LAB - 2 [8 Puzzle]

```
from collections import deque
goal_state = [[1, 2, 3],
              [4, 5, 6],
              [7, 8, 0]]
moves = {'up': (-1, 0), 'down': (1, 0), 'left': (0, -1),
'right': (0, 1)}

def find_blank(state):
    for i in range(len(state)):
        for j in range(len(state[i])):
            if state[i][j] == 0:
                return i, j


def is_goal(state):
    return state == goal_state

def get_neighbors(state):
    neighbors = []
    blank_row, blank_col = find_blank(state)
    for move, (dr, dc) in moves.items():
        new_row, new_col = blank_row + dr, blank_col + dc
        if 0 <= new_row < 3 and 0 <= new_col < 3:

            new_state = [row[:] for row in state]

            new_state[blank_row][blank_col],
new_state[new_row][new_col] = new_state[new_row]
[new_col], new_state[blank_row][blank_col]
            neighbors.append((new_state, move))
    return neighbors

def print_puzzle(state):
    for row in state:
        print(row)
    print()


def bfs(start_state):

    queue = deque([(start_state, [])])
```

```
    visited = set()
    visited.add(tuple(tuple(row) for row in start_state))

    while queue:
        current_state, path = queue.popleft()

        if is_goal(current_state):
            return current_state, path

        for neighbor, move in
get_neighbors(current_state):
            state_tuple = tuple(tuple(row) for row in
neighbor)
            if state_tuple not in visited:
                visited.add(state_tuple)
                queue.append((neighbor, path + [move]))

    return None, None

def get_user_input():
    print("Enter the initial state of the 8-puzzle (row
by row):")
    state = []
    for i in range(3):
        while True:
            try:
                row = list(map(int, input(f"Enter row
{i+1} (3 integers, space-separated): ").split()))
                if len(row) != 3 or any(x not in range(9)
for x in row):
                    raise ValueError
                state.append(row)
                break
            except ValueError:
                print("Invalid input. Please enter 3
integers between 0 and 8.")
    return state
```

```
def demonstrate_solution(start_state, solution_path):
current_state = start_state
    print("Initial state:")
    print_puzzle(current_state)

    for move in solution_path:
        print(f"Move: {move}")
        for neighbor, move_name in
get_neighbors(current_state):
            if move_name == move:
                current_state = neighbor
                print_puzzle(current_state)
                break

if __name__ == "__main__":
    start_state = get_user_input()
    final_state, solution_path = bfs(start_state)
    if solution_path:
        print("Solution found. Steps are demonstrated
below:")
        demonstrate_solution(start_state, solution_path)
    else:
        print("No solution found.")
```

OUTPUT:

```
Enter the initial state of the 8-puzzle (row by row):
Enter row 1 (3 integers, space-separated): 1 2 3
Enter row 2 (3 integers, space-separated): 0 4 6
Enter row 3 (3 integers, space-separated): 7 5 8
Solution found. Steps are demonstrated below:
Initial state:
[1, 2, 3]
[0, 4, 6]
[7, 5, 8]

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

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

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

```
Enter the initial state of the 8-puzzle (row by row):
Enter row 1 (3 integers, space-separated): 1 3 2
Enter row 2 (3 integers, space-separated): 4 6 0
Enter row 3 (3 integers, space-separated): 7 5 8
No solution found.
```