**LAB-04**

**Implement A\* algorithm using - a) Number of misplaced tiles**

**b) Manhattan Distance**

import heapq

GOAL\_STATE = ((1, 2, 3),

              (8, 0, 4),

              (7, 6, 5))

def misplaced\_tile(state):

    misplaced = 0

    for i in range(3):

        for j in range(3):

            if state[i][j] != 0 and state[i][j] != GOAL\_STATE[i][j]:

                misplaced += 1

    return misplaced

def find\_blank(state):

    for i in range(3):

        for j in range(3):

            if state[i][j] == 0:

                return i, j

def generate\_neighbors(state):

    neighbors = []

    x, y = find\_blank(state)

    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    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]

            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))

    return neighbors

def reconstruct\_path(came\_from, current):

    path = [current]

    while current in came\_from:

        current = came\_from[current]

        path.append(current)

    path.reverse()

    return path

def a\_star(start):

    open\_list = []

    heapq.heappush(open\_list, (0 + misplaced\_tile(start), 0, start))

    g\_score = {start: 0}

    came\_from = {}

    visited = set()

    while open\_list:

        \_, g, current = heapq.heappop(open\_list)

        if current == GOAL\_STATE:

            path = reconstruct\_path(came\_from, current)

            return path, g

        visited.add(current)

        for neighbor in generate\_neighbors(current):

            if neighbor in visited:

                continue

            tentative\_g = g\_score[current] + 1

            if tentative\_g < g\_score.get(neighbor, float('inf')):

                came\_from[neighbor] = current

                g\_score[neighbor] = tentative\_g

                f\_score = tentative\_g + misplaced\_tile(neighbor)  # f(n) = g(n) + h(n)

                heapq.heappush(open\_list, (f\_score, tentative\_g, neighbor))

    return None, None

def print\_state(state):

    for row in state:

        print(row)

    print()

if \_\_name\_\_ == "\_\_main\_\_":

    start\_state = ((2, 8, 3),

                   (1, 6, 4),

                   (7, 0, 5))

    print("Initial State:")

    print\_state(start\_state)

    print("Goal State:")

    print\_state(GOAL\_STATE)

    solution, cost = a\_star(start\_state)

    if solution:

        print(f"Goal state achieved with cost: {cost}")

        print("Steps:")

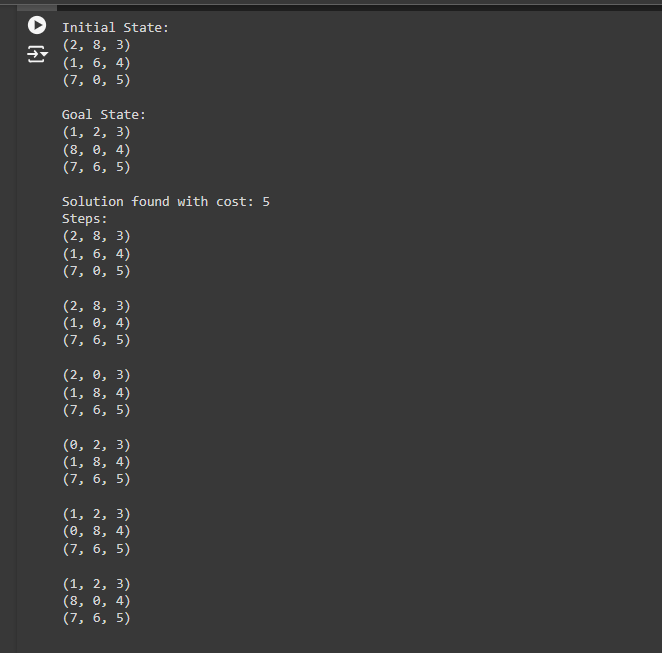
        for step in solution:

            print\_state(step)

    else:

        print("No solution found.")

**OUTPUT:**

****