```python
import heapq

# Manhattan Distance Heuristic
def manhattan_distance(state):
    goal_state = ((1, 2, 3), (4, 5, 6), (7, 8, 0))  # Goal state as tuple of tuples
    distance = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0:
                # Find the goal position of each tile
                goal_pos = divmod(state[i][j] - 1, 3)
                distance += abs(i - goal_pos[0]) + abs(j - goal_pos[1])
    return distance

# Check if the current state is the goal state
def is_goal(state):
    return state == ((1, 2, 3), (4, 5, 6), (7, 8, 0))

# Find the position of the blank (0) in the puzzle
def find_blank(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j
```

```python
# Generate possible moves from the current state
def get_neighbors(state):
    neighbors = []
    x, y = find_blank(state)
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # Up, Down, Left, Right

    for dx, dy in directions:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_state = [list(row) for row in state]  # Deep copy the state (convert back to list of lists)
            new_state[x][y], new_state[nx][ny] = new_state[nx][ny], new_state[x][y]
            neighbors.append(tuple(tuple(row) for row in new_state))  # Convert back to tuple of tuples

    return neighbors

# A* algorithm implementation
def a_star(initial_state):
    print("\n--- A* Solution ---")

    # Convert initial state to tuple of tuples
    initial_state_tuple = tuple(tuple(row) for row in initial_state)

    open_list = []
    closed_set = set()
    g_score = {initial_state_tuple: 0}
    f_score = {initial_state_tuple: manhattan_distance(initial_state_tuple)}
    came_from = {}

    heapq.heappush(open_list, (f_score[initial_state_tuple], initial_state_tuple))
```

```python
    while open_list:
        _, current = heapq.heappop(open_list)

        # Print the current state and the associated scores
        current_g = g_score[current]
        current_h = manhattan_distance(current)
        current_f = current_g + current_h

        print("\nCurrent State:")
        display_state(current)
        print(f"g(n) = {current_g}, h(n) = {current_h}, f(n) = {current_f}")

        if is_goal(current):
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(initial_state_tuple)
            return path[::-1]  # Return the path from start to goal

        closed_set.add(current)

        for neighbor in get_neighbors(current):
            if neighbor in closed_set:
                continue

            tentative_g_score = g_score[current] + 1

            if neighbor not in g_score or tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
```

```python
# Utility function to display the state
def display_state(state):
    for row in state:
        print(" ".join(str(x) if x != 0 else " " for x in row))
    print()

# Main function
if __name__ == "__main__":
    print("Name : Vyom Gupta")
    print("USN  : 1BM22CS333\n")

    print("Enter the initial state of the 8-puzzle (row by row, use 0 for blank):")
    initial_state = []
    for i in range(3):
        row = list(map(int, input(f"Row {i + 1}: ").split()))
        initial_state.append(row)

    print("\nInitial State:")
    display_state(initial_state)

    # Solve using A*
    a_star_solution = a_star(initial_state)
    if a_star_solution:
        print("\nA* Solution Steps:")
        for step in a_star_solution:
            display_state(step)
    else:
        print("No solution found using A*.")
```

```
Name : Vyom Gupta
USN  : 1BM22CS333

Enter the initial state of the 8-puzzle (row by row, use 0 for blank):
Row 1: 1 2 3
Row 2: 4 5 6
Row 3: 7 0 8

Initial State:
1 2 3
4 5 6
7   8


--- A* Solution ---

Current State:
1 2 3
4 5 6
7   8

g(n) = 0, h(n) = 1, f(n) = 1

Current State:
1 2 3
4 5 6
7 8

g(n) = 1, h(n) = 0, f(n) = 1
```

```
A* Solution Steps:
1 2 3
4 5 6
7   8

1 2 3
4 5 6
7 8
```