**ASSIGNMENT 6**

Name: Aaryan Aher

PRN: 22210645

Div:- TY A [1]

Roll No.: 371003

**Title:**  Implement N- queen Problem using Back tracking method. A suitable message is to be displayed if the given Problem instance doesn't have a solution.

**CODE:**

# Size of the chessboard (N x N)

N = 4

# Function to print the solution (the board configuration)

def print\_solution(board):

    # Loop over each row of the board

    for i in range(N):

        # Loop over each column of the board

        for j in range(N):

            # Print the current element (either 1 for a queen or 0 for empty)

            print(board[i][j], end=" ")

        print()  # Newline after each row

# Function to check if placing a queen at board[row][col] is safe

def is\_safe(board, row, col):

    # Check the left side of the current row for any queens

    for i in range(col):

        if board[row][i]:

            return False  # Unsafe if a queen is found in the same row to the left

    # Check upper diagonal on the left side

    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

        if board[i][j]:

            return False  # Unsafe if a queen is found in the upper diagonal

    # Check lower diagonal on the left side

    for i, j in zip(range(row, N, 1), range(col, -1, -1)):

        if board[i][j]:

            return False  # Unsafe if a queen is found in the lower diagonal

    return True  # Safe to place a queen here

# Recursive utility function to solve the N-Queens problem

def solve\_nq\_util(board, col):

    # Base case: If all queens are placed, return True

    if col >= N:

        return True

    # Consider this column and try placing queens in all rows one by one

    for i in range(N):

        # Check if placing a queen at board[i][col] is safe

        if is\_safe(board, i, col):

            # Place a queen at board[i][col]

            board[i][col] = 1

            # Recur to place the rest of the queens

            if solve\_nq\_util(board, col + 1):

                return True

            # If placing queen in board[i][col] doesn't lead to a solution, remove it (backtracking)

            board[i][col] = 0

    # If no place is found in this column, return False

    return False

# Function to solve the N-Queens problem

def solve\_nq():

    # Initialize the board with all zeros

    board = [[0] \* N for \_ in range(N)]

    # Use solve\_nq\_util() to solve the problem

    if not solve\_nq\_util(board, 0):

        print("Solution does not exist")

        return False

    # Print the solution (the board configuration)

    print\_solution(board)

    return True

# Driver code

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

    solve\_nq()

**OUTPUT:**

