N = 4  # Define the size of the chessboard (N x N)

def print\_solution(board):

    """Print the board configuration."""

    for i in range(N):

        for j in range(N):

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

        print()

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

    """Check if it's safe to place a queen at position (row, col)."""

    # Check this row on the left side

    for i in range(col):

        if board[row][i]:  # If there's a queen, it's not safe

            return False

    # Check the upper diagonal on the left side

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

        if board[i][j]:  # If there's a queen, it's not safe

            return False

    # Check the lower diagonal on the left side

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

        if board[i][j]:  # If there's a queen, it's not safe

            return False

    return True  # The position is safe for a queen

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

    """Utilize backtracking to solve the N-Queens problem."""

    # Base case: If all queens are placed

    if col >= N:

        return True

    # Consider each row in the current column

    for i in range(N):

        if is\_safe(board, i, col):  # Check if we can place a queen here

            board[i][col] = 1  # Place queen

            # Recur to place queens in the next column

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

                return True

            # Backtrack if placing queen here doesn't lead to a solution

            board[i][col] = 0

    return False  # Return false if no safe position is found in this column

def solve\_nq():

    """Main function to solve the N-Queens problem."""

    board = [[0] \* N for \_ in range(N)]  # Initialize the board with 0s

    if not solve\_nq\_util(board, 0):  # Start from the first column

        print("Solution does not exist")  # No solution found

        return False

    print\_solution(board)  # Print the solution

    return True

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

    solve\_nq()  # Execute the solver function