

LAB 3

8 Puzzle problem

CODE:

```
import copy

class Node:
    def __init__(self, state, parent=None, action=None, path_cost=0):
        self.state = state
        self.parent = parent
        self.action = action
        self.path_cost = path_cost

    def __lt__(self, other):
        return self.path_cost < other.path_cost

    def expand(self):
        children = []
        row, col = self.find_blank()
        possible_actions = []
        if row > 0:
            possible_actions.append('Up')
        if row < 2:
            possible_actions.append('Down')
        if col > 0:
            possible_actions.append('Left')
        if col < 2:
            possible_actions.append('Right')

        for action in possible_actions:
            new_state = copy.deepcopy(self.state)
            if action == 'Up':
                new_state[row][col], new_state[row - 1][col] =
new_state[row - 1][col], new_state[row][col]
            elif action == 'Down':
                new_state[row][col], new_state[row + 1][col] =
new_state[row + 1][col], new_state[row][col]
            elif action == 'Left':
```

```

        new_state[row][col], new_state[row][col - 1] =
new_state[row][col - 1], new_state[row][col]
        elif action == 'Right':
            new_state[row][col], new_state[row][col + 1] =
new_state[row][col + 1], new_state[row][col]
            children.append(Node(new_state, self, action, self.path_cost +
1))

    return children

def find_blank(self):
    for row in range(3):
        for col in range(3):
            if self.state[row][col] == 0:
                return row, col

def depth_first_search(initial_state, goal_state):
    frontier = [Node(initial_state)]
    explored = set()

    while frontier:
        node = frontier.pop()
        if node.state == goal_state:
            return node
        explored.add(tuple(map(tuple, node.state))) # Convert the state to
a tuple of tuples

        for child in node.expand():
            child_state_tuple = tuple(map(tuple, child.state))
            if child_state_tuple not in explored:
                frontier.append(child)
    return None

def print_solution(node):
    path = []

    while node is not None:
        path.append((node.action, node.state))
        node = node.parent

    path.reverse() # Reverse the path to start from the initial state

```

```

for action, state in path:
    if action:
        print(f"Action: {action}")
        for row in state:
            print(row)
        print()

# Test the DFS with initial and goal states
initial_state = [[1, 2, 3], [0, 4, 6], [7, 5, 8]]
goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

solution = depth_first_search(initial_state, goal_state)

if solution:
    print("Solution found:")
    print_solution(solution)
else:
    print("Solution not found.")

```

OUTPUT:

Streaming output truncated to the last 5000 lines.

```

Action: Right
[3, 2, 7]
[6, 4, 8]
[5, 0, 1]

Action: Up
[3, 2, 7]
[6, 0, 8]
[5, 4, 1]

Action: Right
[3, 2, 7]
[6, 8, 0]
[5, 4, 1]

Action: Down
[3, 2, 7]
[6, 8, 1]
[5, 4, 0]

Action: Left
[3, 2, 7]
[6, 8, 1]
[5, 0, 4]

Action: Left
[3, 2, 7]
[6, 8, 1]
[0, 5, 4]

Action: Up
[3, 2, 7]

```