Assignment B4

```
class NQBranchAndBond:
    def printSolution(self, board):
        print("N Queen Branch And Bound Solution:")
        for line in board:
            print(" ".join(map(str, line)))

    def isSafe(
        self,
        row,
        col,
        slashCode,
        backslashCode,
        rowLookup,
        slashCodeLookup,
        backslashCodeLookup,
    ):
        return not (
            slashCodeLookup[slashCode[row][col]]
            or backslashCodeLookup[backslashCode[row][col]]
            or rowLookup[row]
        )


    def solveNQUtil(
        self,
        board,
        col,
        slashCode,
        backslashCode,
        rowLookup,
        slashCodeLookup,
        backslashCodeLookup,
    ):

        if col >= N:
            return True

        for i in range(N):
            if self.isSafe(
                i,
                col,
                slashCode,
                backslashCode,
                rowLookup,
                slashCodeLookup,
                backslashCodeLookup,
            ):

                board[i][col] = 1
                rowLookup[i] = True
                slashCodeLookup[slashCode[i][col]] = True
                backslashCodeLookup[backslashCode[i][col]] = True

                if self.solveNQUtil(
                    board,
                    col + 1,
                    slashCode,
                    backslashCode,
                    rowLookup,
                    slashCodeLookup,
                    backslashCodeLookup,
                ):
                    return True

                board[i][col] = 0
                rowLookup[i] = False
                slashCodeLookup[slashCode[i][col]] = False
                backslashCodeLookup[backslashCode[i][col]] = False

    def solveNQ(self):
        board = [[0 for i in range(N)] for j in range(N)]

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

        rowLookup = [False] * N

        x = 2 * N - 1
        slashCodeLookup = [False] * x
        backslashCodeLookup = [False] * x

        for rr in range(N):
            for cc in range(N):
                slashCode[rr][cc] = rr + cc
                backslashCode[rr][cc] = rr - cc + N - 1

        if (
            self.solveNQUtil(
                board,
                0,
                slashCode,
                backslashCode,
                rowLookup,
                slashCodeLookup,
                backslashCodeLookup,
            )
            == False
        ):
            print("Solution does not exist")
            return False

        self.printSolution(board)
        return True


class NQBacktracking:
    def __init__(self):
        self.ld = [0] * 30
        self.rd = [0] * 30
        self.cl = [0] * 30

    def printSolution(self, board):
        print("\n\nN Queen Backtracking Solution:")
        for line in board:
            print(" ".join(map(str, line)))

    def solveNQUtil(self, board, col):

        if col >= N:
            return True
        for i in range(N):

            if (self.ld[i - col + N - 1] != 1 and
                self.rd[i + col] != 1) and self.cl[i] != 1:

                board[i][col] = 1
                self.ld[i - col + N - 1] = self.rd[i + col] = self.cl[i]
= 1

                if self.solveNQUtil(board, col + 1):
                    return True

                board[i][col] = 0  # BACKTRACK
                self.ld[i - col + N - 1] = self.rd[i + col] = self.cl[i]
= 0

    def solveNQ(self):
        board = [[0 for _ in range(N)] for __ in range(N)]
        if self.solveNQUtil(board, 0) == False:
            print("Solution does not exist")
            return False
        self.printSolution(board)
        return True


if __name__ == "__main__":
    N = 8

    NQBaB = NQBranchAndBond()
    NQBaB.solveNQ()

    NQBt = NQBacktracking()
    NQBt.solveNQ()
```

Output :-

```
N Queen Branch And Bound Solution:
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0


N Queen Backtracking Solution:
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
```