In [2]:

```python
#KETHARNATH R - 111723102089 - CSE C 1(B)
def print_board(board):
    """Prints the board in a readable format."""
    for row in board:
        print(" ".join(str(cell) for cell in row))
    print()

def is_safe(board, row, col, N):
    """Checks if placing a queen at (row, col) is safe."""

    # Check the column
    for i in range(row):
        if board[i][col] == 1:
            return False

    # Check upper left diagonal
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    # Check upper right diagonal
    for i, j in zip(range(row, -1, -1), range(col, N)):
        if board[i][j] == 1:
            return False

    return True

def solve_n_queens(board, row, N):
    """Solves the N-Queens problem using backtracking."""
    if row >= N:
        return True

    for col in range(N):
        if is_safe(board, row, col, N):
            board[row][col] = 1  # Place the queen

            if solve_n_queens(board, row + 1, N):
                return True  # Continue to the next row

            board[row][col] = 0  # Backtrack if no solution found

    return False

# Driver Code
if __name__ == "__main__":
    print("Enter the number of queens:")
    N = int(input())

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

    if solve_n_queens(board, 0, N):
        print("Solution exists. Placements of queens:")
        print_board(board)
    else:
        print("No solution exists.")
```

```
Enter the number of queens:
```

```
Solution exists. Placements of queens:
1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
```

In [ ]: