8 Puzzle using DFS and BFS

i) DFS :

cnt = 0;

def print\_state(in\_array):

global cnt cnt += 1

for row in in\_array:

print(' '.join(str(num) for num in row))

print() # Print a blank line for better readability

def helper(goal, in\_array, row, col, vis):

# Mark the current position as visited vis[row][col] = 1

drow = [-1, 0, 1, 0] # Directions for row movements: up, right, down, left dcol = [0, 1, 0, -1] # Directions for column movements

dchange = ['U', 'R', 'D', 'L']

# Print the current state print("Current state:") print\_state(in\_array)

# Check if the current state is the goal state if in\_array == goal:

print\_state(in\_array) print(f"Number of states : {cnt}") return True

# Explore all possible directions for i in range(4):

nrow = row + drow[i] ncol = col + dcol[i]

# Check if the new position is within bounds and not visited

if 0 <= nrow < len(in\_array) and 0 <= ncol < len(in\_array[0]) and not vis[nrow][ncol]: # Make the move (swap the empty space with the adjacent tile)

print(f"Took a {dchange[i]} move")

in\_array[row][col], in\_array[nrow][ncol] = in\_array[nrow][ncol], in\_array[row][col]

# Recursive call

if helper(goal, in\_array, nrow, ncol, vis):

return True

# Backtrack (undo the move)

in\_array[row][col], in\_array[nrow][ncol] = in\_array[nrow][ncol], in\_array[row][col]

