TITLE: Write the python program to solve 8-Puzzle problem

CODE:

```
import heapq

# Define the goal state
goal_state = (0, 1, 2, 3, 4, 5, 6, 7, 8)

# Define the movements allowed
moves = [(1, 0), (0, 1), (-1, 0), (0, -1)]  # down, right, up, left

# Define heuristic function (Manhattan distance)
def heuristic(state):
    distance = 0
    for i in range(3):
        for j in range(3):
            tile = state[i*3 + j]
            if tile != 0:
                goal_position = (tile - 1) // 3, (tile - 1) % 3
                distance += abs(goal_position[0] - i) + abs(goal_position[1] -
j)
    return distance

# Define the state class
```

```
class State:
    def __init__(self, board, cost, previous):
        self.board = board
        self.cost = cost
        self.previous = previous

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

# Function to generate possible next states
def generate_next_states(current_state):
    next_states = []
    empty_index = current_state.board.index(0)
    empty_row, empty_col = empty_index // 3, empty_index % 3
    for move in moves:
        new_row, new_col = empty_row + move[0], empty_col +
move[1]
        if 0 <= new_row < 3 and 0 <= new_col < 3:
            new_board = list(current_state.board)
            new_board[empty_row * 3 + empty_col],
new_board[new_row * 3 + new_col] = \
                new_board[new_row * 3 + new_col],
new_board[empty_row * 3 + empty_col]
            next_states.append(State(tuple(new_board),
current_state.cost + 1, current_state))
```

```
    return next_states

# A* search algorithm
def astar(start_state):
    visited = set()
    priority_queue = []
    heapq.heappush(priority_queue, start_state)

    while priority_queue:
        current_state = heapq.heappop(priority_queue)

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

        if current_state.board not in visited:
            visited.add(current_state.board)
            next_states = generate_next_states(current_state)
            for next_state in next_states:
                heapq.heappush(priority_queue, next_state)
```

```
    return None

# Main function
def main():
    # Example initial state
    initial_state = (1, 0, 3, 4, 5, 6, 2, 7, 8)

    start_state = State(initial_state, 0, None)
    solution = astar(start_state)

    if solution:
        print("Solution found in", len(solution) - 1, "steps:")
        for step, state in enumerate(solution):
            print("Step", step, ":")
            print(state[0], state[1], state[2])
            print(state[3], state[4], state[5])
            print(state[6], state[7], state[8])
            print()
    else:
        print("No solution found.")

if __name__ == "__main__":
    main()
```

OUTPUT:

IDLE Shell 3.11.8

File Edit Shell Debug Options Window Help

```
Python 3.11.8 (tags/v3.11.8:db85d51, Feb  6 2024, 22:03:32) [MSC v.1937 64 bit (AM
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:\python\8 puzzle.py
Solution found in 25 steps:
Step 0 :
1 0 3
4 5 6
2 7 8

Step 1 :
1 5 3
4 0 6
2 7 8

Step 2 :
1 5 3
0 4 6
2 7 8

Step 3 :
1 5 3
2 4 6
0 7 8

Step 4 :
1 5 3
2 4 6
7 0 8

Step 5 :
1 5 3
2 0 6
7 4 8

Step 6 :
1 5 3
2 6 0
7 4 8

Step 7 :
1 5 0
2 6 3
7 4 8

Step 8 :
1 0 5
2 6 3
7 4 8
```