Lab - 3

Manhattan:

```
import heapq
GOAL_STATE = ((1, 2, 3),
              (8, 0, 4),
              (7, 6, 5))
def manhattan_distance(state):
    distance = 0
    for i in range(3):
        for j in range(3):
            value = state[i][j]
            if value != 0:

                goal_x, goal_y = divmod(value - 1, 3)
                distance += abs(goal_x - i) + abs(goal_y - j)
    return distance
def find_blank(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j

def generate_neighbors(state):
    neighbors = []
    x, y = find_blank(state)
    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    for dx, dy in directions:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_state = [list(row) for row in state]
            new_state[x][y], new_state[nx][ny] =
new_state[nx][ny], new_state[x][y]
            neighbors.append(tuple(tuple(row) for row in
new_state))
    return neighbors

def reconstruct_path(came_from, current):
    path = [current]
    while current in came_from:
        current = came_from[current]
        path.append(current)
    path.reverse()
    return path
```

```
def a_star(start):
    open_list = []
    heapq.heappush(open_list, (manhattan_distance(start), 0,
start))

    g_score = {start: 0}
    came_from = {}

    visited = set()

    while open_list:
        f, g, current = heapq.heappop(open_list)

        if current == GOAL_STATE:
            path = reconstruct_path(came_from, current)
            return path, g

        visited.add(current)
        for neighbor in generate_neighbors(current):
            if neighbor in visited:
                continue

            tentative_g = g_score[current] + 1

            if tentative_g < g_score.get(neighbor,
float('inf')):
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g
                f_score = tentative_g +
manhattan_distance(neighbor)

                heapq.heappush(open_list, (f_score,
tentative_g, neighbor))

    return None, None

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

if __name__ == "__main__":
    start_state = ((2, 8, 3),
                   (1, 6, 4),
```

```
                    (7, 0, 5))

    print("Initial State:")
    print_state(start_state)

    print("Goal State:")
    print_state(GOAL_STATE)

    solution, cost = a_star(start_state)

    if solution:
        print(f"Solution found with cost: {cost}")
        print("Steps:")
        for step in solution:
            print_state(step)
    else:
        print("No solution found.”)
```

Output:

```
Initial State:
(2, 8, 3)
(1, 6, 4)
(7, 0, 5)

Goal State:
(1, 2, 3)
(8, 0, 4)
(7, 6, 5)

Solution found with cost: 5
Steps:
(2, 8, 3)
(1, 6, 4)
(7, 0, 5)

(2, 8, 3)
(1, 0, 4)
(7, 6, 5)

(2, 0, 3)
(1, 8, 4)
(7, 6, 5)

(0, 2, 3)
(1, 8, 4)
(7, 6, 5)

(1, 2, 3)
(0, 8, 4)
(7, 6, 5)

(1, 2, 3)
(8, 0, 4)
(7, 6, 5)
```