append('d')
    if b not in [0, 3, 6]:
        d.append('l')
    if b not in [2, 5, 8]:
```

```
        d.append('r')

    # Generate possible moves
    pos_moves_it_can = []
    for i in d:
        pos_moves_it_can.append(gen(state, i, b))

    return [move_it_can for move_it_can in pos_moves_it_can if move_it_can
not in visited_states]

def gen(state, m, b):
    temp = state.copy()

    if m == 'd':
        temp[b + 3], temp[b] = temp[b], temp[b + 3]
    elif m == 'u':
        temp[b - 3], temp[b] = temp[b], temp[b - 3]
    elif m == 'l':
        temp[b - 1], temp[b] = temp[b], temp[b - 1]
    elif m == 'r':
        temp[b + 1], temp[b] = temp[b], temp[b + 1]

    return temp

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

def print_matrix(state):
    matrix = convert_to_matrix(state)
    for row in matrix:
        print(row)
    print()

# Example usage
src = [1, 0, 3, 4, 2, 6, 7, 5, 8]
target = [1, 2, 3, 4, 5, 6, 7, 8, 0]
bfs(src, target)
```

Output:

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

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

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

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

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

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

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

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

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

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

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

Success!
```

```
#DEPTH FIRST SEARCH
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 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 3
0 4 6
7 5 8

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

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

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

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

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

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

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

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

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