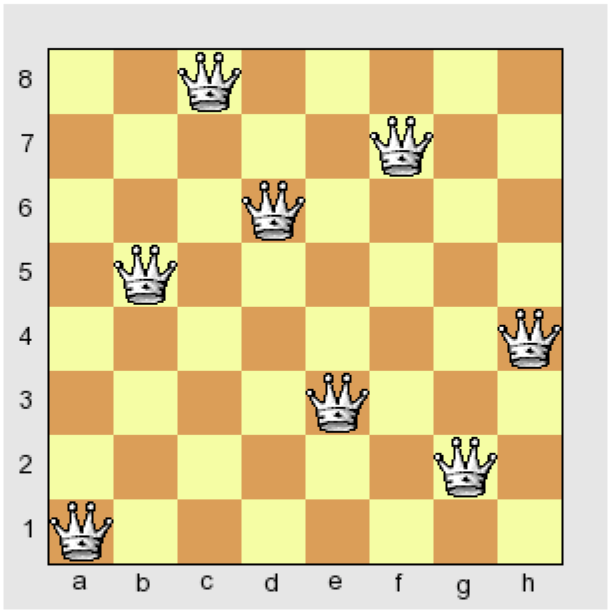
The eight queens puzzle is the problem of placing eight chess queens on an 8x8 chessboard so that none of them can capture any other using the standard chess queen's moves. If you are unfamiliar with chess, a queen can move any number of squares in one direction per turn (horizontally, vertically or diagonally).



Our goal for this recitation is to design and implement a solver.

1. One way this could be represented is as a 2 dimensional grid (i.e. a list of lists), where each cell of the grid is individually accessible by a row and column value. Show in Python how you can create this grid, called board, assuming (row=0, col=0) is in the upper left of the board. Assume each cell has a value of None to begin with.

2. Using the board from question 1, how do you store the string 'Q' at row=3, column=4. Likewise, how do you check the cell (1,4) has nothing in it?

3. a) In Python, what is wrong with making a copy of the board in question 2 by doing:
    newBoard = board?

   b) How do you make a full copy of the board?

4. a) How do you generate the collection of all possible neighbor configs for a given config? Is it possible that any of the neighbors are invalid configs?

   b) Assuming a Queen has been placed into the top left cell of the grid, how can you generate a list of potential neighbors without generating too many unnecessary ones?

5. Assuming our board is 4 x 4, what are the neighbors (valid and invalid) for the following configuration:

```
Q _ _ _
_ _ _ _
_ _ _ _
_ _ _ _
```

6. Show the search tree that is generated when performing solving using backtracking on the following 4x4 config.

```
_ _ _ _
Q _ _ _
_ _ _ _
_ _ _ _
```

7. Why is it with the following 4x4 config that you do not need to generate neighbors for it? What is the technical term for this?

```
Q _Q _
_ _ _ _
_Q _ _
_ _ _ _
```

8. Write the solve() function for a NQueens program. First, review the class design (Board class), and the three utility functions that will be used by solve() - isGoal(), generateNeighbors(), isValid().

Now write the solve() method. It takes a current Board object as its sole parameter and should return the solution config (if one exists), or None (no solution).