Python code for 8-puzzle A* implementation, to calculate, f(n), considering:
g(n): Depth of the node, h(n): Manhattan Distance

```
import heapq


class PuzzleState:
    def __init__(self, state, empty_tile_pos, g, h, path, level):
        self.state = state
        self.empty_tile_pos = empty_tile_pos  # (row, col)
        self.g = g  # Cost from start to current state
        self.h = h  # Heuristic cost to goal
        self.f = g + h  # Total cost
        self.path = path  # Path taken to reach this state
        self.level = level  # Depth level in the state space


    def __lt__(self, other):
        return self.f < other.f  # Priority queue comparison based on f value


def astar_manhattan_distance(start_state, goal_state):
    directions = {
        (-1, 0): 'Up',
        (1, 0): 'Down',
        (0, -1): 'Left',
        (0, 1): 'Right'
    }
```

```
def calculate_heuristic(state):
    h = 0
    for i in range(3):
        for j in range(3):
            value = state[i][j]
            if value != 0:
                goal_row = (value - 1) // 3
                goal_col = (value - 1) % 3
                h += abs(goal_row - i) + abs(goal_col - j)
    return h


def generate_moves(state, empty_tile_pos):
    moves = []
    row, col = empty_tile_pos
    for (dr, dc), direction in directions.items():
        new_row, new_col = row + dr, col + dc
        if 0 <= new_row < 3 and 0 <= new_col < 3:
            new_state = [list(r) for r in state]  # Deep copy
            # Swap the empty tile with the adjacent tile
            new_state[row][col], new_state[new_row][new_col] =
new_state[new_row][new_col], new_state[row][col]
            moves.append((new_state, (new_row, new_col), direction))
    return moves
```

```
    visited = set()
    start_empty_pos = next((i, j) for i in range(3) for j in range(3) if
start_state[i][j] == 0)
    start_h = calculate_heuristic(start_state)
    start_node = PuzzleState(start_state, start_empty_pos, 0, start_h,
[start_state], 0)

    priority_queue = []
    heapq.heappush(priority_queue, start_node)

    while priority_queue:
        current_node = heapq.heappop(priority_queue)

        # Check if we reached the goal
        if current_node.state == goal_state:
            print("Goal state reached!")
            for step in current_node.path:
                for row in step:
                    print(row)
                print()
            return

        visited.add(tuple(map(tuple, current_node.state)))  # Add current state to
visited

        # Generate possible moves
```

```
        for new_state, new_empty_pos, direction in generate_moves(current_node.state, current_node.empty_tile_pos):
            if tuple(map(tuple, new_state)) not in visited:
                g = current_node.g + 1  # Cost from start
                h = calculate_heuristic(new_state)  # Heuristic
                new_path = current_node.path + [new_state]
                new_node = PuzzleState(new_state, new_empty_pos, g, h, new_path, current_node.level + 1)

                # Print state information
                print(f"Level: {new_node.level}, Direction: {direction}, Heuristic: {new_node.h}")
                for row in new_node.state:
                    print(row)
                print()

                heapq.heappush(priority_queue, new_node)

def main():
    print("Enter initial state (3x3 grid, use 0 for empty tile):")
    start_state = [list(map(int, input().split())) for _ in range(3)]
    print("Enter goal state (3x3 grid, use 0 for empty tile):")
    goal_state = [list(map(int, input().split())) for _ in range(3)]
    astar_manhattan_distance(start_state, goal_state)
if __name__ == "__main__":
    main()
```

Output:

Enter initial state (3x3 grid, use 0 for empty tile):

2 8 3

1 6 4

7 0 5

Enter goal state (3x3 grid, use 0 for empty tile):

1 2 3

8 0 4

7 6 5

Level: 1, Direction: Up, Heuristic: 10

[2, 8, 3]

[1, 0, 4]

[7, 6, 5]

Level: 1, Direction: Left, Heuristic: 10

[2, 8, 3]

[1, 6, 4]

[0, 7, 5]

Level: 1, Direction: Right, Heuristic: 8

[2, 8, 3]

[1, 6, 4]

[7, 5, 0]

Level: 2, Direction: Up, Heuristic: 9

[2, 8, 3]

[1, 6, 0]

[7, 5, 4]

Level: 2, Direction: Up, Heuristic: 11

[2, 8, 3]

[0, 6, 4]

[1, 7, 5]

Level: 2, Direction: Up, Heuristic: 9

[2, 0, 3]

[1, 8, 4]

[7, 6, 5]

Level: 2, Direction: Left, Heuristic: 11

[2, 8, 3]

[0, 1, 4]

[7, 6, 5]

Level: 2, Direction: Right, Heuristic: 9

[2, 8, 3]

[1, 4, 0]

[7, 6, 5]

Level: 3, Direction: Up, Heuristic: 10

[2, 8, 0]

[1, 6, 3]

[7, 5, 4]

Level: 3, Direction: Left, Heuristic: 8

[2, 8, 3]

[1, 0, 6]

[7, 5, 4]

Level: 3, Direction: Left, Heuristic: 8

[0, 2, 3]

[1, 8, 4]

[7, 6, 5]

Level: 3, Direction: Right, Heuristic: 10

[2, 3, 0]

[1, 8, 4]

[7, 6, 5]

Level: 4, Direction: Up, Heuristic: 7

[2, 0, 3]

[1, 8, 6]

[7, 5, 4]

