

N-Queens

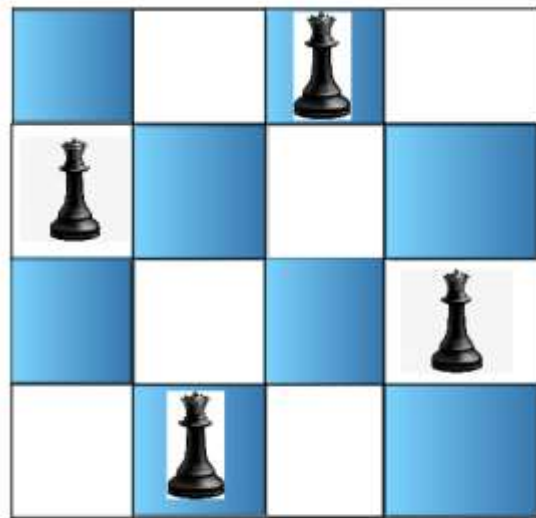
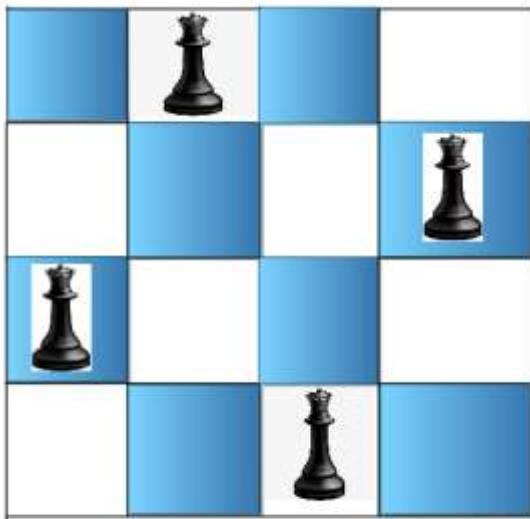
Problem Statement: The n-queens is the problem of placing n queens on $n \times n$ chessboard such that no two queens can attack each other. Given an integer n, return all distinct solutions to the n -queens puzzle. Each solution contains a distinct boards configuration of the queen's placement, where 'Q' and '.' indicate queen and empty space respectively.

Examples:

Input: $n = 4$

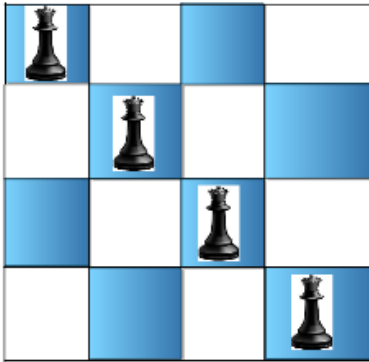
Output: `[[".Q..", "...Q", "Q...", "..Q."], ["..Q.", "Q...", "...Q", ".Q.."]]`

Explanation: There exist two distinct solutions to the 4-queens puzzle as shown below



Two arrangements possible for 4 queens

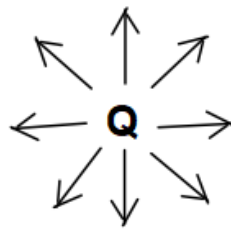
Let us first understand how can we place queens in a chessboard so that no attack on either of them can take place.



Rules for n-Queen in chessboard

1. Every row should have one Queen
2. Every column should have one Queen
3. No two queens can attack each other

Queen attack can take place in following way

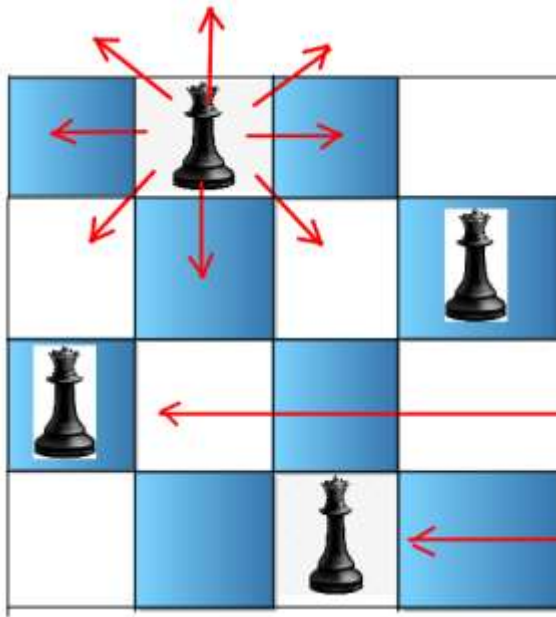


Solution

Disclaimer: Don't jump directly to the solution, try it out yourself first.

Solution 1:

Intuition: Using the concept of Backtracking, we will place Queen at different positions of the chessboard and find the right arrangement where all the n queens can be placed on the $n \times n$ grid.

