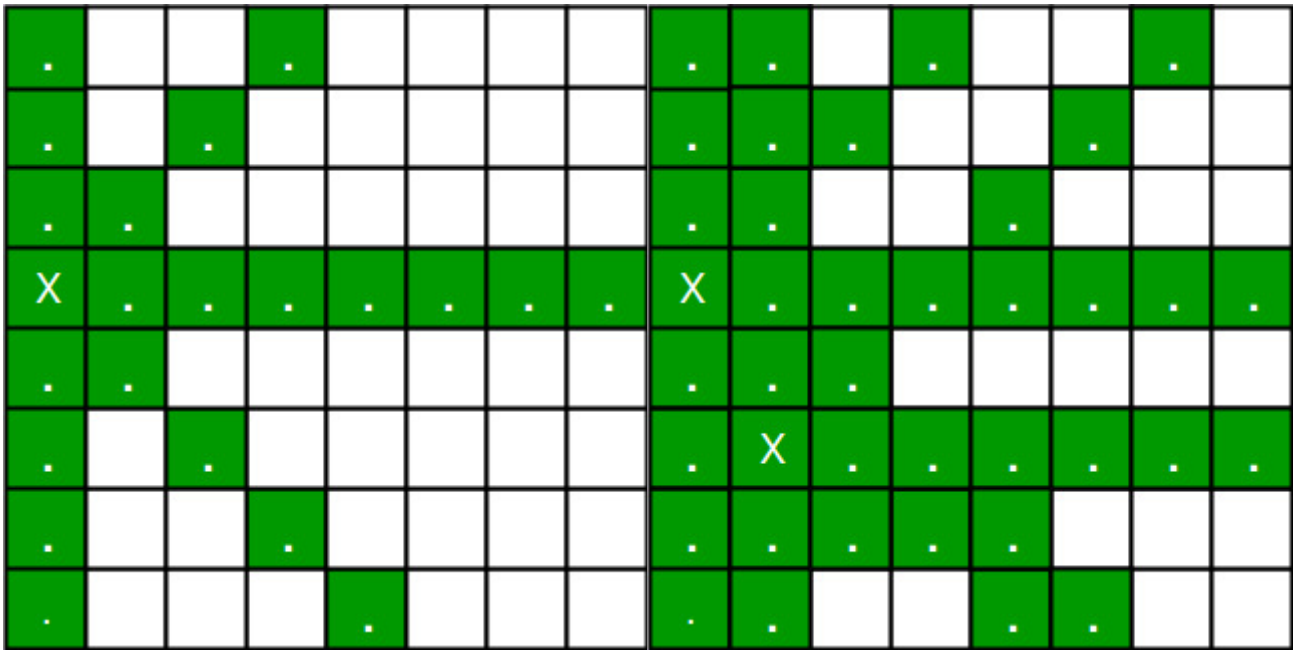
N Queens Problem using Backtracking | Branch and Bound Algorithm

In Branch and Bound solution, after building a partial solution, we figure out that there is no point going any deeper as we are going to hit a dead end.

Let's begin by describing backtracking solution. "The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes, then we backtrack and return false."



1. For the 1st Queen, there are total 8 possibilities as we can place 1st Queen in any row of first column. Let's place Queen 1 on row 3.
2. After placing 1st Queen, there are 7 possibilities left for the 2nd Queen. But wait, we don't really have 7 possibilities. We cannot place Queen 2 on rows 2, 3 or 4 as those cells are under attack from Queen 1. So, Queen 2 has only 8 – 3 = 5 valid positions left.
3. After picking a position for Queen 2, Queen 3 has even fewer options as most of the cells in its column are under attack from the first 2 Queens.

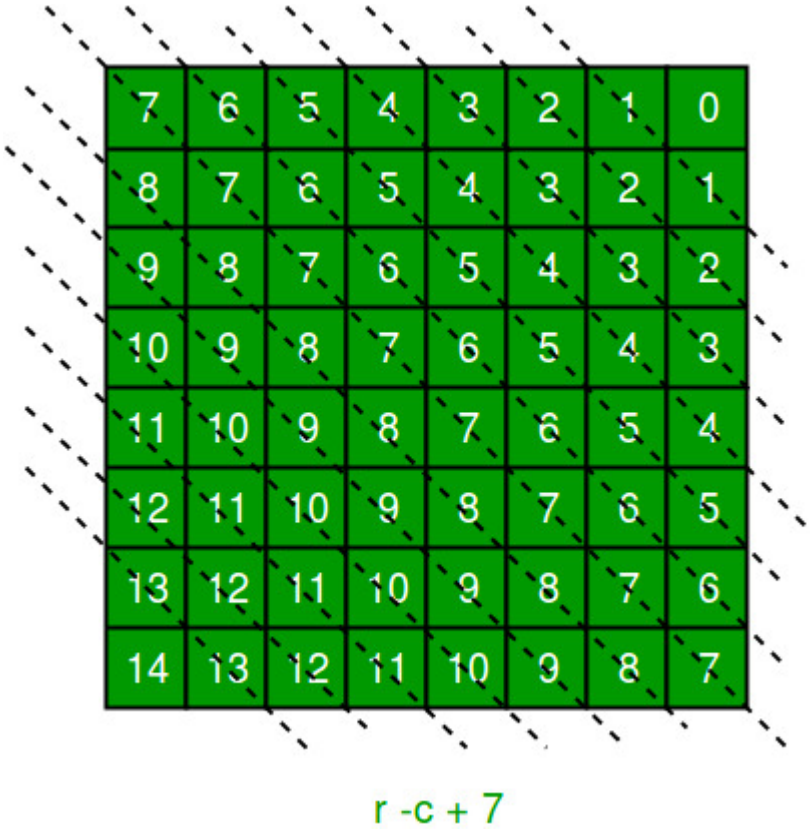
We need to figure out an efficient way of keeping track of which cells are under attack. In previous solution we kept an 8-by-8 Boolean matrix and update it each time we placed a queen, but that required linear time to update as we need to check for safe cells.
Basically, we have to ensure 4 things:
1. No two queens share a column.
2. No two queens share a row.

