3.2.2 Two Queens Problem

Given an $n \times n$ chessboard, our next problem is to count the number of ways we can place *two* queens on the board in such a way that they do not attack each other. For example, as Fig. 3.2 shows, there are eight ways to place two queens on the 3×3 board. Let $q(n)$ denote the number of valid combinations for an $n \times n$ board. For example, $q(3) = 8$, and Table 3.3 shows the values of $q(n)$ for $1 \le n \le 10$.

To start with, a simple way to solve the problem is to go through all possible ways to place two queens on the board and count the combinations where the queens do not attack each other. Such an algorithm works in $O(n^4)$ time, because there are n^2 ways to choose the position of the first queen, and for each such position, there are $n^2 - 1$ ways to choose the position of the second queen.

Since the number of combinations grows fast, an algorithm that counts combinations one by one will certainly be too slow for processing larger values of n. Thus, to create an efficient algorithm, we need to find a way to count combinations in *groups*. One useful observation is that it is quite easy to calculate the number of squares that a single queen attacks (Fig. 3.3). First, it always attacks $n - 1$ squares horizontally and $n - 1$ squares vertically. Then, for both diagonals, it attacks $d - 1$ squares where d is the number of squares on the diagonal. Using this information, we can calculate

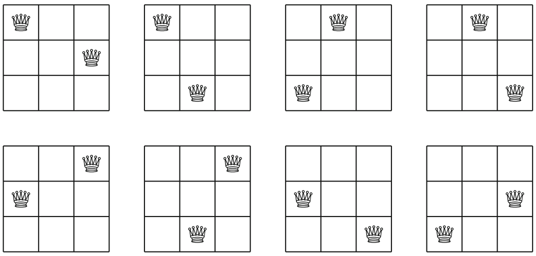

Fig. 3.2 All possible ways to place two non-attacking queens on the 3×3 chessboard

Table 3.3 First values of the function $q(n)$: the number of ways to place two non-attacking queens on an $n \times n$ chessboard

Board size n	Number of ways $q(n)$
1	0
2	0
3	8
4	44
5	140
6	340
7	700
8	1288
9	2184
10	3480

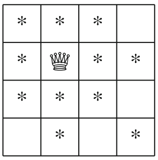
Fig. 3.3 The queen attacks all squares marked with "*" on the board

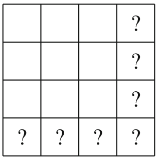
Fig. 3.4 Possible positions for queens on the last row and column

in $O(1)$ time the number of squares where the other queen can be placed, which yields an $O(n^2)$ time algorithm.

Another way to approach the problem is to try to formulate a recursive function that counts the number of combinations. The question is: if we know the value of $q(n)$, how can we use it to calculate the value of $q(n+1)$?

To get a recursive solution, we may focus on the last row and last column of the $n \times n$ board (Fig. 3.4). First, if there are no queens on the last row or column, the number of combinations is simply $q(n-1)$. Then, there are $2n-1$ positions for a queen on the last row or column. It attacks $3(n-1)$ squares, so there are $n^2 - 3(n-1) - 1$ positions for the other queen. Finally, there are $(n-1)(n-2)$ combinations where both queens are on the last row or column. Since we counted those combinations twice, we have to remove this number from the result. By combining all this, we get a recursive formula

$$\begin{aligned} q(n) &= q(n-1) + (2n-1)(n^2 - 3(n-1) - 1) - (n-1)(n-2) \\ &= q(n-1) + 2(n-1)^2(n-2), \end{aligned}$$

which provides an $O(n)$ solution to the problem.

Finally, it turns out that there is also a closed-form formula

$$q(n) = \frac{n^4}{2} - \frac{5n^3}{3} + \frac{3n^2}{2} - \frac{n}{3},$$

which can be proved using induction and the recursive formula. Using this formula, we can solve the problem in $O(1)$ time.