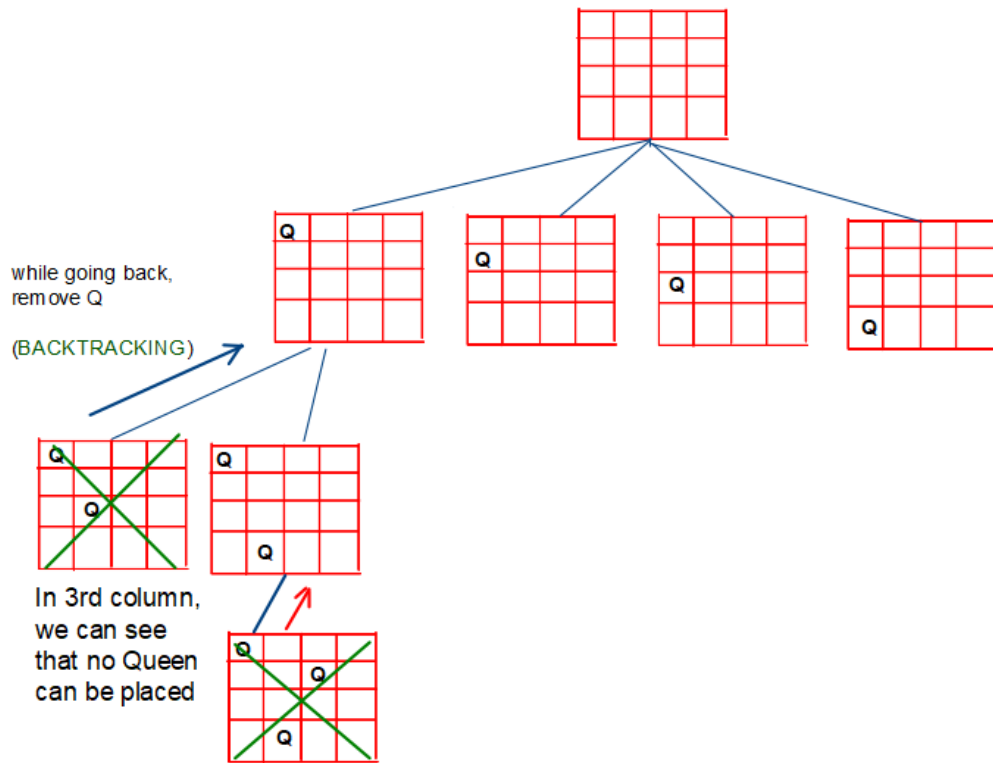


Here is a position where queen will not attack other queen

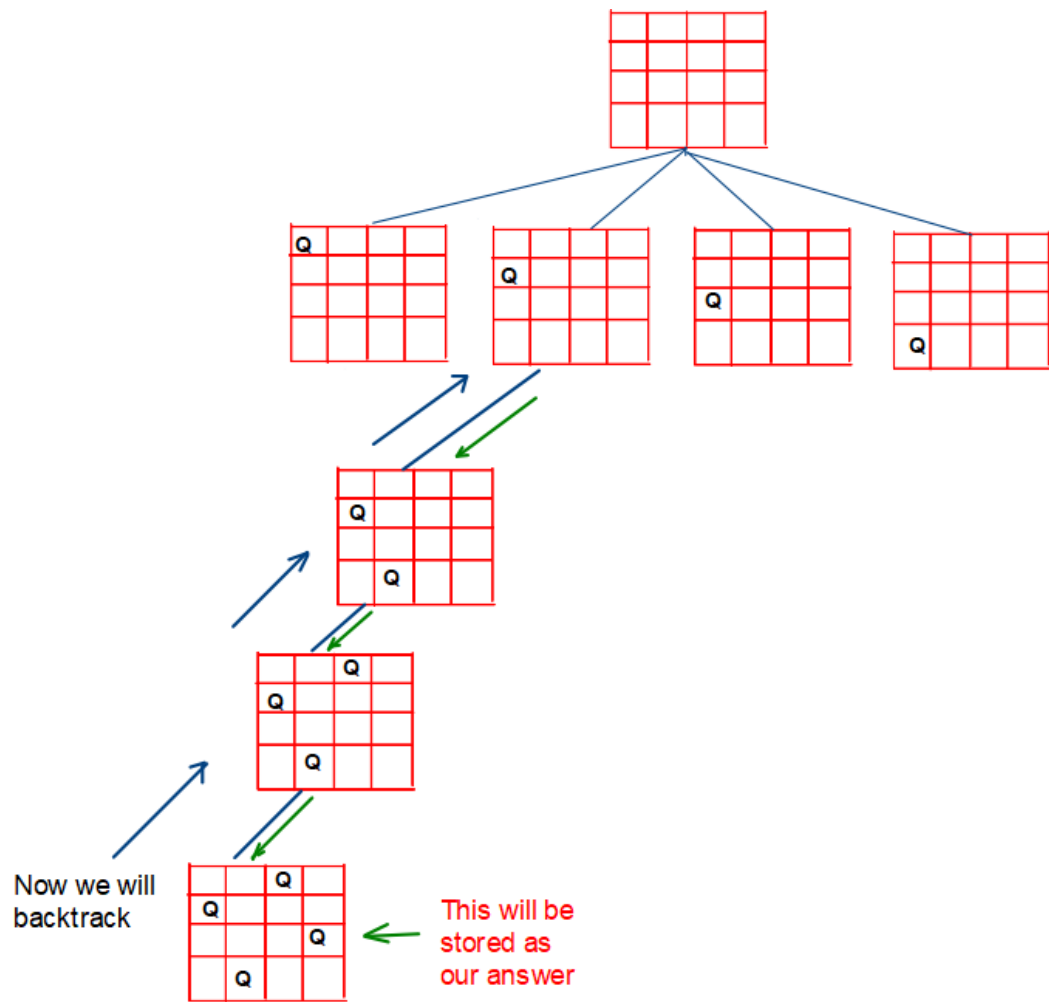
Similar case with other queens as well.

