Lab 4 - A* Algorithm for 8 Puzzle Problem

Code for Manhattan Distance

```
import heapq

def solve(src, target):
    queue = []
    heapq.heappush(queue, (0, src, 0, []))  # (cost, state, depth, path of moves)
    visited = {}
    visited[tuple(src)] = None  # Store the parent of the initial state as None

    while len(queue) > 0:
        cost, source, depth, moves = heapq.heappop(queue)
        print("---------------------------------")
        print_state(source)
        print("Cost:", cost)
        print("Depth:", depth)
        print("Moves:", " ".join(moves))

        if source == target:
            total_cost = cost + depth
            print("Success with total cost:", total_cost)
            print("Path to target:", reconstruct_path(visited, source))
            return

        poss_moves_to_do = possible_moves(source, visited)
        for move, direction in poss_moves_to_do:
            move_tuple = tuple(move)
            if move_tuple not in visited:
                move_cost = calculate_cost(move, target)
                heapq.heappush(queue, (move_cost, move, depth + 1, moves + [direction]))
                visited[move_tuple] = tuple(source)  # Record the parent state

def print_state(state):
    for i in range(9):
```

```
        if i % 3 == 0:
            print("\n")
        if state[i] == 0:
            print("_ ", end="")
        else:
            print(str(state[i]) + " ", end="")
    print("\n")

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

    # Add all the possible directions with corresponding moves
    if b not in [0, 1, 2]:  # Up
        new_state = gen(state, 'u', b)
        directions.append((new_state, 'u'))
    if b not in [6, 7, 8]:  # Down
        new_state = gen(state, 'd', b)
        directions.append((new_state, 'd'))
    if b not in [0, 3, 6]:  # Left
        new_state = gen(state, 'l', b)
        directions.append((new_state, 'l'))
    if b not in [2, 5, 8]:  # Right
        new_state = gen(state, 'r', b)
        directions.append((new_state, 'r'))

    # Filter out visited states
    return [(move, direction) for move, direction in directions if
tuple(move) not in visited_states]

def gen(state, move, b):
    temp = state.copy()
    if move == 'd':
        temp[b], temp[b + 3] = temp[b + 3], temp[b]
    elif move == 'u':
        temp[b], temp[b - 3] = temp[b - 3], temp[b]
    elif move == 'l':
        temp[b], temp[b - 1] = temp[b - 1], temp[b]
    elif move == 'r':
        temp[b], temp[b + 1] = temp[b + 1], temp[b]
```

```
    return temp

def calculate_cost(state, target):
    cost = 0
    for i in range(len(state)):
        if state[i] != 0:  # Exclude the empty space
            target_index = target.index(state[i])
            cost += abs(target_index // 3 - i // 3) + abs(target_index % 3
- i % 3)
    return cost

def reconstruct_path(visited, target):
    path = []
    current = tuple(target)
    while current is not None:
        path.append(current)
        current = visited.get(current)  # Use get to avoid KeyError
    return path[::-1]  # Return reversed path

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

Output for Manhattan Distance