```
def print_solution(board):
    N = len(board)
    for row in board:
        print(" ".join(str(cell) for cell in row))
    print()
def is_safe(board, row, col):
    N = len(board)
    # Check left side of the current row
    for i in range(col):
        if board[row][i]:
            return False
    # Check upper diagonal on left side
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j]:
            return False
    # Check lower diagonal on left side
    for i, j in zip(range(row, N, 1), range(col, -1, -1)):
        if board[i][j]:
            return False
    return True
def solve_nq_util(board, col):
    N = len(board)
    # All queens are placed
    if col >= N:
        print_solution(board)
        return True
    res = False
    for i in range(N):
        if is_safe(board, i, col):
            board[i][col] = 1
            res = solve_nq_util(board, col + 1) or res
            board[i][col] = 0  # Backtrack
    return res
def solve_n_queens(N):
    board = [[0 for _ in range(N)] for _ in range(N)]
    if not solve_nq_util(board, 0):
        print("No solution exists")
        return False
    return True
# Example usage:
solve_n_queens(4)
```