LAB 05

To Implement a program to solve the N-Queen problem.

OBJECTIVE: To Implement a program to solve the N-Queen problem.
To compute correlation of discrete time signals and study their properties.
Algorithm:
1) Start in the leftmost column
2) If all queens are placed return true
3) Try all rows in the current column. Do following for every tried row.
    a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
    b) If placing the queen in [row, column] leads to a solution then return true.
    c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
4) If all rows have been tried and nothing worked, return false to trigger backtracking.

```
def is_safe(board, row, col, N):
    # Check the row on the left side
    for i in range(col):
        if board[row][i] == 'Q':
            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] == 'Q':
            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] == 'Q':
            return False

    return True

def solve_n_queens(board, col, N):
    if col >= N:
        return True

    for i in range(N):
        if is_safe(board, i, col, N):
            board[i][col] = 'Q'

            if solve_n_queens(board, col+1, N):
                return True
```

```
            board[i][col] = 0

    return False

def print_board(board, N):
    for i in range(N):
        for j in range(N):
            print(board[i][j], end=" ")
        print()


def n_queens(N):
    board = [[0 for i in range(N)] for j in range(N)]

    if solve_n_queens(board, 0, N) == False:
        print("Solution does not exist.")
        return False

    print_board(board, N)
    return True

n_queens(8)
```