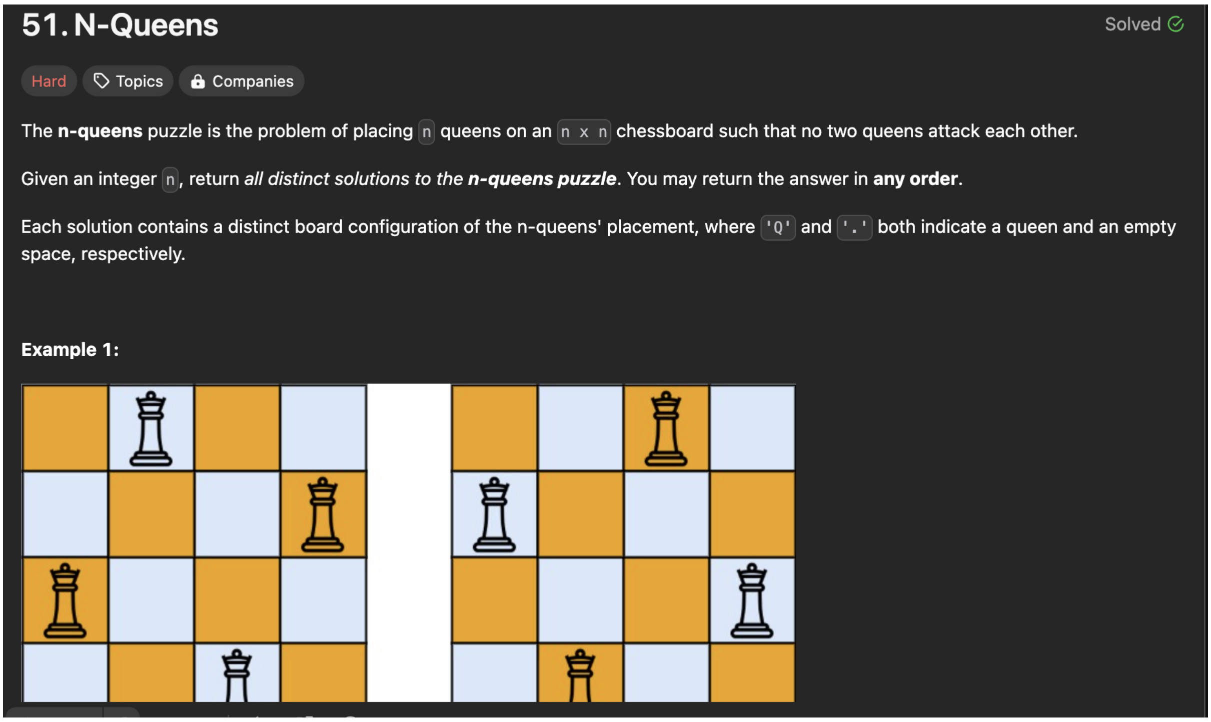

51. N-Queens

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 *all distinct solutions to the **n-queens puzzle***. You may return the answer in **any order**.

Each solution contains a distinct board configuration of the n-queens' placement, where `'Q'` and `'.'` both indicate a queen and an empty space, respectively.

Example 1:

```
class Solution {
    var ans = [[String]]()
    func solveNQueens(_ n: Int) -> [[String]] {
        let a:[Int] = Array(repeating: 0, count:n)
        var matrix:[[Int]] = Array(repeating: a, count:n)
        solve(0,n,&matrix,0)
        return ans
    }

    func solve(_ column:Int,_ n: Int, _ matrix:inout [[Int]], _ queenCount: Int) {
        if(column == n && queenCount == n) {
            insertIntoans(matrix)
            return
        }

        for i in 0..<n {
            if(isQueenSafe(i,column,matrix)) {
                matrix[i][column] = 1
                solve(column+1,n,&matrix,queenCount+1)
                matrix[i][column] = 0
            }
        }
```

```
}
func insertIntoans(_ matrix:[[Int]]) {
    var temp = [String]()
    for i in matrix {
        var a = ""
        for j in 0..<i.count{
            a = a + ((i[j]==0) ? "." : "Q")
        }
        temp.append(a)
    }

    ans.append(temp)

}

func isQueenSafe(_ row: Int, _ column: Int, _ matrix:[[Int]]) -> Bool {
    var x = row
    var y = column

    // checkrow

    while(y>=0) {
        if(matrix[x][y] == 1) {
            return false
        }
        y = y - 1
    }

    x = row
    y = column

    while(y>=0 && x >= 0) {
        if(matrix[x][y] == 1) {
            return false
        }
        x = x - 1
        y = y - 1
        }

        x = row
        y = column
        while(y>=0 && x < matrix.count) {
            if(matrix[x][y] == 1) {
                return false
            }
            x = x + 1
            y = y - 1
        }
        return true
    }
}
```