```
#using misplaced tiles h(n)
import heapq

# Function to count misplaced tiles
def misplaced_tiles(state, goal):
    count = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != goal[i][j] and state[i][j] != 0:
                count += 1
    return count

# Function to get the position of the empty tile (0)
def get_empty_tile_position(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j

# Function to generate new states by moving the empty tile
def generate_new_states(state):
    i, j = get_empty_tile_position(state)
    possible_moves = []
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # Up, Down, Left, Right
    for di, dj in directions:
        new_i, new_j = i + di, j + dj
        if 0 <= new_i < 3 and 0 <= new_j < 3:
            new_state = [row[:] for row in state]
            # Swap the empty tile with the neighboring tile
            new_state[i][j], new_state[new_i][new_j] = new_state[new_i][new_j], new_state[i][j]
            possible_moves.append(new_state)
    return possible_moves

# Function to print the puzzle state in a readable format
def print_puzzle(state):
    for row in state:
        print(' '.join(str(x) if x != 0 else '_' for x in row))
    print()

# A* algorithm with detailed logging
def a_star(start, goal):
    # Priority queue (min-heap) where each entry is (f(n), g(n), state, previous moves)
    queue = []
    heapq.heappush(queue, (misplaced_tiles(start, goal), 0, start, []))

    # Set to store visited states
    visited = set()
    visited.add(tuple(tuple(row) for row in start))

    while queue:
        # Select the node with the lowest f(n)
        f, g, current_state, path = heapq.heappop(queue)

        # Print the current step
        print(f"Step {g}:")
        print(f"g(n) = {g}, h(n) = {f - g}")
```

```
        print("Current puzzle state:")
        print_puzzle(current_state)

        # If current state is the goal, we are done
        if current_state == goal:
            print("Goal state reached!")
            print(f"Total moves: {g}")
            print("Solution path:")
            for step, state in enumerate(path + [current_state], 1):
                print(f"Move {step}:")
                print_puzzle(state)
            return

        # Generate possible next states
        print("Generated new states (possible moves):")
        trial_states = []
        for new_state in generate_new_states(current_state):
            if tuple(tuple(row) for row in new_state) not in visited:
                h = misplaced_tiles(new_state, goal)
                trial_states.append((g + h + 1, g + 1, new_state, h))
                print(f"g(n) = {g + 1}, h(n) = {h}")
                print_puzzle(new_state)

        # Add unvisited states to the queue
        print("Evaluating and choosing the best state based on f(n):")
        for f_new, g_new, state, h_new in trial_states:
            if tuple(tuple(row) for row in state) not in visited:
                heapq.heappush(queue, (f_new, g_new, state, path + [current_state]))
                visited.add(tuple(tuple(row) for row in state))
                print(f"Chosen state with f(n) = {f_new} (g(n) = {g_new}, h(n) = {h_new}):")
                print_puzzle(state)

    print("No solution found")
    return

# Example usage
start_state = [
    [1, 2, 3],
    [4, 0, 6],
    [7, 5, 8]
]

goal_state = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 0]
]

a_star(start_state, goal_state)
```

output:

```
Step 0:
g(n) = 0, h(n) = 2
Current puzzle state:
1 2 3
```

```
4 _ 6
7 5 8

Generated new states (possible moves):
g(n) = 1, h(n) = 3
1 _ 3
4 2 6
7 5 8

g(n) = 1, h(n) = 1
1 2 3
4 5 6
7 _ 8

g(n) = 1, h(n) = 3
1 2 3
_ 4 6
7 5 8

g(n) = 1, h(n) = 3
1 2 3
4 6 _
7 5 8

Evaluating and choosing the best state based on f(n):
Chosen state with f(n) = 4 (g(n) = 1, h(n) = 3):
1 _ 3
4 2 6
7 5 8

Chosen state with f(n) = 2 (g(n) = 1, h(n) = 1):
1 2 3
4 5 6
7 _ 8

Chosen state with f(n) = 4 (g(n) = 1, h(n) = 3):
1 2 3
_ 4 6
7 5 8

Chosen state with f(n) = 4 (g(n) = 1, h(n) = 3):
1 2 3
4 6 _
7 5 8

Step 1:
g(n) = 1, h(n) = 1
Current puzzle state:
1 2 3
4 5 6
7 _ 8

Generated new states (possible moves):
```

```
g(n) = 2, h(n) = 2
1 2 3
4 5 6
_ 7 8

g(n) = 2, h(n) = 0
1 2 3
4 5 6
7 8 _

Evaluating and choosing the best state based on f(n):
Chosen state with f(n) = 4 (g(n) = 2, h(n) = 2):
1 2 3
4 5 6
_ 7 8

Chosen state with f(n) = 2 (g(n) = 2, h(n) = 0):
1 2 3
4 5 6
7 8 _

Step 2:
g(n) = 2, h(n) = 0
Current puzzle state:
1 2 3
4 5 6
7 8 _

Goal state reached!
Total moves: 2
Solution path:
Move 1:
1 2 3
4 _ 6
7 5 8

Move 2:
1 2 3
4 5 6
7 _ 8

Move 3:
1 2 3
4 5 6
7 8 _
```