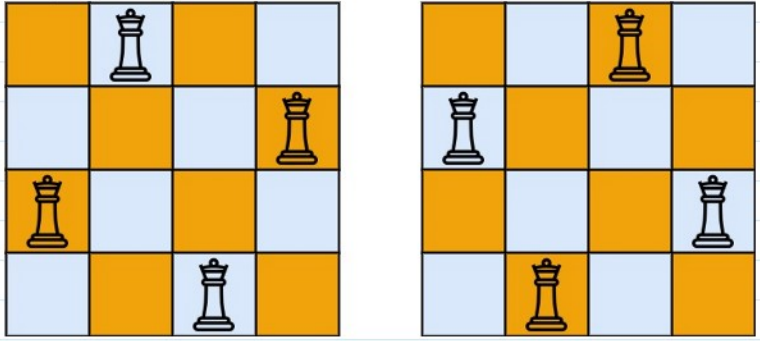
L17 N-Queens II

Tuesday, January 10, 2023 10:53 AM

The **n-queens** puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other.
Given an integer n, return *the number of distinct solutions to the **n-queens puzzle***.

Example 1:

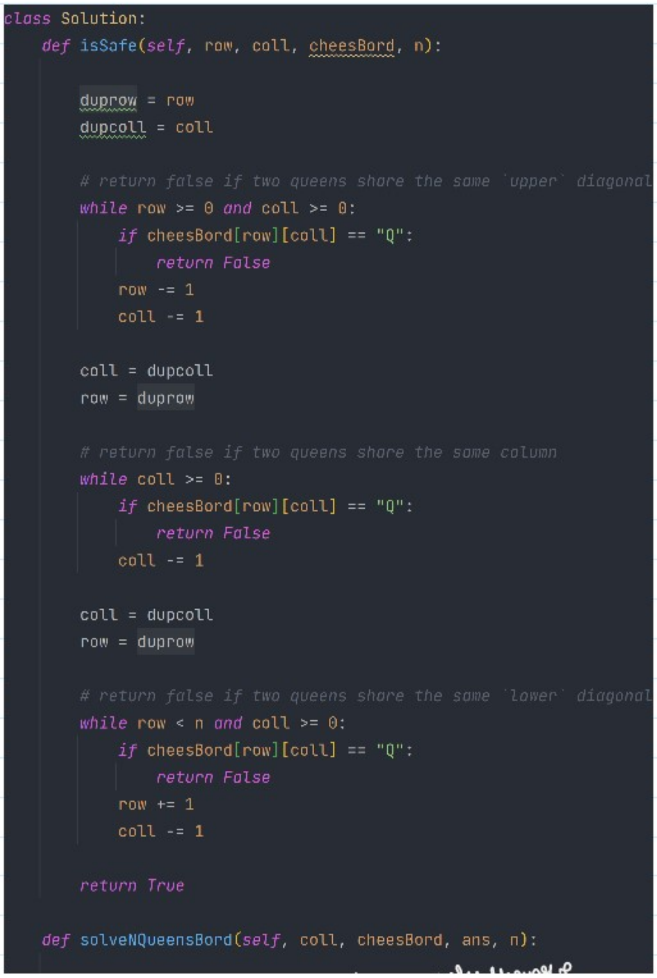


↳ Basically this is the Extended problem of N-Queen I

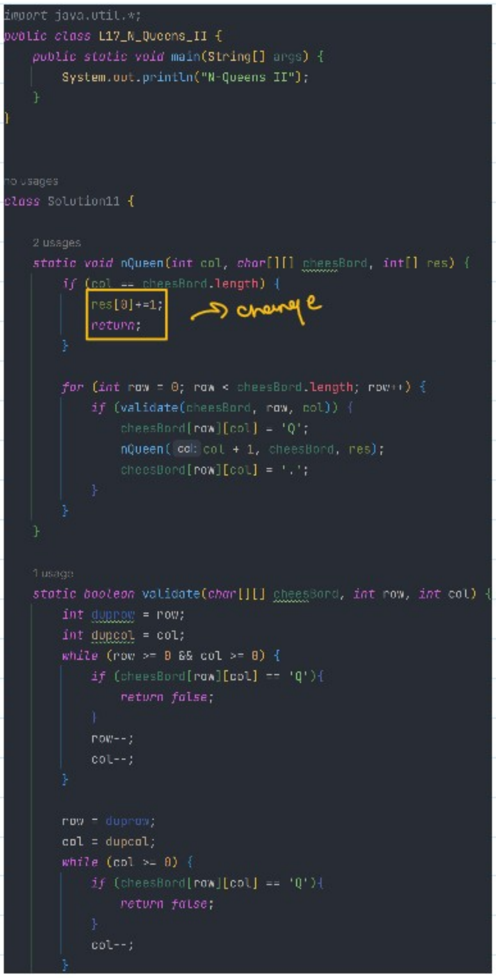
Input: n = 4
Output: 2
Explanation: There are two distinct solutions to the 4-queens puzzle as shown.

→ In N-Queen I problem our task is finding the All the possible path and Chessboard and follow the Rules of N-Queen placing

→ In this problem our task is only Count the possible of the Chessboard that follow all Rules

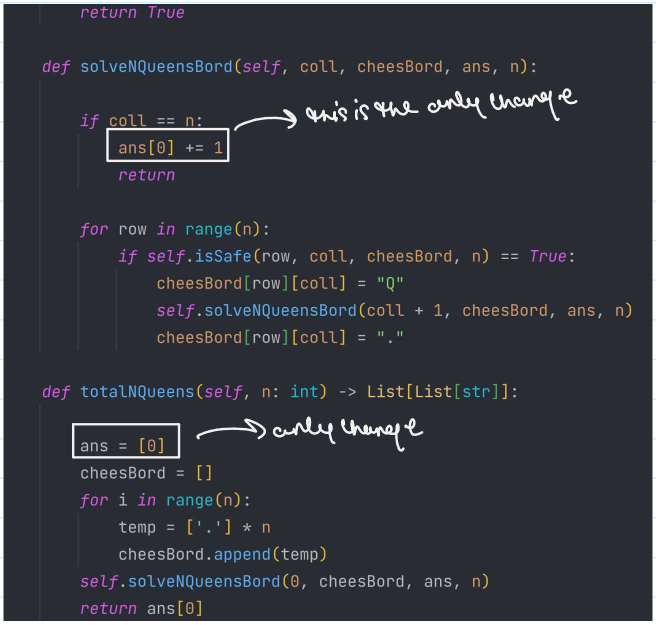


```python
class Solution:
    def isSafe(self, row, coll, cheesBord, n):

        duprow = row
        dupcoll = coll

        # return false if two queens share the same `upper` diagonal
        while row >= 0 and coll >= 0:
            if cheesBord[row][coll] == "Q":
                return False
            row -= 1
            coll -= 1

        coll = dupcoll
        row = duprow

        # return false if two queens share the same column
        while coll >= 0:
            if cheesBord[row][coll] == "Q":
                return False
            coll -= 1

        coll = dupcoll
        row = duprow

        # return false if two queens share the same `lower` diagonal
        while row < n and coll >= 0:
            if cheesBord[row][coll] == "Q":
                return False
            row += 1
            coll -= 1

        return True

    def solveNQueensBord(self, coll, cheesBord, ans, n):
```



```java
import java.util.*;
public class L17_N_Queens_II {
    public static void main(String[] args) {
        System.out.println("N-Queens II");
    }
}


no usages
class Solution11 {

    2 usages
    static void nQueen(int col, char[][] cheesBord, int[] res) {
        if (col == cheesBord.length) {
            res[0]+=1;
            return;
        }

        for (int row = 0; row < cheesBord.length; row++) {
            if (validate(cheesBord, row, col)) {
                cheesBord[row][col] = 'Q';
                nQueen( col: col + 1, cheesBord, res);
                cheesBord[row][col] = '.';
            }
        }
    }

    1 usage
    static boolean validate(char[][] cheesBord, int row, int col) {
        int duprow = row;
        int dupcol = col;
        while (row >= 0 && col >= 0) {
            if (cheesBord[row][col] == 'Q'){
                return false;
            }
            row--;
            col--;
        }

        row = duprow;
        col = dupcol;
        while (col >= 0) {
            if (cheesBord[row][col] == 'Q'){
                return false;
            }
            col--;
        }
```

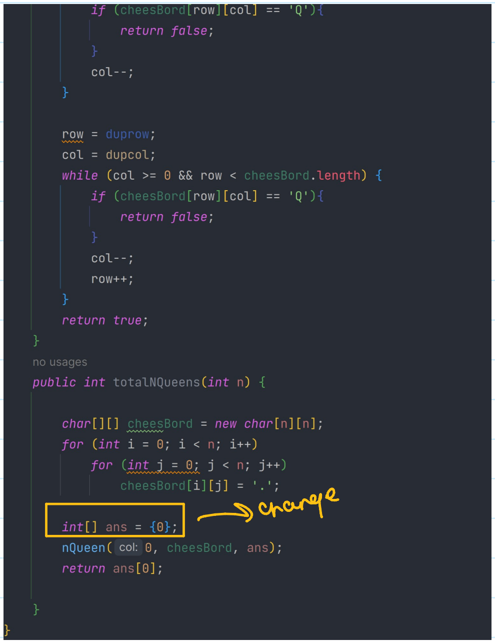
→ change



```python
        return True

    def solveNQueensBord(self, coll, cheesBord, ans, n):

        if coll == n:
            ans[0] += 1
            return

        for row in range(n):
            if self.isSafe(row, coll, cheesBord, n) == True:
                cheesBord[row][coll] = "Q"
                self.solveNQueensBord(coll + 1, cheesBord, ans, n)
                cheesBord[row][coll] = "."

    def totalNQueens(self, n: int) -> List[List[str]]:

        ans = [0]
        cheesBord = []
        for i in range(n):
            temp = ['.'] * n
            cheesBord.append(temp)
        self.solveNQueensBord(0, cheesBord, ans, n)
        return ans[0]
```

→ this is the only change

→ only change



```java
            if (cheesBord[row][col] == 'Q'){
                return false;
            }
            col--;
        }

        row = duprow;
        col = dupcol;
        while (col >= 0 && row < cheesBord.length) {
            if (cheesBord[row][col] == 'Q'){
                return false;
            }
            col--;
            row++;
        }
        return true;
    }
    no usages
    public int totalNQueens(int n) {

        char[][] cheesBord = new char[n][n];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                cheesBord[i][j] = '.';

        int[] ans = {0};
        nQueen( col: 0, cheesBord, ans);
        return ans[0];

    }
}
```

→ change

Time Complexity = $O(N!) \times N$

→ for all the possible Combination Recursion

→ for all the Other Searching

Space Complexity = $O(N^2)$

→ beause we used a Nested Arraey of the Size N for Storing the answer