5. Keep Those Queens Apart

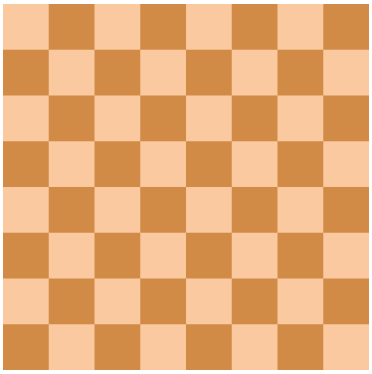
When the world says, "Give up," Hope whispers, "Try it one more time." – Author Unknown.

Programming constructs and algorithmic paradigms covered in this puzzle: Two-dimensional lists, while loops, continue statements and argument defaults in procedures. Exhaustive search via iteration. Conflict detection.

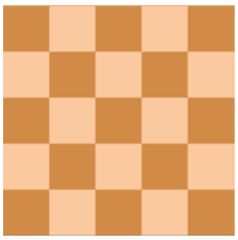
The 8-Queens problem on a chessboard corresponds to finding a placement of 8 queens such that no queen attacks any other queen. This means that

1. No two queens can be on the same column.
2. No two queens can be on the same row.
3. No two queens can be on the same diagonal.

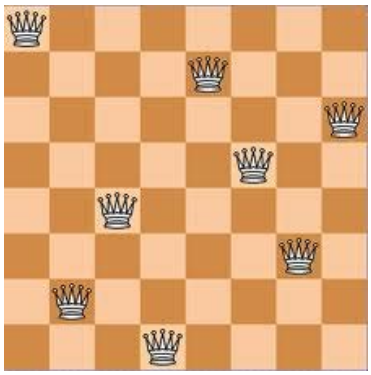
Can you find a solution?



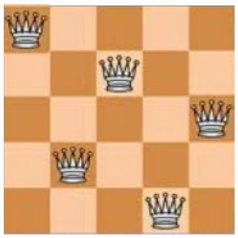
If 8 queens are too many to handle, try the 5-queens problem below!



A solution to the 8-queens problem is shown below. This is not the only solution!



And here's a solution to the 5-queens problem.



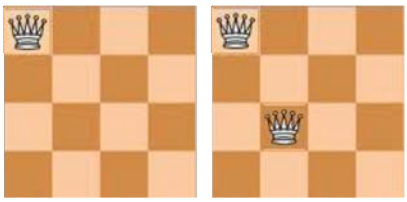
But how can we find the above solutions or different ones? Let's simplify the problem first. Suppose we look at a smaller 2 × 2 board. Can we place two queens so they don't attack each other? The answer is no since a queen on any square of a 2 × 2 board can attack every other square.

What about a 3 × 3 board? Here is an attempt:

For the first placement (above left), we can see that the queen attacks six other squares and occupies one. So there are 7 squares that are disallowed by the very first queen placement, leaving 2 squares available. When we put the second queen on one of those squares, there are no squares available. Of course, we could have tried putting the first queen in a different spot, but that won't help. The first queen regardless of where it is placed will occupy and attack at least 7 squares, leaving two. There is no solution for a 3 × 3 board.

Systematic Search

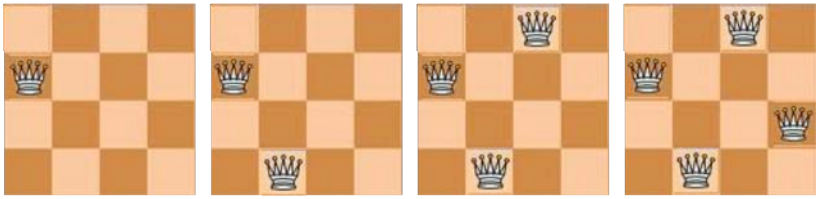
What about a 4 × 4 board? Let's be systematic about searching for a solution. We will try to place queens column by column and change placements if we fail. We start by placing a queen in the top left corner in the first column as shown below (first picture). There are two choices in placing a queen on the second column and we choose one as shown in the second picture below. Now we are stuck! We can't place a queen in the third column ☹



Note that when we tried to place the third queen in the third column above we simply checked that the third queen didn't conflict with the first and the second. We do not need to check that the first two queens do not conflict with each other. This is because we did that check when we placed the second queen. This will be important to remember when we look at the code to check for conflicts.

Given that we failed at finding a solution, should we give up? No, because we could try a different positioning for the second queen. Here we go:

We went further, but are stuck again, this time on the fourth and last column ☹
But we are not out of options. We arbitrarily chose the top corner for the first queen and we could try a different position. (There are four options for the first queen.) Here we go:



Success, the 4-queens problem has a solution ☺ It actually has two; try to find a different one.

It will take a normal person quite a long time to use this strategy to find a solution to the 8-queens problem. But such an exhaustive brute-force strategy should work. And if we can code up the solution, since computers can do calculations billions of times faster than humans, we will able to run the program and find a solution in a matter of seconds!

The first step in writing code for 8-queens is to decide on a data structure for the problem: How are we going to represent the board and the positions of the queens?

Exercises

Exercise 1: Modify the `EightQueens` code so it takes as an additional argument the number of solutions you want to find and prints that many assuming that many exist. Note that default arguments have to be after non-default arguments, so the new argument has to be the first one, followed by the `n=8` default argument.

Puzzle Exercise 2: Modify the `EightQueens` code so it looks for solutions with a queen already placed in a list of locations. You can use a 1-D list `location` as an argument that has non-negative entries for certain columns that correspond to fixed queen positions. For example, `location = [-1, 4, -1, -1, -1, -1, -1, 0]` has two queens placed in the 2nd and 8th columns. Your code should produce `[2, 4, 1, 7, 5, 3, 6, 0]` as a solution consistent with the prescribed queen locations.

Exercise 3: Reduce the level of indentation in the `FourQueens` code by employing the **`continue`** statement. Both solutions should be printed. Be warned that this is a little trickier than you might think!

6.S095 Programming for the Puzzled
January IAP 2018