#WRITE THE CODE TO IMPLEMENT A* ALGORITHM :

```
import heapq

class PuzzleState:
    def __init__(self, board, empty_tile, moves=0, previous=None):
        self.board = board
        self.empty_tile = empty_tile  # (row, col) of the empty tile
        self.moves = moves
        self.previous = previous  # to trace the path back

    def __lt__(self, other):
        return self.f() < other.f()

    def f(self):
        # Total cost function (g + h)
        return self.moves + self.heuristic()

    def heuristic(self):
        # Using Manhattan distance as the heuristic
        total_distance = 0
        for i in range(3):
            for j in range(3):
                if self.board[i][j] != 0:  # Skip empty tile
                    # Calculate the target position of the tile
                    target_x = (self.board[i][j] - 1) // 3
                    target_y = (self.board[i][j] - 1) % 3
                    # Calculate Manhattan distance for each tile
                    distance = abs(target_x - i) + abs(target_y - j)
                    total_distance += distance
        return total_distance

    def get_neighbors(self):
        neighbors = []
        row, col = self.empty_tile
        directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]  # Down, Up, Right, Left

        for dr, dc in directions:
            new_row, new_col = row + dr, col + dc
            if 0 <= new_row < 3 and 0 <= new_col < 3:  # Within bounds
                new_board = [list(row) for row in self.board]
                # Swap the empty tile with the adjacent tile
                new_board[row][col], new_board[new_row][new_col] =
new_board[new_row][new_col], new_board[row][col]
                # Create a new PuzzleState for the neighbor
                neighbors.append(PuzzleState(new_board, (new_row, new_col),
self.moves + 1, self))

        return neighbors
```

```
def a_star(start_board):
    # Find the position of the empty tile
    start_tile = next((i, j) for i in range(3) for j in range(3) if start_board[i][j] == 0)
    start_state = PuzzleState(start_board, start_tile)
    # Define the goal state
    goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

    open_set = []
    closed_set = set()
    heapq.heappush(open_set, start_state)

    while open_set:
        current_state = heapq.heappop(open_set)

        if current_state.board == goal_state:
            path = []
            while current_state:
                path.append(current_state.board)
                current_state = current_state.previous
            return path[::-1]  # Return reversed path

        closed_set.add(tuple(map(tuple, current_state.board)))

        for neighbor in current_state.get_neighbors():
            if tuple(map(tuple, neighbor.board)) in closed_set:
                continue
            heapq.heappush(open_set, neighbor)

    return None  # No solution found

def get_user_input():
    print ("SUMIT KUMAR CHAUDHARY (1BM22CS296)")
    print("Enter the 3x3 puzzle board (use 0 for the empty tile):")
    board = []
    for i in range(3):
        row = input(f"Row {i + 1} (space-separated): ").strip().split()
        if len(row) != 3 or any(not num.isdigit() or int(num) < 0 or int(num) > 8 for num in
row):
            print("Invalid input. Please enter numbers between 0 and 8.")
            return None
        board.append(list(map(int, row)))

    if set(num for row in board for num in row) != set(range(9)):
        print("Invalid input. The board must contain numbers 0 through 8 exactly once.")
        return None

    return board

# Example usage:
start_board = get_user_input()
```

```
if start_board is not None:
    solution = a_star(start_board)
    if solution:
        for step in solution:
            for row in step:
                print(row)
            print()
    else:
        print("No solution found.")
```

```
SUMIT KUMAR CHAUDHARY (1BM22CS296)
Enter the 3x3 puzzle board (use 0 for the empty tile):
Row 1 (space-separated): 1 2 3
Row 2 (space-separated): 4 0 5
Row 3 (space-separated): 7 8 6
[1, 2, 3]
[4, 0, 5]
[7, 8, 6]

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

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