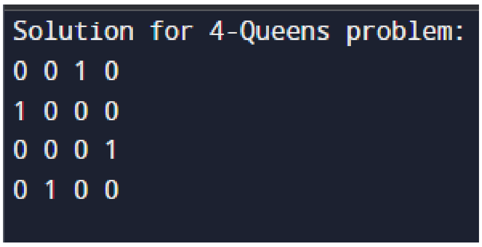
115.N- Queen Problem

AIM: To solve the N-Queen Problem

PROGRAM:

```
def is_safe(board, row, col, N):
    """ Check if it's safe to place a queen at board[row][col] """
    for i in range(col):
        if board[row][i] == 1:
            return False
        for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False
        for i, j in zip(range(row, N, 1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False
    return True
def solve_n_queens_util(board, col, N):
    """ Recursive utility function to solve N-Queens problem """
    if col >= N:
        return True
    for i in range(N):
        if is_safe(board, i, col, N):
            board[i][col] = 1  # Place the queen
            if solve_n_queens_util(board, col + 1, N):
                return True
                        board[i][col] = 0
    return False
def solve_n_queens(N):
    """ Function to solve the N-Queens problem """
    board = [[0] * N for _ in range(N)]
    if not solve_n_queens_util(board, 0, N):
        print(f"No solution exists for {N}-Queens problem.")
        return False
    print(f"Solution for {N}-Queens problem:")
    print_board(board, N)
    return True
def print_board(board, N):
    """ Utility function to print the board """
    for i in range(N):
        for j in range(N):
            print(board[i][j], end=" ")
        print()
N = 4
solve_n_queens(N)
```

OUTPUT: 

```
Solution for 4-Queens problem:
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```

TIME COMPLEXITY: O(N!)