Approach:

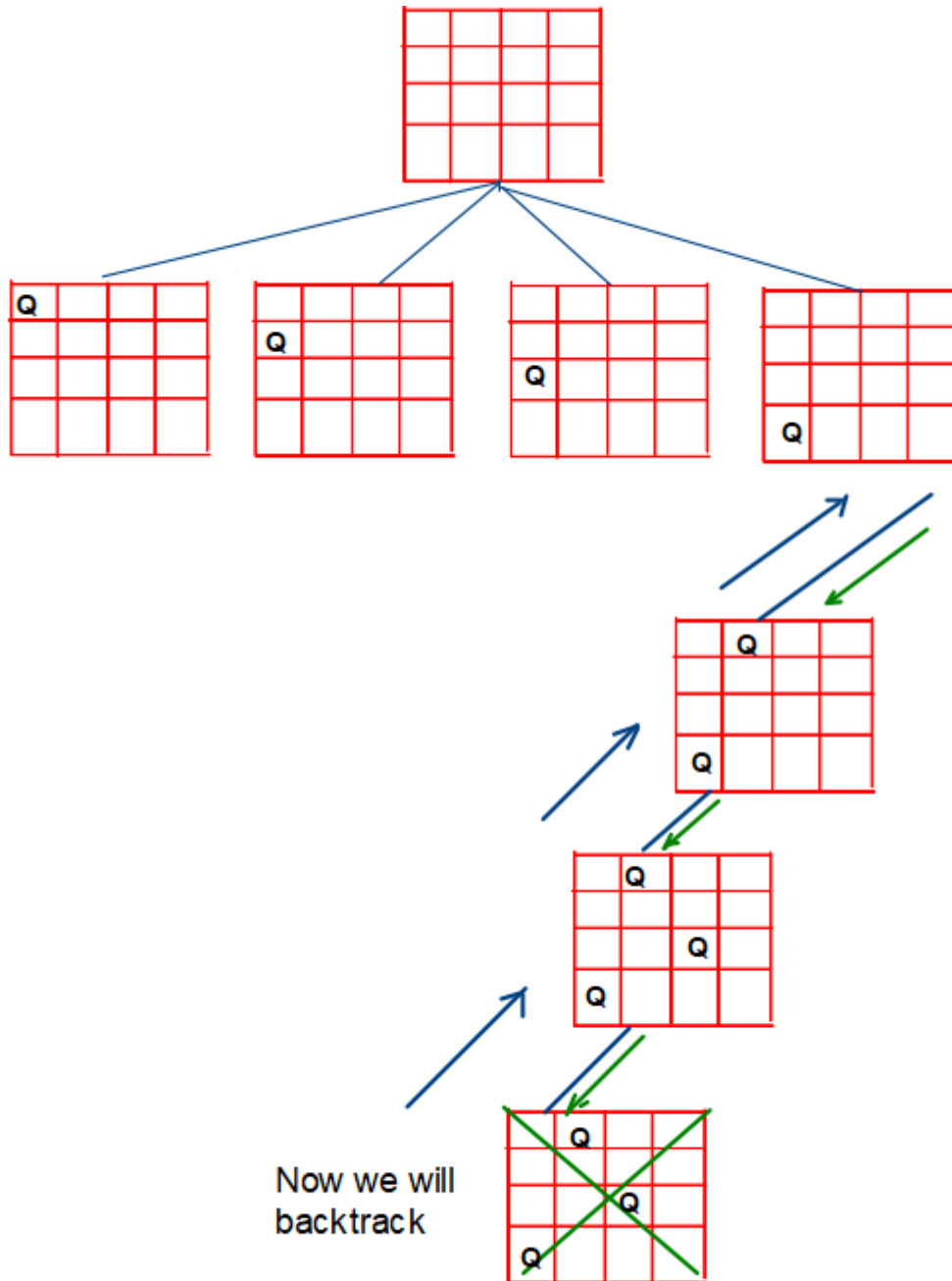
1st position: This is the position where we can see no possible arrangement is found where all queens can be placed since, at the 3rd column, the Queen will be killed at all possible positions of row.



