LAB 2

A\* BFS

from collections import deque

# Goal state

goal\_state = [[1, 2, 3],

[4, 5, 6],

[7, 8, 0]]

# Move directions

moves = {'Up': (-1, 0), 'Down': (1, 0), 'Left': (0, -1), 'Right': (0, 1)}

def find\_zero(state):

for i in range(3):

for j in range(3):

if state[i][j] == 0:

return i, j

def is\_valid(x, y):

return 0 <= x < 3 and 0 <= y < 3

def deepcopy(state):

return [row[:] for row in state]

def state\_to\_tuple(state):

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

def bfs(start):

queue = deque([(start, [], find\_zero(start))])

visited = set()

visited.add(state\_to\_tuple(start))

while queue:

state, path, (zx, zy) = queue.popleft()

if state == goal\_state:

return path

for move, (dx, dy) in moves.items():

nx, ny = zx + dx, zy + dy

if is\_valid(nx, ny):

new\_state = deepcopy(state)

new\_state[zx][zy], new\_state[nx][ny] = new\_state[nx][ny], new\_state[zx][zy]

state\_tuple = state\_to\_tuple(new\_state)

if state\_tuple not in visited:

visited.add(state\_tuple)

queue.append((new\_state, path + [move], (nx, ny)))

return None

def is\_solvable(puzzle):

flat = [num for row in puzzle for num in row if num != 0]

inversions = sum(flat[i] > flat[j] for i in range(len(flat)) for j in range(i+1, len(flat)))

return inversions % 2 == 0

# ---------------------- MAIN ----------------------

print("Enter your 3x3 puzzle row by row:")

puzzle = []

for i in range(3):

row = list(map(int, input(f"Enter row {i+1}: ").split()))

puzzle.append(row)

print("\nInitial state you entered:")

for row in puzzle:

print(row)

print("\nSolving...")

if not is\_solvable(puzzle):

print("No solution exists for this puzzle.")

else:

path = bfs(puzzle)

if path:

print(f"Solution found in {len(path)} moves")

print("Sequence of moves:", " -> ".join(path))

else:

print("No solution found.")