3. No two queens share a top-right to left-bottom diagonal.
4. No two queens share a top-left to bottom-right diagonal.

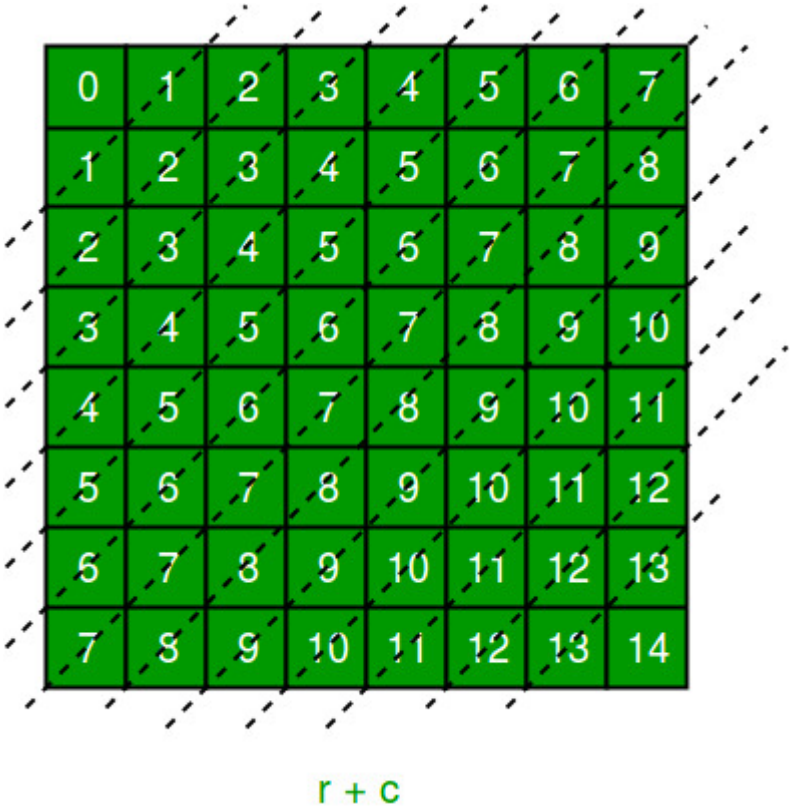
Number 1 is automatic because of the way we store the solution. For number 2, 3 and 4, we can perform updates in O(1) time. The idea is to keep **three Boolean arrays that tell us which rows and which diagonals are occupied**.

Lets do some pre-processing first. Let's create two N x N matrix one for / diagonal and other one for \ diagonal. Let's call them slashCode and backslashCode respectively. The trick is to fill them in such a way that two queens sharing a same /diagonal will have the same value in matrix slashCode, and if they share same \diagonal, they will have the same value in backslashCode matrix.
For an N x N matrix, fill slashCode and backslashCode matrix using below formula –
slashCode[row][col] = row + col
backslashCode[row][col] = row – col + (N-1)

Using above formula will result in below matrices



r -c + 7



The 'N – 1' in the backslash code is there to ensure that the codes are never negative because we will be using the codes as indices in an array.
Now before we place queen i on row j, we first check whether row j is used (use an array to store row info). Then we check whether slash code (j + i) or backslash code (j – i + 7) are used (keep two arrays that will tell us which diagonals are occupied). If yes, then we have to try a different location for queen i. If not, then we mark the row and the two diagonals as used and recurse on queen i + 1. After the recursive call returns and before we try another position for queen i, we need to reset the row, slash code and backslash code as unused again, like in the code from the previous notes.

```
def NqueensBranchAndBound(n):
    cols = [False] * n
    leftDiag = [False] * (2 * n - 1)
    rightDiag = [False] * (2 * n - 1)
    row = 0
    solve(row, cols, leftDiag, rightDiag, '')


def solve(row, cols, leftDiag, rightDiag, ssf):
    if row == len(cols):
        print(ssf+'.')
        return
    for col in range(len(cols)):
        if cols[col] == False and leftDiag[row - col + len(cols) - 1] ==
False and rightDiag[row + col] == False:
            cols[col] = True
            leftDiag[row - col + len(cols) - 1] = True
```

```
            rightDiag[row + col] = True
            solve(row + 1, cols, leftDiag, rightDiag, ssf + str(row) + '-' +
str(col)+' ')
            cols[col] = False
            leftDiag[row - col + len(cols) - 1] = False
            rightDiag[row + col] = False


NqueensBranchAndBound(32)
```