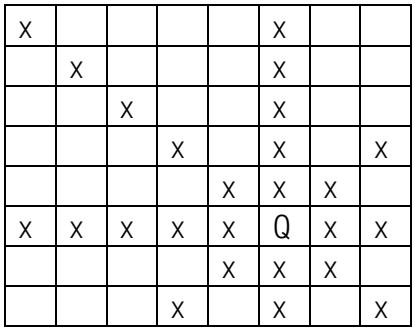
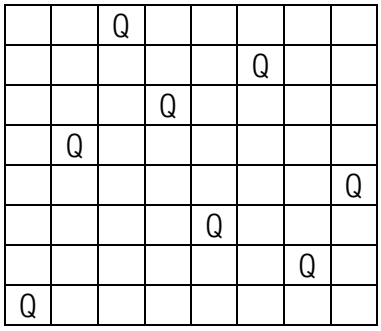
CSCI 503 **Computer Assignment 3** **Spring 2019**
(10 points)

N Queens Puzzle

The most powerful piece in the game of chess is the queen, which can move any number of squares in any direction, horizontally, vertically, or diagonally. For example, the queen shown in the following chessboard of size N = 8 can move to any of the marked squares.



Even though the queen can cover many squares, it is possible to place N queens on an N x N chessboard so that none of them can attack any of the others, as shown in the following diagram:



Write a **recursive** Python program that solves the problem of whether it is possible to place N queens for N = 1, 2, …, 8 on an N x N chessboard so that none of them can move to a square occupied by any of the others in a single turn, using **backtracking**. Your program should either display a solution if it finds one or report that no solutions exists.

- def driver ( ): It starts the program by calling the routine solveNQueens ( ), as explained below, for each value of N, between 1 and 8, to place the N queens on the chessboard.

- def initBoard (N): Initializes a RNG with a chosen seed value. For a random starting point, it uses the time ( ) function from the time package, calling the seed ( ) function from the random package. It also creates a two-dimensional list (matrix) for the board of size N and sets all positions on the board to False and returns the initial board to the following routine.

- def solveNQueens (N): To place the N queens on the board, starting from the first row, it prints out the board size and calls to routine solveNQueensUtil ( ), as explained below. If the returned value of solveNQueensUtil ( ) is True, then it calls the routine printBoard ( ) to print out the contents of the board on stdout by showing the positions of the queens on the board, otherwise, it prints a message indicating that a solution does not exists.

- def solveNqueensUtil (board, row): This **recursive** routine starts on the row number row and gets a random column number col, between 0 and (size (board) – 1) from the RNG by calling the function randint ( ) from the random package, and it checks if a queen can be placed on the location (row, col). It calls the routine isSafe ( ), which is explained below, to determine if the location is safe, so the queen can be placed in that location. If the row is not the last row on the board, it continues to the next row by making a recursive call. If the queen cannot be placed on the column col, then it chooses another column in the given row, and if none of the columns in the row turns out to be valid, then the returned value of the recursive call will be False. In this case, by **backtracking**, this routine simply returns to the previous row and chooses another column to replace the queen in that row. If the **backtracking** goes all the way to the first row and none of the columns in that row results a successful placement, then the routine returns False to the calling routine.

- def isSafe (board, row, col): It checks if a queen can be placed in the row number row and the column number col on the board. If the answer is "yes", then it returns True. If there is another queen in location (i, j), then the row cannot be equal to i, since the queens are placed on the board one piece at a time, but those two queens can be in the same column if col == j, and it can be easily verified that they can be in the same diagonal if abs (row – i) == abs (col – j).

- def printBoard (board): It prints the final state of the board as a properly-formatted rectangular table. See the correct output file prog3.out for the correct formatting.

Name your program as prog3.py, and for a final test of your program, execute: Make N=3. The correct output file, prog3.out, is in the directory: ~cs503/progs/19s/p3. Since you execute your program with a random seed value, you'll get a different placement of the queens on the board each time. For your information, the total number of possible placements is given as: 1 for N = 1; 0 for N = 2, 3; 2 for N = 4; 10 for N = 5; 4 for N = 6; 40 for N = 7; and 92 for N = 8, but some solutions differ only by *symmetry operations* (rotations and reflections) of the board.

The Make script can test your program for any board size of N >= 1, but you only need to test your program for N = 1, 2, …, 8.

To use the functions in packages time and random in your program, insert the following lines at the top of the program.

```
from time import time
from random import seed, randint
```

When your program is ready, mail your program file to your TA, executing mail_prog.503 prog3.py.