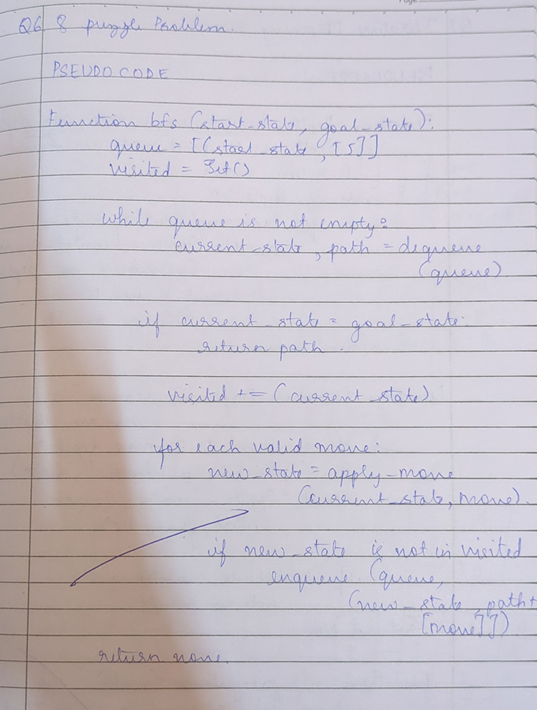
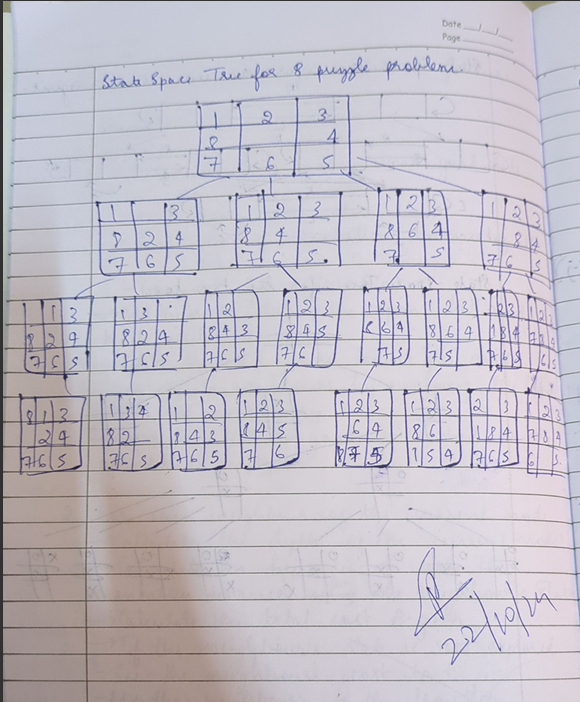
**8 Puzzle Problem**

**Algorithm:**

****

**State Space Tree:**

****

**Code:**

from collections import deque

# Moves: Up, Down, Left, Right

moves = {

'U': (-1, 0),

'D': (1, 0),

'L': (0, -1),

'R': (0, 1)

}

# Convert puzzle to string format for hashing

def puzzle\_to\_string(puzzle):

return ''.join(str(num) for row in puzzle for num in row)

# Print puzzle for better visualization

def print\_puzzle(puzzle):

for row in puzzle:

print(row)

print()

# BFS algorithm

def bfs\_8\_puzzle(start, goal):

queue = deque([(start, [])]) # Each element is (current\_state, path\_taken)

visited = set()

visited.add(puzzle\_to\_string(start))

state\_space\_tree = []

while queue:

current, path = queue.popleft()

if current == goal:

state\_space\_tree.append((current, path)) # Record final state and path

return state\_space\_tree # Puzzle solved, return best path

zero\_pos = [(i, j) for i in range(3) for j in range(3) if current[i][j] == 0][0]

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

x, y = zero\_pos[0] + dx, zero\_pos[1] + dy

if 0 <= x < 3 and 0 <= y < 3:

new\_state = [row[:] for row in current]

new\_state[zero\_pos[0]][zero\_pos[1]], new\_state[x][y] = new\_state[x][y], new\_state[zero\_pos[0]][zero\_pos[1]]

if puzzle\_to\_string(new\_state) not in visited:

visited.add(puzzle\_to\_string(new\_state))

queue.append((new\_state, path + [move])) # Add move to the path

state\_space\_tree.append((new\_state, path + [move])) # Record the state and path

return state\_space\_tree # Return the entire state space tree if no solution

# Function to get input from user

def get\_input():

start\_state = []

goal\_state = []

print("Enter the start state (3x3 matrix, row by row, with 0 representing the blank space):")

for i in range(3):

row = list(map(int, input().split()))

start\_state.append(row)

print("Enter the goal state (3x3 matrix, row by row, with 0 representing the blank space):")

for i in range(3):

row = list(map(int, input().split()))

goal\_state.append(row)

return start\_state, goal\_state

# Example Usage

start\_state, goal\_state = get\_input()

# Get the best path using BFS

solution\_tree = bfs\_8\_puzzle(start\_state, goal\_state)

# Print all states in the best move sequence

if solution\_tree:

print("goal state achieved")

print\_puzzle(goal\_state)

else:

print("No solution found.")

**Output:**

