import heapq  
  
# Goal state for the 8 puzzle  
goal\_state = [[1, 2, 3],  
              [4, 5, 6],  
              [7, 8, 0]]  # 0 is the blank space  
  
# Directions: up, down, left, right  
moves = [(1, 0), (-1, 0), (0, 1), (0, -1)]  
  
  
def manhattan\_distance(state):  
    """Calculate Manhattan distance heuristic for a given state."""  
    distance = 0  
    for i in range(3):  
        for j in range(3):  
            value = state[i][j]  
            if value != 0:  
                target\_x = (value - 1) // 3  
                target\_y = (value - 1) % 3  
                distance += abs(i - target\_x) + abs(j - target\_y)  
    return distance  
  
  
def find\_blank(state):  
    """Find the position of blank (0) in the puzzle."""  
    for i in range(3):  
        for j in range(3):  
            if state[i][j] == 0:  
                return i, j  
  
  
def state\_to\_tuple(state):  
    """Convert list state to tuple (for hashing in sets)."""  
    return tuple(tuple(row) for row in state)  
  
  
def a\_star(start\_state):  
    """A\* algorithm to solve 8-puzzle."""  
    start\_h = manhattan\_distance(start\_state)  
    pq = [(start\_h, 0, start\_state, [])]  # (f, g, state, path)  
    visited = set()  
  
    while pq:  
        f, g, state, path = heapq.heappop(pq)  
  
        if state == goal\_state:  
            return path + [state]  
  
        visited.add(state\_to\_tuple(state))  
        x, y = find\_blank(state)  
  
        for dx, dy in moves:  
            new\_x, new\_y = x + dx, y + dy  
            if 0 <= new\_x < 3 and 0 <= new\_y < 3:  
                new\_state = [row[:] for row in state]  # deep copy  
                # Swap blank with neighbor  
                new\_state[x][y], new\_state[new\_x][new\_y] = new\_state[new\_x][new\_y], new\_state[x][y]  
  
                if state\_to\_tuple(new\_state) not in visited:  
                    h = manhattan\_distance(new\_state)  
                    heapq.heappush(pq, (g + 1 + h, g + 1, new\_state, path + [state]))  
  
    return None  # No solution found  
  
  
# Example usage:  
start\_state = [[1, 2, 3],  
               [4, 5, 6],  
               [0, 7, 8]]  
  
solution = a\_star(start\_state)  
  
print("Steps to reach the goal:")  
for step in solution:  
    for row in step:  
        print(row)  
    print()

Output:

