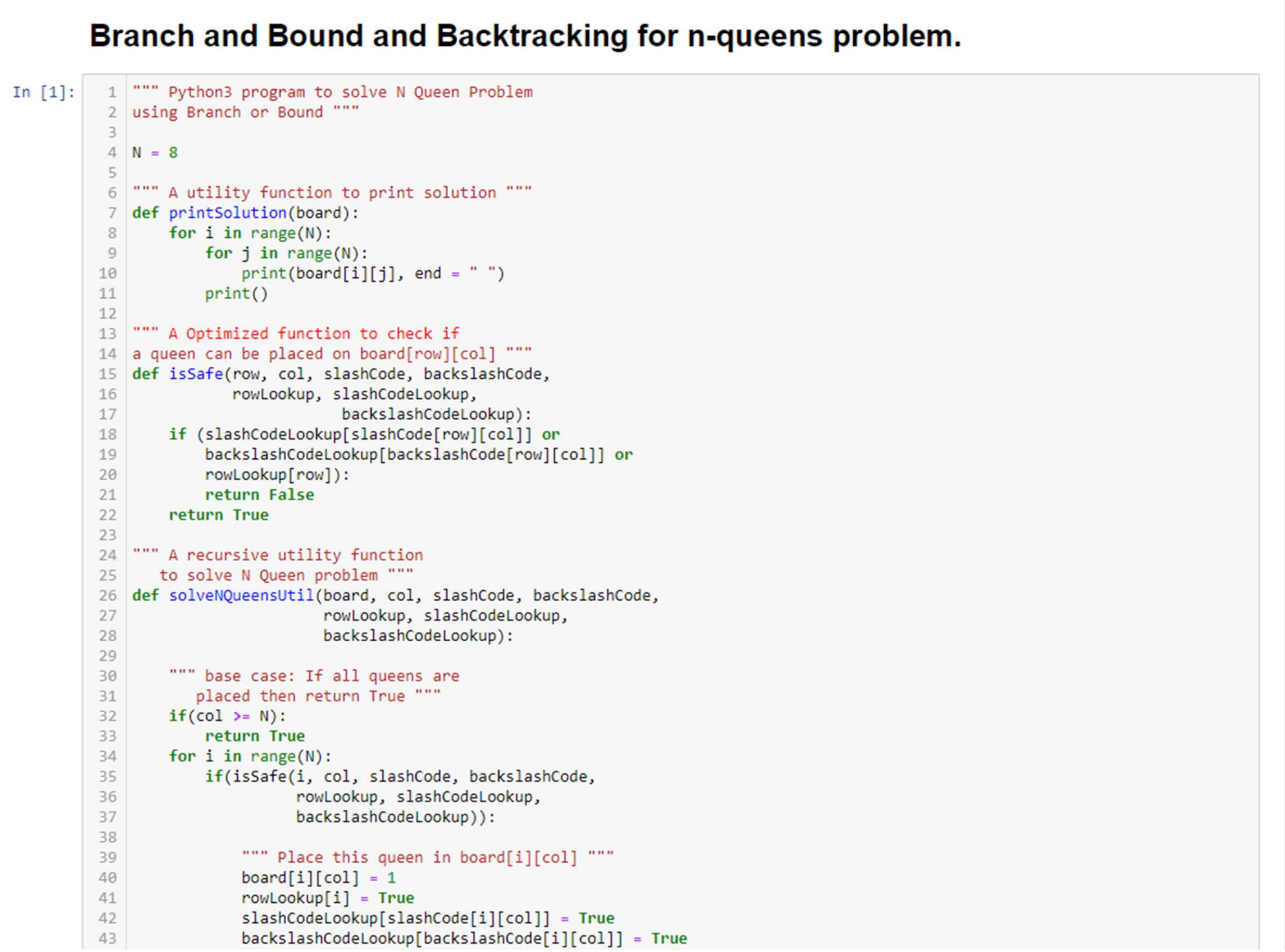
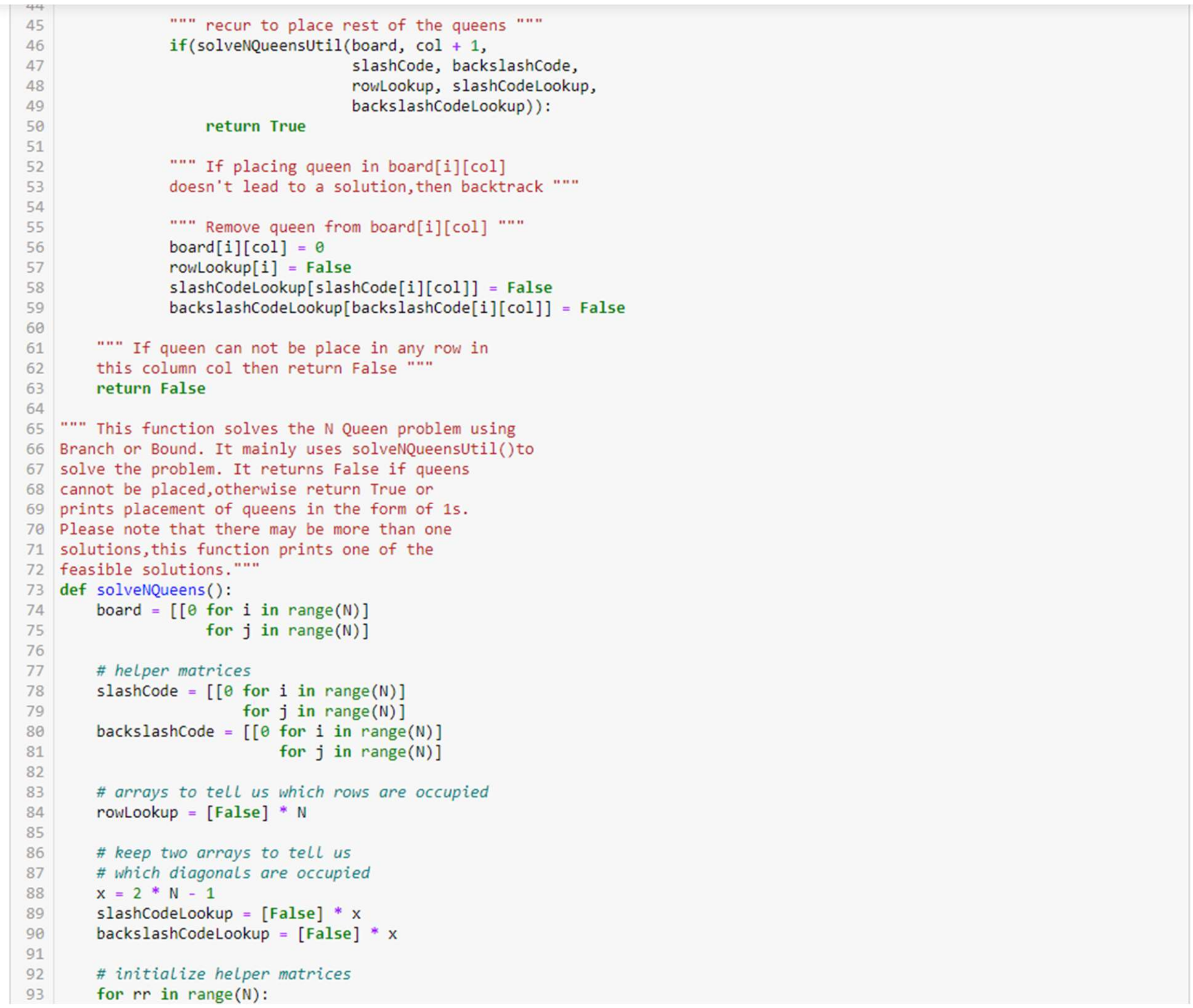


Branch and Bound and Backtracking for n-queens problem.

In [1]:

```python
""" Python3 program to solve N Queen Problem
using Branch or Bound """

N = 8

""" A utility function to print solution """
def printSolution(board):
    for i in range(N):
        for j in range(N):
            print(board[i][j], end = " ")
        print()

""" A Optimized function to check if
a queen can be placed on board[row][col] """
def isSafe(row, col, slashCode, backslashCode,
           rowLookup, slashCodeLookup,
                      backslashCodeLookup):
    if (slashCodeLookup[slashCode[row][col]] or
        backslashCodeLookup[backslashCode[row][col]] or
        rowLookup[row]):
        return False
    return True

""" A recursive utility function
   to solve N Queen problem """
def solveNQueensUtil(board, col, slashCode, backslashCode,
                     rowLookup, slashCodeLookup,
                     backslashCodeLookup):

    """ base case: If all queens are
       placed then return True """
    if(col >= N):
        return True
    for i in range(N):
        if(isSafe(i, col, slashCode, backslashCode,
                  rowLookup, slashCodeLookup,
                  backslashCodeLookup)):

            """ Place this queen in board[i][col] """
            board[i][col] = 1
            rowLookup[i] = True
            slashCodeLookup[slashCode[i][col]] = True
            backslashCodeLookup[backslashCode[i][col]] = True
```



```python
            """ recur to place rest of the queens """
            if(solveNQueensUtil(board, col + 1,
                                slashCode, backslashCode,
                                rowLookup, slashCodeLookup,
                                backslashCodeLookup)):
                return True

            """ If placing queen in board[i][col]
