```python
import heapq

GOAL_STATE = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]
MOVES = [(0, 1), (1, 0), (0, -1), (-1, 0)]

def find_goal_position(tile):
    for i in range(3):
        for j in range(3):
            if GOAL_STATE[i][j] == tile:
                return i, j

def find_empty(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return (i, j)

def h_manhattan_distance(state):
    total_distance = 0
    for i in range(3):
        for j in range(3):
            tile = state[i][j]
            if tile != 0:
                goal_row, goal_col = find_goal_position(tile)
                distance = abs(i - goal_row) + abs(j - goal_col)
                total_distance += distance
    return total_distance

def generate_new_states(state):
    empty_x, empty_y = find_empty(state)
    new_states = []
    for move in MOVES:
        new_x, new_y = empty_x + move[0], empty_y + move[1]
        if 0 <= new_x < 3 and 0 <= new_y < 3:
            new_state = [row[:] for row in state]
            new_state[empty_x][empty_y], new_state[new_x][new_y] = new_state[new_x][new_y], new_state[empty_x][empty_y]
            new_states.append(new_state)
    return new_states

def a_star_search(initial_state):
    priority_queue = []
    heapq.heappush(priority_queue, (0 + h_manhattan_distance(initial_state), 0, initial_state, []))
    visited = set()
    visited.add(tuple(map(tuple, initial_state)))
```

```python
    while priority_queue:
        f, g, current_state, path = heapq.heappop(priority_queue)

        h_value = h_manhattan_distance(current_state)
        f_value = g + h_value

        print(f"Current State at Depth {g}: g(n)={g}, h(n)={h_value}, f(n)={f_value}")
        for row in current_state:
            print(row)
        print()

        if current_state == GOAL_STATE:
            return path + [current_state], g

        for new_state in generate_new_states(current_state):
            state_tuple = tuple(map(tuple, new_state))
            if state_tuple not in visited:
                visited.add(state_tuple)
                heapq.heappush(priority_queue, (g + 1 + h_manhattan_distance(new_state), g + 1, new_state, path + [current_state]))

    return None, None

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

solution_manhattan, total_cost = a_star_search(initial_state)
print(f"Total cost to reach goal: {total_cost}")
```

```
Current State at Depth 0: g(n)=0, h(n)=5, f(n)=5
[2, 8, 3]
[1, 6, 4]
[7, 0, 5]

Current State at Depth 1: g(n)=1, h(n)=4, f(n)=5
[2, 8, 3]
[1, 0, 4]
[7, 6, 5]

Current State at Depth 2: g(n)=2, h(n)=3, f(n)=5
[2, 0, 3]
[1, 8, 4]
[7, 6, 5]

Current State at Depth 3: g(n)=3, h(n)=2, f(n)=5
[0, 2, 3]
[1, 8, 4]
[7, 6, 5]

Current State at Depth 4: g(n)=4, h(n)=1, f(n)=5
[1, 2, 3]
[0, 8, 4]
[7, 6, 5]

Current State at Depth 5: g(n)=5, h(n)=0, f(n)=5
[1, 2, 3]
[8, 0, 4]
[7, 6, 5]

Total cost to reach goal: 5
```

```
import heapq

GOAL_STATE = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]
MOVES = [(0, 1), (1, 0), (0, -1), (-1, 0)]

def find_goal_position(tile):
    for i in range(3):
        for j in range(3):
            if GOAL_STATE[i][j] == tile:
                return i, j

def find_empty(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return (i, j)

def h_manhattan_distance(state):
    total_distance = 0
    for i in range(3):
        for j in range(3):
            tile = state[i][j]
            if tile != 0:
                goal_row, goal_col = find_goal_position(tile)
                distance = abs(i - goal_row) + abs(j - goal_col)
                total_distance += distance
    return total_distance

def generate_new_states(state):
    empty_x, empty_y = find_empty(state)
    new_states = []
    for move in MOVES:
        new_x, new_y = empty_x + move[0], empty_y + move[1]
        if 0 <= new_x < 3 and 0 <= new_y < 3:
            new_state = [row[:] for row in state]
            new_state[empty_x][empty_y], new_state[new_x][new_y] = new_state[new_x][new_y], new_state[empty_x][empty_y]
            new_states.append(new_state)
    return new_states

def a_star_search(initial_state):
    priority_queue = []
    heapq.heappush(priority_queue, (0 + h_manhattan_distance(initial_state), 0, initial_state, []))
    visited = set()
    visited.add(tuple(map(tuple, initial_state)))

    while priority_queue:
        f, g, current_state, path = heapq.heappop(priority_queue)

        h_value = h_manhattan_distance(current_state)
        f_value = g + h_value

        print(f"Current State at Depth {g}: g(n)={g}, h(n)={h_value}, f(n)={f_value}")
        for row in current_state:
            print(row)
        print()

        if current_state == GOAL_STATE:
            return path + [current_state], g

        for new_state in generate_new_states(current_state):
            state_tuple = tuple(map(tuple, new_state))
            if state_tuple not in visited:
                visited.add(state_tuple)
                heapq.heappush(priority_queue, (g + 1 + h_manhattan_distance(new_state), g + 1, new_state, path + [current_state]))

    return None, None

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

solution_manhattan, total_cost = a_star_search(initial_state)
print(f"Total cost to reach goal: {total_cost}")
```

```
Current State at Depth 0: g(n)=0, h(n)=5, f(n)=5
[2, 8, 3]
[1, 6, 4]
[7, 0, 5]

Current State at Depth 1: g(n)=1, h(n)=4, f(n)=5
[2, 8, 3]
[1, 0, 4]
[7, 6, 5]
```

```
Current State at Depth 2: g(n)=2, h(n)=3, f(n)=5
[2, 0, 3]
[1, 8, 4]
[7, 6, 5]

Current State at Depth 3: g(n)=3, h(n)=2, f(n)=5
[0, 2, 3]
[1, 8, 4]
[7, 6, 5]

Current State at Depth 4: g(n)=4, h(n)=1, f(n)=5
[1, 2, 3]
[0, 8, 4]
[7, 6, 5]

Current State at Depth 5: g(n)=5, h(n)=0, f(n)=5
[1, 2, 3]
[8, 0, 4]
[7, 6, 5]

Total cost to reach goal: 5
```