LAB PROGRAM 4

Solve 8 puzzles using A* algorithm:

(a) Number of misplaced tiles:

PSEUDOCODE:

CODE:

```
from heapq import heappush, heappop

class PuzzleState:
    def __init__(self, board, parent=None, move=None, depth=0):
        self.board = board
        self.parent = parent
        self.move = move
        self.depth = depth
        self.zero_pos = self.board.index(0)

    def is_goal(self, goal):
        return self.board == goal

    def get_moves(self):
        moves = []
        zero = self.zero_pos
        row, col = zero // 3, zero % 3

        directions = {
            'Up': (row - 1, col),
            'Down': (row + 1, col),
            'Left': (row, col - 1),
            'Right': (row, col + 1)
        }

        for move, (r, c) in directions.items():
            if 0 <= r < 3 and 0 <= c < 3:
                new_zero = r * 3 + c
                new_board = list(self.board)
                new_board[zero], new_board[new_zero] =
new_board[new_zero], new_board[zero]
                moves.append(PuzzleState(tuple(new_board), self, move,
self.depth + 1))
        return moves

    def misplaced_tiles(self, goal):
        return sum(1 for i, tile in enumerate(self.board) if tile != 0
and tile != goal[i])
```

```
    def __lt__(self, other):
        return True

def a_star(start, goal):
    open_list = []
    closed_set = set()
    start_state = PuzzleState(start)
    heappush(open_list, (start_state.misplaced_tiles(goal),
start_state))

    while open_list:
        _, current = heappop(open_list)

        if current.is_goal(goal):
            return reconstruct_path(current)

        closed_set.add(current.board)

        for neighbor in current.get_moves():
            if neighbor.board in closed_set:
                continue
            cost = neighbor.depth + neighbor.misplaced_tiles(goal)
            heappush(open_list, (cost, neighbor))

    return None

def reconstruct_path(state):
    path = []
    while state.parent is not None:
        path.append(state.move)
        state = state.parent
    path.reverse()
    return path
if __name__ == "__main__":
    start = (2, 8, 3,
             1, 6, 4,
             7, 0, 5)

    goal = (1, 2, 3,
            8, 0, 4,
            7, 6, 5)
    print("Sinchana Hemanth (1BM23CS330)")
    solution = a_star(start, goal)
    if solution:
        print(f"Solution found in {len(solution)} moves: {solution}")
    else:
        print("No solution found.")
```

OUTPUT:

```
Sinchana Hemanth (1BM23CS330)
Solution found in 5 moves: ['Up', 'Up', 'Left', 'Down', 'Right']
```

(b) Manhattan distance:

PSEUDOCODE:

CODE:

```
from heapq import heappush, heappop

class PuzzleState:
    def __init__(self, board, parent=None, move=None, depth=0):
        self.board = board
        self.parent = parent
        self.move = move
        self.depth = depth
        self.zero_pos = self.board.index(0)

    def is_goal(self, goal):
        return self.board == goal

    def get_moves(self):
        moves = []
        zero = self.zero_pos
        row, col = zero // 3, zero % 3

        directions = {
            'Up': (row - 1, col),
            'Down': (row + 1, col),
            'Left': (row, col - 1),
            'Right': (row, col + 1)
        }

        for move, (r, c) in directions.items():
            if 0 <= r < 3 and 0 <= c < 3:
                new_zero = r * 3 + c
                new_board = list(self.board)
                new_board[zero], new_board[new_zero] =
new_board[new_zero], new_board[zero]
                moves.append(PuzzleState(tuple(new_board), self, move,
self.depth + 1))
        return moves

    def manhattan_distance(self, goal):
```

```
        distance = 0
        for i, tile in enumerate(self.board):
            if tile != 0:
                goal_index = goal.index(tile)
                current_row, current_col = i // 3, i % 3
                goal_row, goal_col = goal_index // 3, goal_index % 3
                distance += abs(current_row - goal_row) +
abs(current_col - goal_col)
        return distance

    def __lt__(self, other):
        return True

def a_star(start, goal):
    open_list = []
    closed_set = set()
    start_state = PuzzleState(start)
    heappush(open_list, (start_state.manhattan_distance(goal),
start_state))

    while open_list:
        _, current = heappop(open_list)

        if current.is_goal(goal):
            return reconstruct_path(current)

        closed_set.add(current.board)

        for neighbor in current.get_moves():
            if neighbor.board in closed_set:
                continue
            cost = neighbor.depth + neighbor.manhattan_distance(goal)
            heappush(open_list, (cost, neighbor))

    return None

def reconstruct_path(state):
    path = []
    while state.parent is not None:
        path.append(state.move)
        state = state.parent
    path.reverse()
    return path

if __name__ == "__main__":
    start = (2, 8, 3,
             1, 6, 4,
             7, 0, 5)
```

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

    print("Sinchana Hemanth (1BM23CS330)")
    solution = a_star(start, goal)
    if solution:
        print(f"Solution found in {len(solution)} moves: {solution}")
    else:
        print("No solution found.")
```

OUTPUT:

```
Sinchana Hemanth (1BM23CS330)
Solution found in 5 moves: ['Up', 'Up', 'Left', 'Down', 'Right']
```