```
import heapq

class Node:
    def __init__(self, state, parent=None, g=0, h=0):
        self.state = state
        self.parent = parent
        self.g = g
        self.h = h
        self.f = g + h

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

def get_zero_position(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return (i, j)

def get_neighbors(state):
    neighbors = []
    zero_pos = get_zero_position(state)
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    for direction in directions:
        new_pos = (zero_pos[0] + direction[0], zero_pos[1] + direction[1])
        if 0 <= new_pos[0] < 3 and 0 <= new_pos[1] < 3:
            new_state = [list(row) for row in state]
            new_state[zero_pos[0]][zero_pos[1]], new_state[new_pos[0]][new_pos[1]] = \
                new_state[new_pos[0]][new_pos[1]], new_state[zero_pos[0]][zero_pos[1]]
            neighbors.append(new_state)
    return neighbors

def manhattan_distance(state, goal_state):
    distance = 0
    for i in range(3):
        for j in range(3):
            value = state[i][j]
            if value != 0:
                goal_x, goal_y = divmod(value - 1, 3)
                distance += abs(goal_x - i) + abs(goal_y - j)
    return distance

def a_star(start_state, goal_state):
    open_list = []
    closed_set = set()
    start_node = Node(start_state, None, 0, manhattan_distance(start_state, goal_state))
    heapq.heappush(open_list, start_node)

    while open_list:
        current_node = heapq.heappop(open_list)

        print("\nCurrent Node:")
        for row in current_node.state:
            print(row)
        print(f"g: {current_node.g}, h: {current_node.h}, f: {current_node.f}")

        if current_node.state == goal_state:
            path = []
            while current_node:
                path.append(current_node.state)
                current_node = current_node.parent
            return path[::-1]

        closed_set.add(tuple(map(tuple, current_node.state)))

        for neighbor in get_neighbors(current_node.state):
            if tuple(map(tuple, neighbor)) in closed_set:
                continue

            g = current_node.g + 1
            h = manhattan_distance(neighbor, goal_state)
            neighbor_node = Node(neighbor, current_node, g, h)
            heapq.heappush(open_list, neighbor_node)

            print("Neighbors:")
```

```
            for row in neighbor:
                print(row)
            print(f"g: {g}, h: {h}, f: {g + h}")

    return None

def get_matrix_input(prompt):
    print(prompt)
    matrix = []
    for i in range(3):
        row = list(map(int, input(f"Enter row {i + 1} (3 numbers separated by spaces): ").strip().split()))
        if len(row) != 3:
            raise ValueError("Each row must contain exactly 3 numbers.")
        matrix.append(row)
    return matrix

try:
    initial_state = get_matrix_input("Enter the initial state of the 8-puzzle:")
    goal_state = get_matrix_input("Enter the goal state of the 8-puzzle:")

    solution = a_star(initial_state, goal_state)
    if solution:
        print("\nSolution found!")
        for step in solution:
            for row in step:
                print(row)
            print("---")
    else:
        print("No solution exists.")
except ValueError as e:
    print(f"Error: {e}")
```