# Mark the position as unvisited before returning vis[row][col] = 0

return False

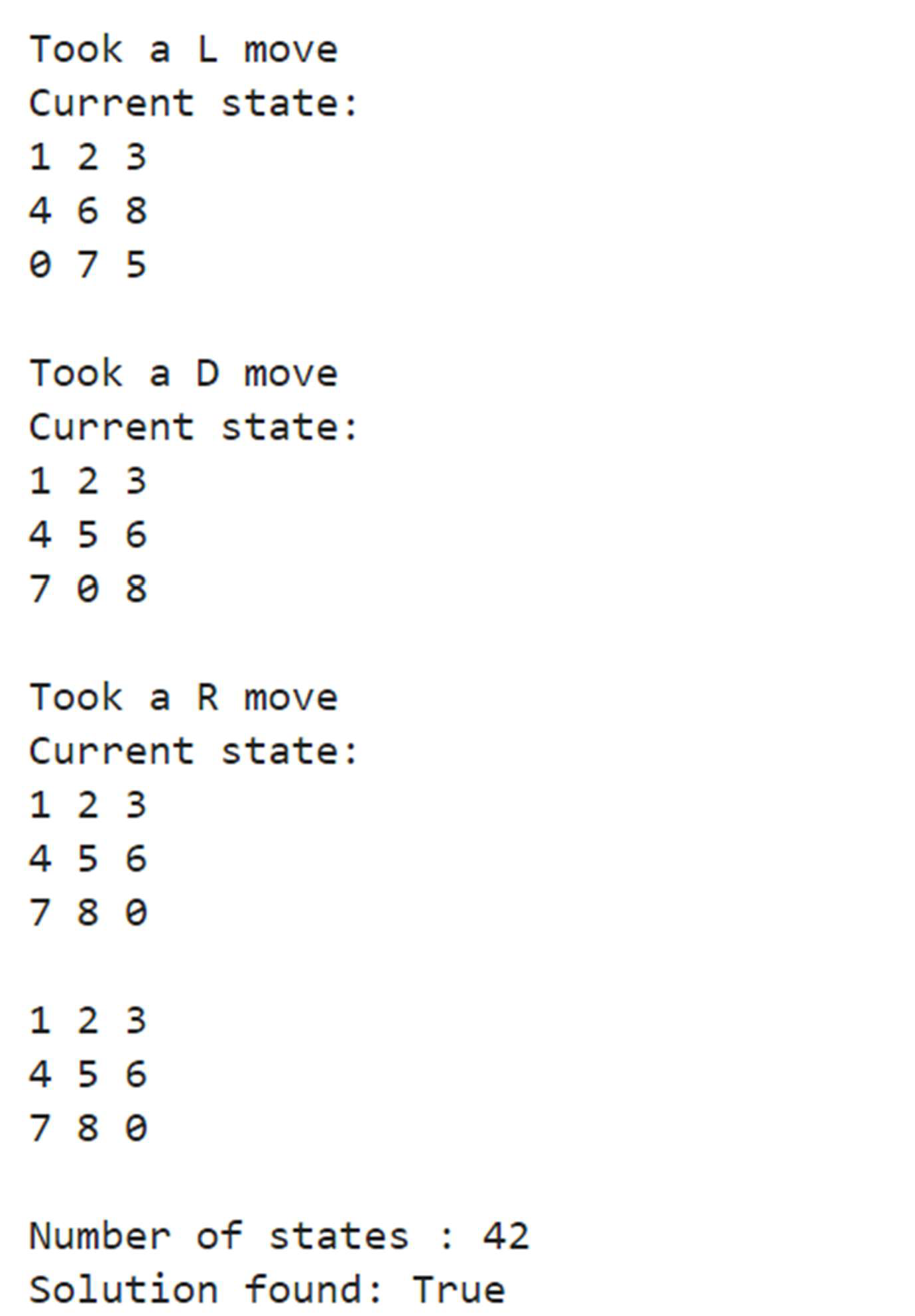
# Example usage

initial\_state = [[1, 2, 3], [0, 4, 6], [7, 5, 8]] # 0 represents the empty space

goal\_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

visited = [[0] \* 3 for \_ in range(3)] # 3x3 visited matrix empty\_row, empty\_col = 1, 0 # Initial position of the empty space

found\_solution = helper(goal\_state, initial\_state, empty\_row, empty\_col, visited) print("Solution found:", found\_solution)

Output :

1. BFS :

from collections import deque GOAL\_STATE = (1, 2, 3, 4, 5, 6, 7, 8, 0)

def ﬁnd\_empty(state):

return state.index(0)

def get\_neighbors(state):

neighbors = []

empty\_index = ﬁnd\_empty(state) row, col = divmod(empty\_index, 3)

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

for dr, dc in directions:

new\_row, new\_col = row + dr, col + dc

if 0 <= new\_row < 3 and 0 <= new\_col < 3:

new\_index = new\_row \* 3 + new\_col new\_state = list(state)

new\_state[empty\_index], new\_state[new\_index] = new\_state[new\_index], new\_state[empty\_index]

neighbors.append(tuple(new\_state)) return neighbors

def bfs(initial\_state):

queue = deque([(initial\_state, [])]) visited = set() visited.add(initial\_state)

visited\_count = 1 # Initialize visited count while queue:

current\_state, path = queue.popleft() if current\_state == GOAL\_STATE:

return path, visited\_count # Return path and count for neighbor in get\_neighbors(current\_state):

if neighbor not in visited:

visited.add(neighbor) queue.append((neighbor, path + [neighbor])) visited\_count += 1 # Increment visited count

return None, visited\_count # Return count if no solution found

def input\_start\_state():

print("Enter the starting state as 9 numbers (0 for the empty space):") input\_state = input("Format: 1 2 3 4 5 6 7 8 0\n")

numbers = list(map(int, input\_state.split()))

if len(numbers) != 9 or set(numbers) != set(range(9)):

print("Invalid input. Please enter numbers from 0 to 8 with no duplicates.") return input\_start\_state()

return tuple(numbers)

def print\_matrix(state):

for i in range(0, 9, 3): print(state[i:i+3])

