**LAB – 2**

1. **Implement 8 puzzle problem using dfs and ids algorithm.**

**DFS algorithm:**

from copy import deepcopy

# Define goal state

GOAL\_STATE = [[1, 2, 3],

[4, 5, 6],

[7, 8, 0]]

# Directions to move: (row, col)

DIRECTIONS = {

'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\_goal(state):

return state == GOAL\_STATE

def serialize(state):

"""Convert 2D state into a tuple for hashing."""

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

def get\_neighbors(state):

neighbors = []

x, y = find\_zero(state)

for direction, (dx, dy) in DIRECTIONS.items():

nx, ny = x + dx, y + dy

if 0 <= nx < 3 and 0 <= ny < 3:

new\_state = deepcopy(state)

# Swap 0 with neighbor

new\_state[x][y], new\_state[nx][ny] = new\_state[nx][ny], new\_state[x][y]

neighbors.append((new\_state, direction))

return neighbors

def dfs(start\_state):

stack = [(start\_state, [])] # (current\_state, path\_to\_state)

visited = set()

while stack:

state, path = stack.pop()

state\_serial = serialize(state)

if state\_serial in visited:

continue

visited.add(state\_serial)

if is\_goal(state):

return path

for neighbor, move in get\_neighbors(state):

stack.append((neighbor, path + [move]))

return None # No solution found

# Example usage

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

[4, 5, 6],

[0, 7, 8]]

solution = dfs(start\_state)

if solution:

print("Solution found:")

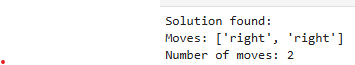
print("Moves:", solution)

print("Number of moves:", len(solution))

else:

print("No solution found.")

**OUTPUT** :



**IDS algorithm :**

from copy import deepcopy

GOAL\_STATE = [[1, 2, 3],

[4, 5, 6],

[7, 8, 0]]

DIRECTIONS = {

'up': (-1, 0),

'down': (1, 0),

'left': (0, -1),

'right': (0, 1)

}

def serialize(state):

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

def is\_goal(state):

return state == GOAL\_STATE

def find\_zero(state):

for i in range(3):

for j in range(3):

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

return i, j

def get\_neighbors(state):

neighbors = []

x, y = find\_zero(state)

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

nx, ny = x + dx, y + dy

if 0 <= nx < 3 and 0 <= ny < 3:

new\_state = deepcopy(state)

new\_state[x][y], new\_state[nx][ny] = new\_state[nx][ny], new\_state[x][y]

neighbors.append((new\_state, move))

return neighbors

def dls(state, path, depth\_limit, visited):

if is\_goal(state):

return path

if depth\_limit <= 0:

return None

visited.add(serialize(state))

for neighbor, move in get\_neighbors(state):

serial = serialize(neighbor)

if serial not in visited:

result = dls(neighbor, path + [move], depth\_limit - 1, visited)

if result is not None:

return result

visited.remove(serialize(state))

return None

def ids(start\_state, max\_depth=50):

for depth in range(max\_depth):

visited = set()

result = dls(start\_state, [], depth, visited)

if result is not None:

return result

return None

# Example start state

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

[4, 5, 6],

[7, 0, 8]]

solution = ids(start\_state)

if solution:

print("Solution found!")

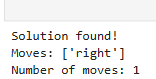
print("Moves:", solution)

print("Number of moves:", len(solution))

else:

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

**OUTPUT:**

****