Level: 4, Direction: Down, Heuristic: 7

[2, 8, 3]

[1, 5, 6]

[7, 0, 4]

Level: 4, Direction: Left, Heuristic: 9

[2, 8, 3]

[0, 1, 6]

[7, 5, 4]

Level: 4, Direction: Down, Heuristic: 7

[1, 2, 3]

[0, 8, 4]

[7, 6, 5]

Level: 5, Direction: Left, Heuristic: 6

[0, 2, 3]

[1, 8, 6]

[7, 5, 4]

Level: 5, Direction: Right, Heuristic: 8

[2, 3, 0]

[1, 8, 6]

[7, 5, 4]

Level: 3, Direction: Up, Heuristic: 10

[2, 8, 0]

[1, 4, 3]

[7, 6, 5]

Level: 3, Direction: Down, Heuristic: 8

[2, 8, 3]

[1, 4, 5]

[7, 6, 0]

Level: 5, Direction: Left, Heuristic: 8

[2, 8, 3]

[1, 5, 6]

[0, 7, 4]

Level: 5, Direction: Right, Heuristic: 6

[2, 8, 3]

[1, 5, 6]

[7, 4, 0]

Level: 5, Direction: Down, Heuristic: 8

[1, 2, 3]

[7, 8, 4]

[0, 6, 5]

Level: 5, Direction: Right, Heuristic: 8

[1, 2, 3]

[8, 0, 4]

[7, 6, 5]

Level: 6, Direction: Up, Heuristic: 7

[2, 8, 3]

[1, 5, 0]

[7, 4, 6]

Level: 4, Direction: Left, Heuristic: 7

[2, 8, 3]

[1, 4, 5]

[7, 0, 6]

Level: 6, Direction: Down, Heuristic: 5

[1, 2, 3]

[0, 8, 6]

[7, 5, 4]

Level: 5, Direction: Up, Heuristic: 8

[2, 8, 3]

[1, 0, 5]

[7, 4, 6]

Level: 5, Direction: Left, Heuristic: 8

[2, 8, 3]

[1, 4, 5]

[0, 7, 6]

Level: 7, Direction: Down, Heuristic: 6

[1, 2, 3]

[7, 8, 6]

[0, 5, 4]

Level: 7, Direction: Right, Heuristic: 6

[1, 2, 3]

[8, 0, 6]

[7, 5, 4]

Level: 3, Direction: Up, Heuristic: 12

[0, 8, 3]

[2, 6, 4]

[1, 7, 5]

Level: 3, Direction: Right, Heuristic: 12

[2, 8, 3]

[6, 0, 4]

[1, 7, 5]

Level: 5, Direction: Up, Heuristic: 10

[0, 8, 3]

[2, 1, 6]

[7, 5, 4]

Level: 5, Direction: Down, Heuristic: 10

[2, 8, 3]

[7, 1, 6]

[0, 5, 4]

Level: 6, Direction: Up, Heuristic: 9

[2, 8, 3]

[0, 4, 5]

[1, 7, 6]

Level: 8, Direction: Right, Heuristic: 7

[1, 2, 3]

[7, 8, 6]

[5, 0, 4]

Level: 8, Direction: Up, Heuristic: 7

[1, 0, 3]

[8, 2, 6]

[7, 5, 4]

Level: 8, Direction: Down, Heuristic: 5

[1, 2, 3]

[8, 5, 6]

[7, 0, 4]

Level: 8, Direction: Right, Heuristic: 7

[1, 2, 3]

[8, 6, 0]

[7, 5, 4]

Level: 4, Direction: Down, Heuristic: 11

[2, 3, 4]

[1, 8, 0]

[7, 6, 5]

Level: 6, Direction: Up, Heuristic: 7

[2, 0, 3]

[1, 8, 5]

[7, 4, 6]

Level: 6, Direction: Left, Heuristic: 9

[2, 8, 3]

[0, 1, 5]

[7, 4, 6]

Level: 6, Direction: Right, Heuristic: 7

[2, 8, 3]

[1, 5, 0]

[7, 4, 6]

Level: 6, Direction: Right, Heuristic: 9

[1, 2, 3]

[7, 8, 4]

[6, 0, 5]

Level: 4, Direction: Left, Heuristic: 11

[2, 0, 8]

[1, 4, 3]

[7, 6, 5]

Level: 6, Direction: Up, Heuristic: 9

[2, 8, 3]

[0, 5, 6]

[1, 7, 4]

Level: 7, Direction: Up, Heuristic: 8

[2, 8, 0]

[1, 5, 3]

[7, 4, 6]

Level: 7, Direction: Up, Heuristic: 8

[2, 8, 0]

[1, 5, 3]

[7, 4, 6]

Level: 4, Direction: Left, Heuristic: 11

[2, 0, 8]

[1, 6, 3]

[7, 5, 4]

Level: 6, Direction: Down, Heuristic: 9

[2, 3, 6]

[1, 8, 0]

[7, 5, 4]

Goal state reached!

[2, 8, 3]

[1, 6, 4]

[7, 0, 5]

[2, 8, 3]

[1, 0, 4]

[7, 6, 5]

[2, 0, 3]

[1, 8, 4]

[7, 6, 5]

[0, 2, 3]

[1, 8, 4]

[7, 6, 5]

[1, 2, 3]

[0, 8, 4]

[7, 6, 5]

[1, 2, 3]

[8, 0, 4]

[7, 6, 5]