LAB 3

8 puzzle using IDDFS

from collections import deque

import copy

# Puzzle size

N = 3

# Directions: (delta\_row, delta\_col)

DIRECTIONS = {

    'Up': (-1, 0),

    'Down': (1, 0),

    'Left': (0, -1),

    'Right': (0, 1)

}

# Convert 2D list state to tuple of tuples for hashing in sets

def list\_to\_tuple(state):

    return tuple(tuple(row) for row in state)

# Find the blank tile (0) position in the 2D list state

def find\_blank(state):

    for i in range(N):

        for j in range(N):

            if state[i][j] == 0:

                return i, j

    return None

# Generate all valid successor states (as 2D lists) from the current state

def get\_successors(state):

    blank\_row, blank\_col = find\_blank(state)

    successors = []

    for action, (dr, dc) in DIRECTIONS.items():

        new\_row, new\_col = blank\_row + dr, blank\_col + dc

        if 0 <= new\_row < N and 0 <= new\_col < N:

            new\_state = copy.deepcopy(state)

            # Swap blank tile with adjacent tile

            new\_state[blank\_row][blank\_col], new\_state[new\_row][new\_col] = new\_state[new\_row][new\_col], new\_state[blank\_row][blank\_col]

            successors.append((new\_state, action))

    return successors

# Recursive Depth-Limited Search (DLS)

def dls(current\_state, goal\_state, depth, path, visited):

    if current\_state == goal\_state:

        return path

    if depth == 0:

        return None

    visited.add(list\_to\_tuple(current\_state))

    for neighbor, action in get\_successors(current\_state):

        neighbor\_tuple = list\_to\_tuple(neighbor)

        if neighbor\_tuple not in visited:

            result = dls(neighbor, goal\_state, depth - 1, path + [action], visited)

            if result is not None:

                return result

    return None

# Iterative Deepening DFS (IDDFS)

def iddfs(start, goal, max\_depth=50):

    for depth in range(max\_depth):

        visited = set()

        result = dls(start, goal, depth, [], visited)

        if result is not None:

            return result

    return None

# Take 2D array input from user

def get\_input\_state(prompt):

    print(prompt)

    state = []

    for i in range(N):

        while True:

            try:

                row = input(f"Enter row {i+1} (3 numbers space-separated, use 0 for blank): ").strip().split()

                if len(row) != N:

                    raise ValueError("Incorrect number of elements.")

                row = [int(num) for num in row]

                state.append(row)

                break

            except ValueError as e:

                print("Invalid input, please enter exactly 3 integers separated by spaces.")

    return state

def main():

    print("8 Puzzle Solver using IDDFS")

    start = get\_input\_state("Enter START state:")

    goal = get\_input\_state("Enter GOAL state:")

    print("\nSolving...\n")

    solution = iddfs(start, goal)

    if solution is not None:

        print("Solution found!")

        print("Moves:", ' -> '.join(solution))

        print(f"Number of steps: {len(solution)}")

    else:

        print("No solution found within depth limit.")

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

    main()

output:

