

Lab 3

Q1) Misplace tiles :

```
def mistil(state, goal):
```

```
    count = 0
```

```
    for i in range(3):
```

```
        for j in range(3):
```

```
            if state[i][j] != goal[i][j]:
```

```
                count += 1
```

```
    return count
```

```
def findmin(open_list, goal):
```

```
    minv = float('inf')
```

```
    best_state = None
```

```
    for state in open_list:
```

```
        h = mistil(state['state'], goal)
```

```
        f = state['g'] + h
```

```
        if f < minv:
```

```
            minv = f
```

```
            best_state = state
```

```
    open_list.remove(best_state)
```

```
    return best_state
```

```
def operation(state):
```

```
    next_states = []
```

```
    blank_pos = find_blank_position(state['state'])
```

```
    for move in ['up', 'down', 'left', 'right']:
```

```
        new_state = apply_move(state['state'], blank_pos, move)
```

```
        if new_state:
```

```
            next_states.append({
```

```
                'state': new_state,
```

```
                'parent': state,
```

```

        'move': move,
        'g': state['g'] + 1
    })
    return next_states

```

```

def find_blank_position(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j
    return None

```

```

def apply_move(state, blank_pos, move):
    i, j = blank_pos
    new_state = [row[:] for row in state]
    if move == 'up' and i > 0:
        new_state[i][j], new_state[i - 1][j] = new_state[i - 1][j], new_state[i][j]
    elif move == 'down' and i < 2:
        new_state[i][j], new_state[i + 1][j] = new_state[i + 1][j], new_state[i][j]
    elif move == 'left' and j > 0:
        new_state[i][j], new_state[i][j - 1] = new_state[i][j - 1], new_state[i][j]
    elif move == 'right' and j < 2:
        new_state[i][j], new_state[i][j + 1] = new_state[i][j + 1], new_state[i][j]
    else:
        return None
    return new_state

```

```

def print_state(state):
    for row in state:
        print(' '.join(map(str, row)))

```

```

initial_state = [[2,8,3], [1,6,4], [7,0,5]]
goal_state = [[1,2,3], [8,0,4], [7,6,5]]
open_list = [{'state': initial_state, 'parent': None, 'move': None, 'g': 0}]
visited_states = []

```

```

while open_list:
    best_state = findmin(open_list, goal_state)
    print("Current state:")
    print_state(best_state['state'])
    h = mistil(best_state['state'], goal_state)
    f = best_state['g'] + h
    print(f"g(n): {best_state['g']}, h(n): {h}, f(n): {f}")
    if best_state['move'] is not None:
        print(f"Move: {best_state['move']}")
    print()
    if mistil(best_state['state'], goal_state) == 0:
        goal_state_reached = best_state
        break
    visited_states.append(best_state['state'])
    next_states = operation(best_state)
    for state in next_states:
        if state['state'] not in visited_states:
            open_list.append(state)

```

```

moves = []
while goal_state_reached['move'] is not None:
    moves.append(goal_state_reached['move'])
    goal_state_reached = goal_state_reached['parent']
moves.reverse()

```

```

print("\nMoves to reach the goal state:", moves)

```

```
print("\nGoal state reached:")
```

```
print_state(goal_state)
```

Output :

Current state:

2 8 3

1 6 4

7 0 5

g(n): 0, h(n): 5, f(n): 5

Current state:

2 8 3

1 0 4

7 6 5

g(n): 1, h(n): 3, f(n): 4

Move: up

Current state:

2 0 3

1 8 4

7 6 5

g(n): 2, h(n): 4, f(n): 6

Move: up

Current state:

2 8 3

0 1 4

7 6 5

g(n): 2, h(n): 4, f(n): 6

Move: left

Current state:

0 2 3

1 8 4

7 6 5

$g(n): 3, h(n): 3, f(n): 6$

Move: left

Current state:

1 2 3

0 8 4

7 6 5

$g(n): 4, h(n): 2, f(n): 6$

Move: down

Current state:

1 2 3

8 0 4

7 6 5

$g(n): 5, h(n): 0, f(n): 5$

Move: right

Moves to reach the goal state: ['up', 'up', 'left', 'down', 'right']

Goal state reached:

