**EX.NO:1 DATE:11/9/2024**

**Reg.no:220701012**

**8- QUEENS PROBLEM**

**AIM :** To implement an 8-Queesns problem using Python.

You are given an 8x8 board; find a way to place 8 queens such that no queen can attack any other

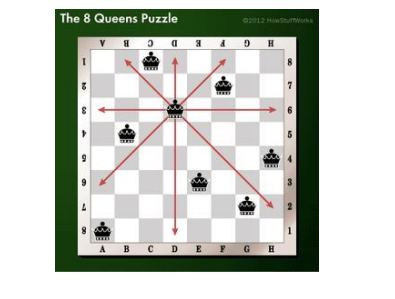
queen on the chessboard. A queen can only be attacked if it lies on the same row, same column,

or the same diagonal as any other queen. Print all the possible configurations.

To solve this problem, we will make use of the Backtracking algorithm. The backtracking

algorithm, in general checks all possible configurations and test whether the required result is

obtained or not. For the given problem, we will explore all possible positions the queens can be

relatively placed at. The solution will be correct when the number of placed queens = 8. ****

**CODE:**

def print\_board(board):

    for row in board:

        print(' '.join('Q' if cell else '.' for cell in row))

    print()

def is\_safe(board, row, col, N):

    # Check this column

    for i in range(row):

        if board[i][col]:

            return False

    # Check upper left diagonal

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

        if board[i][j]:

            return False

    # Check upper right diagonal

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

        if board[i][j]:

            return False

    return True

def solve\_nqueens\_util(board, row, N):

    if row >= N:

        print\_board(board)

        return True

    res = False

    for col in range(N):

        if is\_safe(board, row, col, N):

            board[row][col] = True

            res = solve\_nqueens\_util(board, row + 1, N) or res

            board[row][col] = False

    return res

def solve\_nqueens(N):

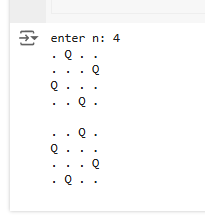
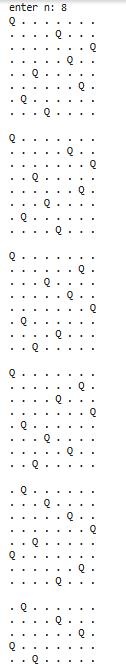
    board = [[False] \* N for \_ in range(N)]

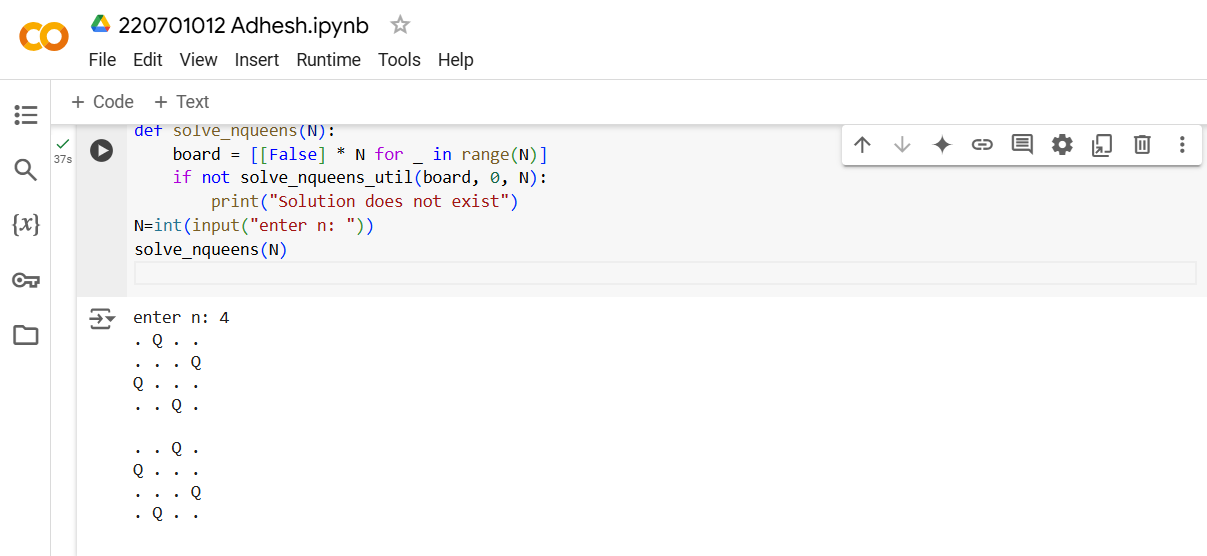
    if not solve\_nqueens\_util(board, 0, N):

        print("Solution does not exist")

N=int(input("enter n: "))

solve\_nqueens(N)

**  
**

****