2nd position: One of the correct possible arrangements is found. So we will store it as our answer.



3rd position: One of the correct possible arrangements is found. So we will store it as our answer.



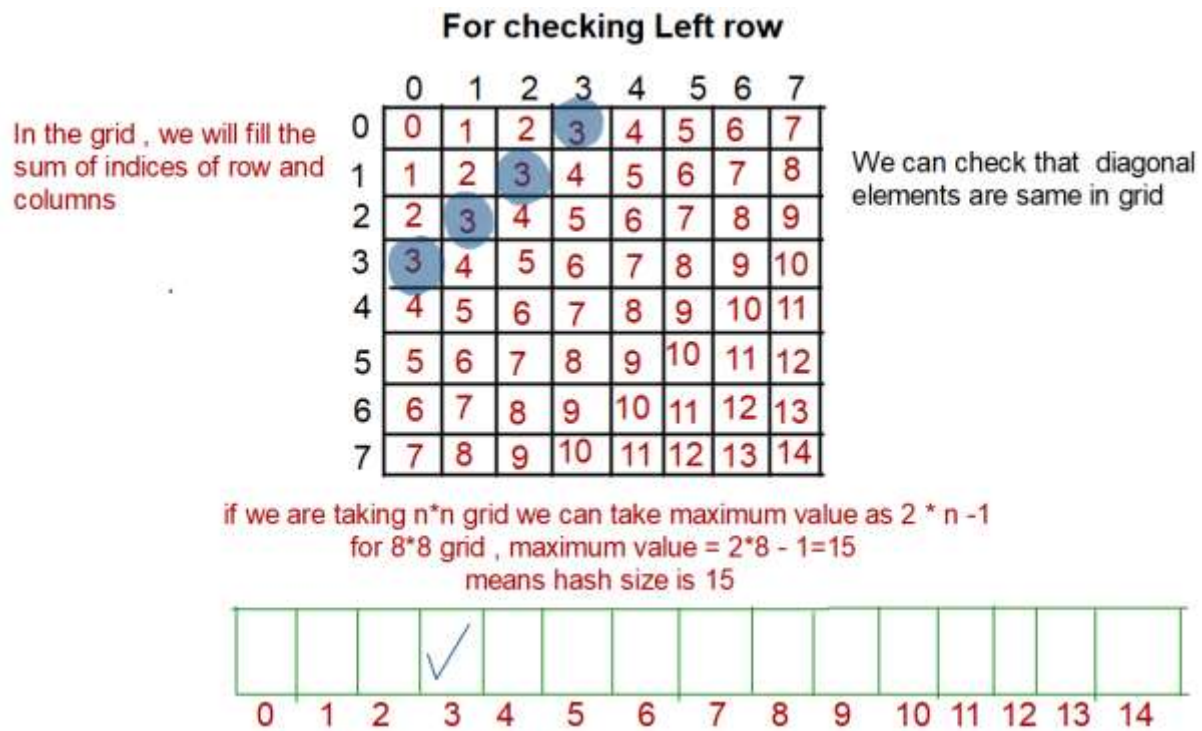
Solution 2:

Intuition: This is the optimization of the issafe function. In the previous issafe function, we need $O(N)$ for a row, $O(N)$ for the column, and $O(N)$ for the diagonal.

Here, we will use hashing to maintain a list to check whether that position can be the right one or not.

Approach:

For checking Left row elements



For checking upper diagonal and lower diagonal

For checking upper diagonal and lower diagonal

In the grid , we will fill the
(n-1) + (row-col)

	0	1	2	3	4	5	6	7
0	7	8	9	10	11	12	13	14
1	6	7	8	9	10	11	12	13
2	5	6	7	8	9	10	11	12
3	4	5	6	7	8	9	10	11
4	3	4	5	6	7	8	9	10
5	2	3	4	5	6	7	8	9
6	1	2	3	4	5	6	7	8
7	0	1	2	3	4	5	6	7

We can check that diagonal
elements are same in grid

if we are taking $n \times n$ grid we can take maximum value as $2 * n - 1$
for 8×8 grid , maximum value = $2 * 8 - 1 = 15$
means hash size is 15

			✓												
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	