Practical No. 5

Aim : Design 8-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final 8-queen's matrix.

Code :

```
def is_safe(board, row, col, n):
    for i in range(row):
        if board[i][col] == 1:
            return False
    for i, j in zip(range(row - 1, -1, -1), range(col - 1, -1, -1)):
        if board[i][j] == 1:
            return False
    for i, j in zip(range(row - 1, -1, -1), range(col + 1, n)):
        if board[i][j] == 1:
            return False
    return True

def solve_queens(board, row, n):
    if row == n:
        return True
    for col in range(n):
        if is_safe(board, row, col, n):
            board[row][col] = 1
            if solve_queens(board, row + 1, n):
                return True
            board[row][col] = 0
    return False

def print_board(board):
    print("\nSolution found:\n")
    for row in board:
        print(" ".join("Q" if cell == 1 else "." for cell in row))

n = 8
board = [[0 for _ in range(n)] for _ in range(n)]

try:
    row = int(input("Enter row (0–7) for the first Queen: "))
```

```
    col = int(input("Enter column (0–7) for the first Queen: "))

    if 0 <= row < n and 0 <= col < n:
        board[row][col] = 1
        if solve_queens(board, row + 1, n):
            print_board(board)
        else:
            print("No solution found from the given starting position.")
    else:
        print("Invalid input. Row and column must be between 0 and 7.")
except ValueError:
    print("Invalid input. Please enter numeric values.")
```

Output :

```
Enter row (0–7) for the first Queen: 0
Enter column (0–7) for the first Queen: 2

Solution found:

. . Q . . . . .
. . . . . Q . .
. . . . . . . Q
. Q . . . . . .
. . . Q . . . .
Q . . . . . . .
. . . . Q . . .
. . . . . . Q .


Enter row (0–7) for the first Queen: 1
Enter column (0–7) for the first Queen: 3

Solution found:

. . . . . . Q .
. . . Q . . . .
Q . . . . . . .
. . . . . Q . .
. Q . . . . . .
. . . . . . . Q
. . Q . . . . .
. . . . Q . . .
```