

Input:

```
def print_solution(board):
    for row in board:
        print(" ".join(str(cell) for cell in row))
    print()

def is_safe(board, row, col, n):
    for i in range(row):
        if board[i][col] == 1:
            return False

    i, j = row, col
    while i >= 0 and j >= 0:
        if board[i][j] == 1:
            return False
        i -= 1
        j -= 1

    i, j = row, col
    while i >= 0 and j < n:
        if board[i][j] == 1:
            return False
        i -= 1
        j += 1

    return True

def solve_n_queens(board, row, n):
    if row >= n:
        print("Solution:")
        print_solution(board)
        return True

    for col in range(n):
        if is_safe(board, row, col, n):
            board[row][col] = 1

            if solve_n_queens(board, row + 1, n):
                return True

            board[row][col] = 0

    return False

def n_queens_with_first_queen_placed(n, first_row, first_col):
    board = [[0 for _ in range(n)] for _ in range(n)]

    board[first_row][first_col] = 1

    if not solve_n_queens(board, first_row + 1, n):
        print("No solution exists for this configuration.")

def main():
```

```
n = int(input("Enter the value of N for the NxN board: "))
first_row = int(input("Enter the row index (0-based) for the first queen:
"))
first_col = int(input("Enter the column index (0-based) for the first queen:
"))

print(f"Initial board with the first queen placed at ({first_row},
{first_col}):")
n_queens_with_first_queen_placed(n, first_row, first_col)

main()
```

Output:

```
Enter the value of N for the NxN board: 4
Enter the row index (0-based) for the first queen: 0
Enter the column index (0-based) for the first queen: 1
Initial board with the first queen placed at (0, 1):
Solution:
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0
```