Solving The Eight Queens Chess Problem

Echchalim Aymen.
Github : AymenE19

Can you place 8 queens on a chessboard so that none of the queens can attack each other ?

If so, how many ways can we place 8 queens on the board ?

Quick view

So there are 64 objects, 8 queens and 56 blanks, there are 64! ways to arrange these objects. Intuitively, we can consider all queens are the same. There are 8 queens, and they can be swapped around between each other.

$$\frac{64!}{56!8!} = 4,426,165,368$$

Warning

That's not considering how the queens attack each other.

So how many solutions are there, out of that 4 **billion** figure ?

There exist 92 distinct solutions...

The eight queen puzzle is a special case of the more general n queens problem of placing n non attacking queens on a $n \times n$ chessboard.

Solutions exist for all natural numbers n with the exception of $n \in \{2, 3\}$ Although the exact number of solutions is only known for $n \leq 27$.

The asymptotic growth of the number of solutions is approximately $(0.143)^n$.

Abstraction and Modeling

- The move of a queen is the intersection of four lines one vertical, one horizontal, and two diagonals, one North-East with slope 1, and the other South-East with slope -1.
- There must be one and only one queen on each column and each row.
- From this observation, we note the position of the queens by lists of eight numbers.

$$(Q_1, Q_2, \ldots, Q_8)$$

where Q_i gives the row of the queen in column i, counting from the bottom to the top. This ensures that there is only one queen per column. When there is only one queen per row, all the Q_i are distinct, and such a list is called a permutation of the numbers from 1 to 8. All solutions must be permutations, but not all permutations are solutions. To ensure this, the diagonals must be checked.

Two queens respectively in positions (j, Q_j), (k, Q_k) are in the same diagonal if this equality holds true :

$$|Q_j - j| = |Q_k - k|$$

One step backward, two steps forward
4x4 chessboard

1. We place the first queen (1,x,x,x).
2. The position (1,2,x,x) is tested and is invalid.
3. We then move to (1,3,x,x).None of the possibilities (1,3,2,x) or (1,3,4,x) work.
4. We backtrack and move to (1,4,x,x).Then, (1,4,2,x) passes the test, but (1,4,2,3) does not.
5. We backtrack again and change the first choice: (2,x,x,x).
6. We move to (2,4,x,x) after testing (2,1,x,x) and (2,3,x,x).
7. From there, we reach the solution (2,4,1,3).
8. The algorithm continues with (3,x,x,x) and will quickly find the other solution (3,1,4,2), which is a 180-degree rotation of the one we found. These are the only two possible positions.

Decision Tree

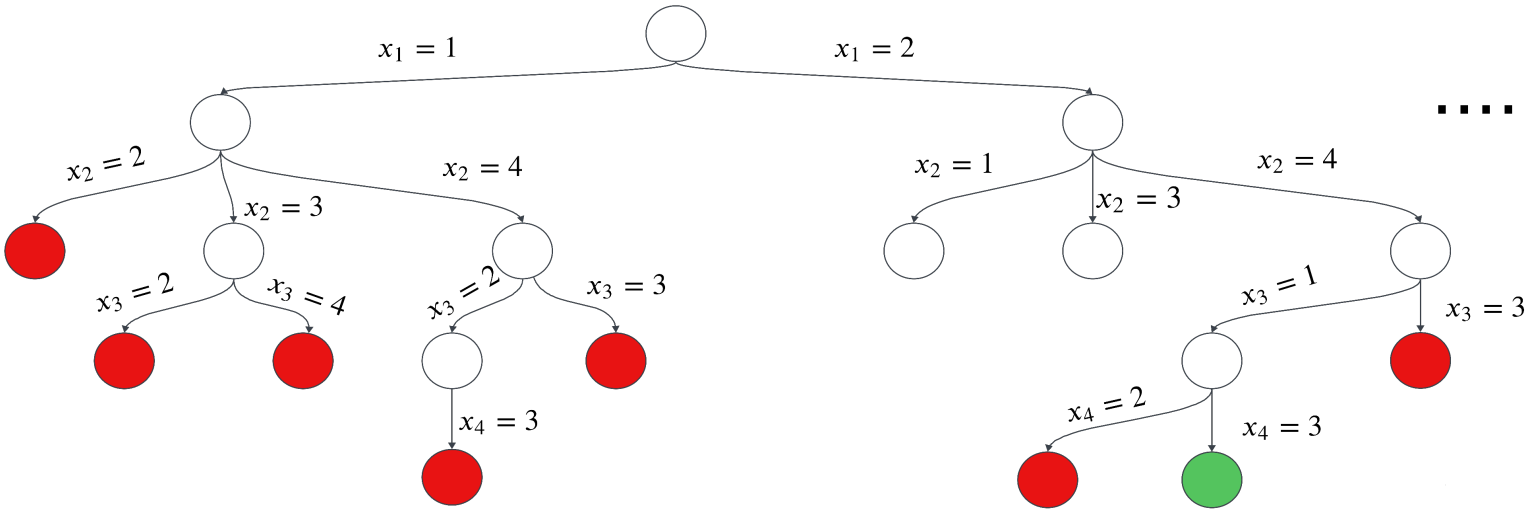

Figure: Decision tree for 4x4 Queen problem

Red nodes portray dead ends, thanks to the use of a test function, that makes sure that every two queens can never attack each other.

for technical reasons, I only checked where the first queen is placed in first and second row, same logic applies for the two remaining positions. for the 4×4 queen problem, we found two solutions, $(2, 4, 1, 3), (3, 1, 4, 2)$.

Algorithm Description

The solution of this problem is encoded in a list L of size n. Place algorithm tests whether the block in row k and column i is menaced by other queens or not, if so, we return False, and True if not.

NQueen algorithm uses backtracking to explore all possible configurations. If a valid configuration is found, it prints the solution. If a dead end is reached, the algorithm backtracks and tries a new configuration.

Test function

```
def place(k, i):
    # returns True if Queen is placed at k-th row and i-th
    # column otherwise return False
    for j in range(k):
        if L[j] == i or abs(L[j] - i) == abs(j - k):
            return False
    return True
```

Recursive Method

```
def NQueens(k, n):
    #using backtracking, this procedure prints all possible
    # combinations of solutions
    for i in range(n):
        if place(k, i):
            L[k] = i  # Place the queen at column i in row k
            if k == n - 1:
                print([x + 1 for x in L[0:n]])  # Print the solution
            else:
                NQueens(k + 1, n)

NQueens(0, N)
```

In these slides, we will not cover algorithm complexity in depth, to do so we will have to get into some prerequisites that are considered as pillars of this theory.

THANK YOU