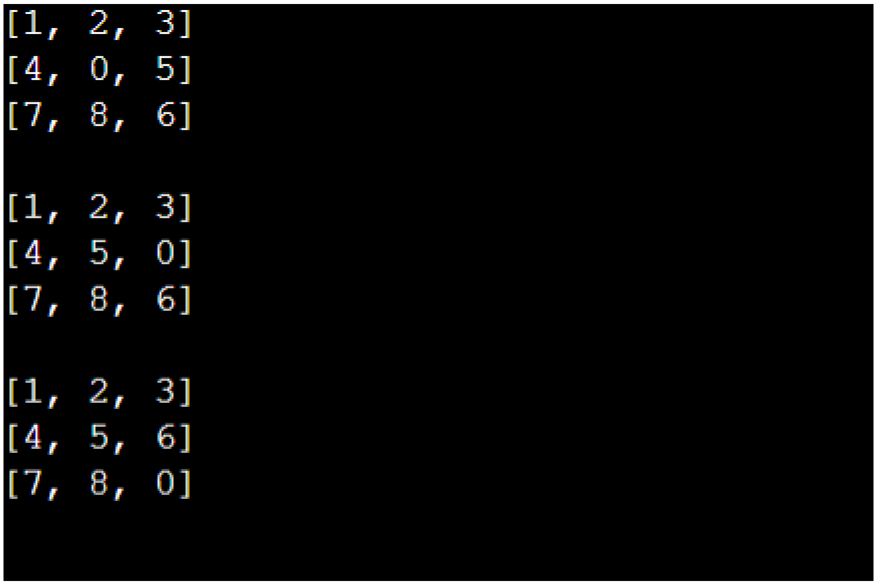
A* implementation(8 Puzzle problem)

```
import heapq


def manhattan_distance(state, goal_state):
    distance = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0:
                goal_pos = [(row.index(state[i][j]), idx) for idx, row in enumerate(goal_state) if state[i][j] in row][0]
                distance += abs(i - goal_pos[1]) + abs(j - goal_pos[0])
    return distance


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

def move_tile(state, direction, empty_tile):
    i, j = empty_tile
    new_state = [row[:] for row in state]
    if direction == "up" and i > 0:
        new_state[i][j], new_state[i-1][j] = new_state[i-1][j], new_state[i][j]
    elif direction == "down" and i < 2:
        new_state[i][j], new_state[i+1][j] = new_state[i+1][j], new_state[i][j]
    elif direction == "left" and j > 0:
        new_state[i][j], new_state[i][j-1] = new_state[i][j-1], new_state[i][j]
    elif direction == "right" and j < 2:
        new_state[i][j], new_state[i][j+1] = new_state[i][j+1], new_state[i][j]
    return new_state


def print_state(state):
    for row in state:
        print(row)
    print()

def a_star(start, goal):

    priority_queue = []
    heapq.heappush(priority_queue, (0, start))
    visited = set()
    visited.add(str(start))

    parent = {}
    parent[str(start)] = None


    g_cost = {}
    g_cost[str(start)] = 0

    directions = ["up", "down", "left", "right"]

    while priority_queue:
        current_cost, current_state = heapq.heappop(priority_queue)


        if current_state == goal:
            return reconstruct_path(parent, current_state)

        empty_tile = get_empty_tile_position(current_state)


        for direction in directions:
            new_state = move_tile(current_state, direction, empty_tile)

            if str(new_state) not in visited:
                visited.add(str(new_state))
                parent[str(new_state)] = current_state
                g_cost[str(new_state)] = g_cost[str(current_state)] + 1
                f_cost = g_cost[str(new_state)] + manhattan_distance(new_state, goal)
                heapq.heappush(priority_queue, (f_cost, new_state))

    return None

def reconstruct_path(parent, state):
    path = []
    while state is not None:
        path.append(state)
        state = parent[str(state)]
    return path[::-1]
```

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

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

solution = a_star(start_state, goal_state)

if solution:
    for step in solution:
        print_state(step)
else:
    print("No solution found.")
```



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

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

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