            doesn't lead to a solution,then backtrack """

            """ Remove queen from board[i][col] """
            board[i][col] = 0
            rowLookup[i] = False
            slashCodeLookup[slashCode[i][col]] = False
            backslashCodeLookup[backslashCode[i][col]] = False

    """ If queen can not be place in any row in
    this column col then return False """
    return False

""" This function solves the N Queen problem using
Branch or Bound. It mainly uses solveNQueensUtil()to
solve the problem. It returns False if queens
cannot be placed,otherwise return True or
prints placement of queens in the form of 1s.
Please note that there may be more than one
solutions,this function prints one of the
feasible solutions."""
def solveNQueens():
    board = [[0 for i in range(N)]
             for j in range(N)]

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

    # arrays to tell us which rows are occupied
    rowLookup = [False] * N

    # keep two arrays to tell us
    # which diagonals are occupied
    x = 2 * N - 1
    slashCodeLookup = [False] * x
    backslashCodeLookup = [False] * x

    # initialize helper matrices
    for rr in range(N):
```

```python

    # arrays to tell us which rows are occupied
    rowLookup = [False] * N

    # keep two arrays to tell us
    # which diagonals are occupied
    x = 2 * N - 1
    slashCodeLookup = [False] * x
    backslashCodeLookup = [False] * x

    # initialize helper matrices
    for rr in range(N):
        for cc in range(N):
            slashCode[rr][cc] = rr + cc
            backslashCode[rr][cc] = rr - cc + 7

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

    # solution found
    printSolution(board)
    return True

# Driver Code
solveNQueens()
```

```
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
```

```
Out[1]: True
```