Name: Faraz Pathan

Roll no: COTB70

**PROGRAM:**

class NQueens:

    def \_\_init\_\_(self, n):

        self.n = n

        self.board = [[0] \* n for \_ in range(n)]

        self.solutions = []

    def is\_safe(self, row, col):

        # Check if there is any queen in the same column

        for i in range(row):

            if self.board[i][col] == 1:

                return False

        # Check upper left diagonal

        for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

            if self.board[i][j] == 1:

                return False

        # Check upper right diagonal

        for i, j in zip(range(row, -1, -1), range(col, self.n)):

            if self.board[i][j] == 1:

                return False

        return True

    def solve(self, row):

        if row == self.n:

            # If all queens are placed successfully, add the solution

            self.solutions.append([row[:] for row in self.board])

            return

        for col in range(self.n):

            if self.is\_safe(row, col):

                # Place the queen

                self.board[row][col] = 1

                # Recur to place rest of the queens

                self.solve(row + 1)

                # If placing queen in board[row][col] doesn't lead to a solution

                # then remove the queen from board[row][col]

                self.board[row][col] = 0

    def find\_solutions(self):

        self.solve(0)

        return self.solutions

# Example usage:

n = 8

n\_queens = NQueens(n)

solutions = n\_queens.find\_solutions()

print(f"Number of solutions for {n}-queens problem:", len(solutions))

print("One of the solutions:")

for row in solutions[0]:

    print(row)

**OUTPUT:**

Number of solutions for 8-queens problem: 92

One of the solutions:

[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]