```
#Manhattan Distance
import heapq

class PuzzleState:
    def __init__(self, board, g=0):
        self.board = board
        self.g = g  # Cost from start to this state
        self.zero_pos = board.index(0)  # Position of the empty space

    def h(self):
        # Calculate the Manhattan distance
        distance = 0
        for i in range(9):
            if self.board[i] != 0:
                target_x, target_y = divmod(self.board[i] - 1, 3)
                current_x, current_y = divmod(i, 3)
                distance += abs(target_x - current_x) + abs(target_y - current_y)
        return distance

    def f(self):
        return self.g + self.h()

    def get_neighbors(self):
        neighbors = []
        x, y = divmod(self.zero_pos, 3)
        directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # Up, Down, Left, Right
        for dx, dy in directions:
            new_x, new_y = x + dx, y + dy
            if 0 <= new_x < 3 and 0 <= new_y < 3:
                new_zero_pos = new_x * 3 + new_y
                new_board = self.board[:]
                # Swap zero with the neighboring tile
                new_board[self.zero_pos], new_board[new_zero_pos] = new_board[new_zero_pos], new_board[self.ze
                neighbors.append(PuzzleState(new_board, self.g + 1))
        return neighbors

def a_star(initial_state, goal_state):
    open_set = []
    heapq.heappush(open_set, (initial_state.f(), 0, initial_state))  # Push (f, unique_id, state)
    came_from = {}
    g_score = {tuple(initial_state.board): 0}

    while open_set:
        current_f, _, current = heapq.heappop(open_set)

        if current.board == goal_state:
            return reconstruct_path(came_from, current)

        for neighbor in current.get_neighbors():
            neighbor_tuple = tuple(neighbor.board)
            tentative_g_score = g_score[tuple(current.board)] + 1

            if neighbor_tuple not in g_score or tentative_g_score < g_score[neighbor_tuple]:
                came_from[neighbor_tuple] = current
                g_score[neighbor_tuple] = tentative_g_score
                # Push (f, unique_id, state)
                heapq.heappush(open_set, (neighbor.f(), neighbor.g, neighbor))  # Using g as a tie-breaker

    return None  # If no solution is found

def reconstruct_path(came_from, current):
    path = []
    while current is not None:
```

```
            path.append(current.board)
            current = came_from.get(tuple(current.board), None)
        return path[::-1]

    # Example usage
    initial_state = PuzzleState([1, 2, 3, 4, 5, 6, 0, 7, 8])
    goal_state = [1, 2, 3, 4, 5, 6, 7, 8, 0]

    solution = a_star(initial_state, goal_state)
    print('Name:Swapnil Sahil','USN:1BM22CS300',sep="\n")
    if solution:
        for step in solution:
            print(step)
    else:
        print("No solution found")
```

```
Name:Swapnil Sahil
USN:1BM22CS300
[1, 2, 3, 4, 5, 6, 0, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 0, 8]
[1, 2, 3, 4, 5, 6, 7, 8, 0]
```