```
from heapq import heappop, heappush

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

class Node:
    def __init__(self, state, parent=None, cost=0):
        self.state = state
        self.parent = parent
        self.cost = cost
        self.priority = self.cost + self.manhattan_distance()

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

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

    def get_neighbors(self):
        neighbors = []
        x, y = [(ix, iy) for ix, row in enumerate(self.state) for iy, i in
enumerate(row) if i == 0][0]
        directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < 3 and 0 <= ny < 3:
                new_state = [row[:] for row in self.state]
                new_state[x][y], new_state[nx][ny] = new_state[nx][ny], new_state[x][y]
                neighbors.append(Node(new_state, self, self.cost + 1))
        return neighbors

def print_solution_path(node):
    path = []
    while node:
        path.append(node.state)
        node = node.parent
```

```
    for step in path[::-1]:
        for row in step:
            print(row)
        print()

def a_star(initial_state):
    frontier = []
    heappush(frontier, Node(initial_state))
    explored = set()
    visited_states = 0

    while frontier:
        current_node = heappop(frontier)
        visited_states += 1

        if current_node.state == goal_state:
            print("Solution path:")
            print_solution_path(current_node)
            print("Total visited states:", visited_states)
            return current_node.cost

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

        for neighbor in current_node.get_neighbors():
            if tuple(map(tuple, neighbor.state)) not in explored:
                heappush(frontier, neighbor)

    print("No solution found.")
    return None

initial_state = [[1, 2, 3], [4, 0, 6], [7, 5, 8]]
result = a_star(initial_state)
```

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

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

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

Total visited states: 3
```