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

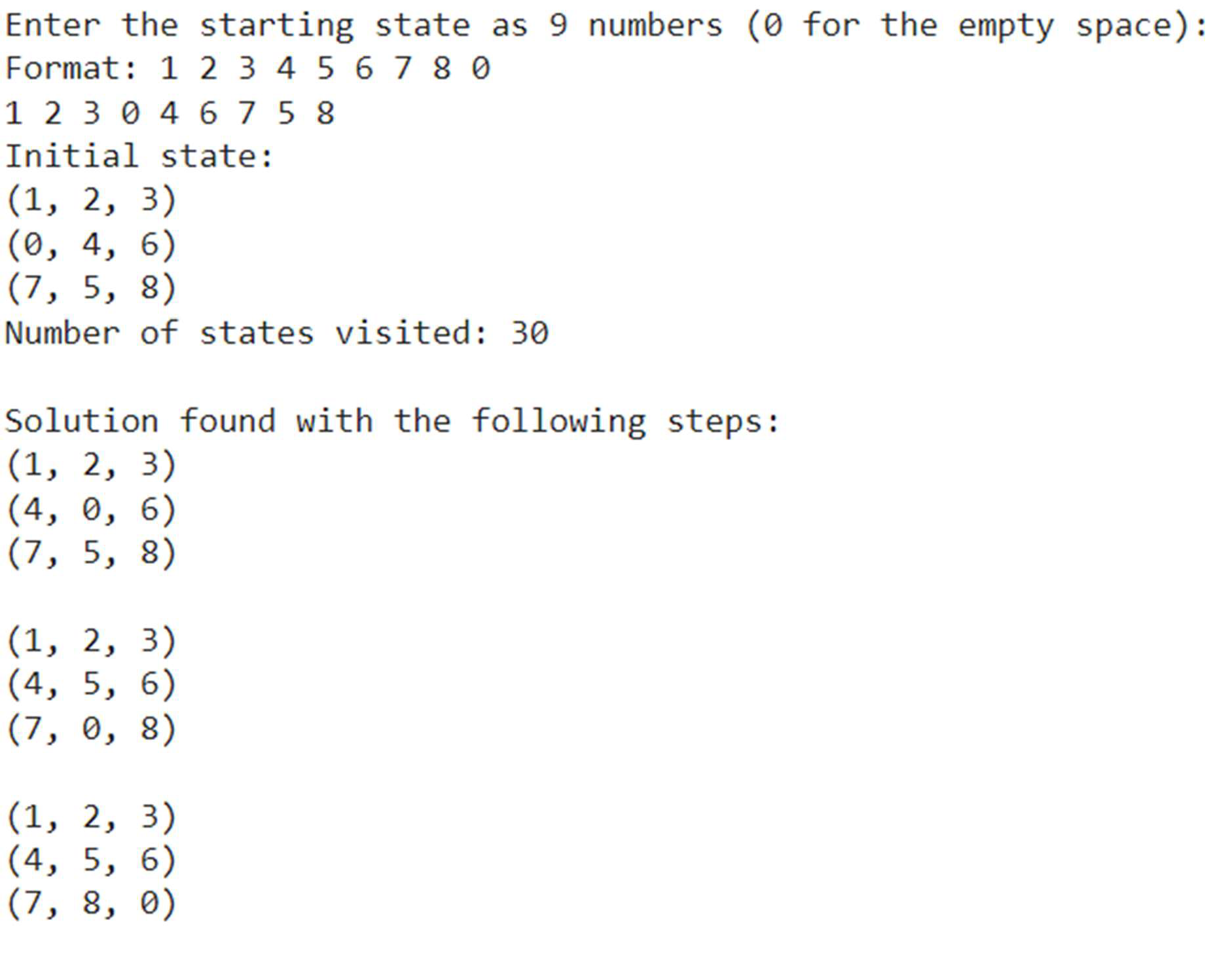
initial\_state = input\_start\_state() print("Initial state:") print\_matrix(initial\_state)

solution, visited\_count = bfs(initial\_state) print(f"Number of states visited: {visited\_count}") if solution:

print("\nSolution found with the following steps:") for step in solution:

print\_matrix(step) print()

else:

print("No solution found.") Output :