1 2 3

8 0 4

7 6 5

Q2) Manhattan distance :

```
def manhattan_distance(state, goal):  
    distance = 0  
    for i in range(3):  
        for j in range(3):  
            tile = state[i][j]  
            if tile != 0: # Ignore the blank space (0)  
                # Find the position of the tile in the goal state  
                for r in range(3):  
                    for c in range(3):  
                        if goal[r][c] == tile:  
                            target_row, target_col = r, c  
                            break  
                # Add the Manhattan distance (absolute difference in rows and columns)  
                distance += abs(target_row - i) + abs(target_col - j)  
    return distance  
  
def findmin(open_list, goal):  
    minv = float('inf')  
    best_state = None  
    for state in open_list:  
        h = manhattan_distance(state['state'], goal) # Use Manhattan distance here  
        f = state['g'] + h  
        if f < minv:  
            minv = f  
            best_state = state  
    open_list.remove(best_state)  
    return best_state  
  
def operation(state):  
    next_states = []
```

```

blank_pos = find_blank_position(state['state'])
for move in ['up', 'down', 'left', 'right']:
    new_state = apply_move(state['state'], blank_pos, move)
    if new_state:
        next_states.append({
            'state': new_state,
            'parent': state,
            'move': move,
            'g': state['g'] + 1
        })
return next_states

```

```

def find_blank_position(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j
    return None

```

```

def apply_move(state, blank_pos, move):
    i, j = blank_pos
    new_state = [row[:] for row in state]
    if move == 'up' and i > 0:
        new_state[i][j], new_state[i - 1][j] = new_state[i - 1][j], new_state[i][j]
    elif move == 'down' and i < 2:
        new_state[i][j], new_state[i + 1][j] = new_state[i + 1][j], new_state[i][j]
    elif move == 'left' and j > 0:
        new_state[i][j], new_state[i][j - 1] = new_state[i][j - 1], new_state[i][j]
    elif move == 'right' and j < 2:
        new_state[i][j], new_state[i][j + 1] = new_state[i][j + 1], new_state[i][j]
    else:

```

```

        return None
    return new_state

def print_state(state):
    for row in state:
        print(' '.join(map(str, row)))

# Initial state and goal state
initial_state = [[2,8,3], [1,6,4], [7,0,5]]
goal_state = [[1,2,3], [8,0,4], [7,6,5]]

# Open list and visited states
open_list = [{'state': initial_state, 'parent': None, 'move': None, 'g': 0}]
visited_states = []

while open_list:
    best_state = findmin(open_list, goal_state)

    print("Current state:")
    print_state(best_state['state'])

    h = manhattan_distance(best_state['state'], goal_state) # Using Manhattan distance here
    f = best_state['g'] + h
    print(f"g(n): {best_state['g']}, h(n): {h}, f(n): {f}")

    if best_state['move'] is not None:
        print(f"Move: {best_state['move']}")
        print()
    if h == 0: # Goal is reached if h == 0
        goal_state_reached = best_state
        break

```



```

visited_states.append(best_state['state'])
next_states = operation(best_state)

for state in next_states:
    if state['state'] not in visited_states:
        open_list.append(state)

# Reconstruct the path of moves
moves = []
while goal_state_reached['move'] is not None:
    moves.append(goal_state_reached['move'])
    goal_state_reached = goal_state_reached['parent']
moves.reverse()

print("\nMoves to reach the goal state:", moves)
print("\nGoal state reached:")
print_state(goal_state)

```

Output :

Current state:

2 8 3

1 6 4

7 0 5

$g(n)$: 0, $h(n)$: 5, $f(n)$: 5

Current state:

2 8 3

1 0 4

7 6 5

$g(n)$: 1, $h(n)$: 3, $f(n)$: 4

Move: up

Current state:

2 0 3

1 8 4

7 6 5

$g(n): 2, h(n): 4, f(n): 6$

Move: up

Current state:

2 8 3

0 1 4

7 6 5

$g(n): 2, h(n): 4, f(n): 6$

Move: left

Current state:

0 2 3

1 8 4

7 6 5

$g(n): 3, h(n): 3, f(n): 6$

Move: left

Current state:

1 2 3

0 8 4

7 6 5

$g(n): 4, h(n): 2, f(n): 6$

Move: down

Current state:

1 2 3

8 0 4

7 6 5

$g(n): 5, h(n): 0, f(n): 5$

Move: right

Moves to reach the goal state: ['up', 'up', 'left', 'down', 'right']

Goal state reached:

1 2 3

8 0 4

7 6 5