# ass 4

```
# Implement the solution for a Constraint Satisfaction Problem using Branch and Bound and

# Backtracking for n-queens problem or a graph coloring problem


# branch and bound


def printSolution(board):

    for i in range(N):

        for j in range(N):

            print(board[i][j], end = " ")

        print()



def isSafe(row, col, nd, rd,rowLookup, ndLookup,rdLookup):

    if (ndLookup[nd[row][col]] or rdLookup[rd[row][col]] or rowLookup[row]):

        return False

    return True



def solveNQueensUtil(board, col, nd, rd,rowLookup, ndLookup,rdLookup):

    if(col >= N):

        return True

    for i in range(N):

        if(isSafe(i, col, nd, rd,rowLookup, ndLookup,rdLookup)):



            board[i][col] = 1

            rowLookup[i] = True

            ndLookup[nd[i][col]] = True

            rdLookup[rd[i][col]] = True
            if(solveNQueensUtil(board, col + 1,nd, rd,rowLookup, ndLookup,rdLookup)):
                return True
            board[i][col] = 0
            rowLookup[i] = False
            ndLookup[nd[i][col]] = False
            rdLookup[rd[i][col]] = False
    return False

def solveNQueens(N):

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



    nd = [[0 for i in range(N)] for j in range(N)]
    rd = [[0 for i in range(N)] for j in range(N)]
    rowLookup = [False] * N



    x = 2 * N - 1
    ndLookup = [False] * x
    rdLookup= [False] * x



    for r in range(N):
        for c in range(N):
            nd[r][c] = r + c
            rd[r][c] = r - c + N - 1

    if(solveNQueensUtil(board, 0, nd, rd,rowLookup, ndLookup,rdLookup) == False):
        print("Solution does not exist")

        return False



    printSolution(board)

    return True



N=int(input("Enter a Number: "))

solveNQueens(N)



# backtracking


from typing import List

boardcount=0

def isboardok(chessboard:List,row:int,col:int):

    for c in range(col):

        if(chessboard[row][c]=='Q'):

            return False

    for r,c in zip(range(row-1,-1,-1),range(col-1,-1,-1)):

        if(chessboard[r][c]=='Q'):

            return False

    for r,c in zip(range(row+1,len(chessboard),1),range(col-1,-1,-1)):

        if(chessboard[r][c]=='Q'):

            return False

    return True



def displayboard(chessboard:List):

    for row in chessboard:
        print(row)
    print()



def placenqueens(chessboard:List,col:int):
    global boardcount
    if(col>=len(chessboard)):
        boardcount+=1
        print("Board"+str(boardcount))
        print("======================")
        displayboard(chessboard)
        print("======================\n\n")
    else:
        for row in range(len(chessboard)):
            chessboard[row][col]='Q'
            if(isboardok(chessboard,row,col)==True):
                placenqueens(chessboard,col+1)
            chessboard[row][col]='.'

chessboard=[]
N=int(input("Enter chessboard size: "))
for i in range(N):
    row=["."]*N
    chessboard.append(row)

placenqueens(chessboard,0)
```

<u># ass 4 done</u>