14. LC 51 N-Queens

51. N-Queens

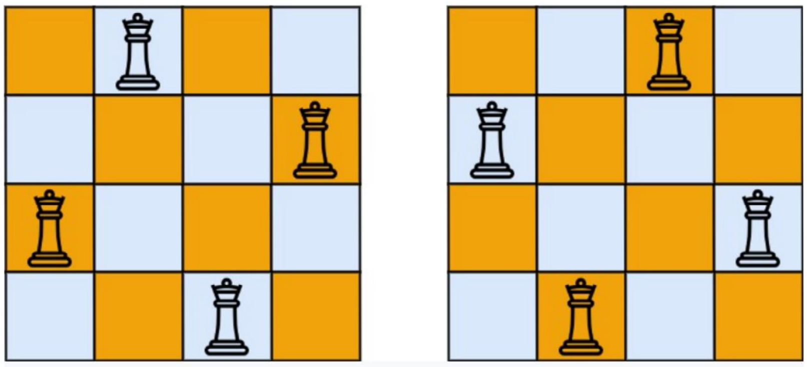
Hard 7403 178 Add to List Share

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**.

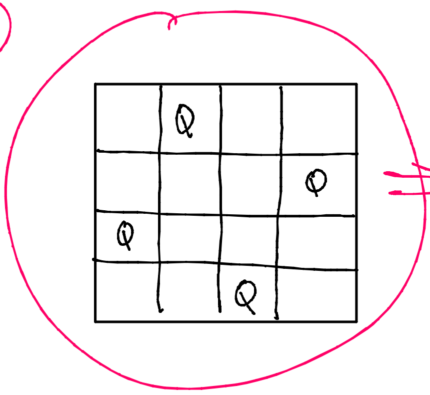
Each solution contains a distinct board configuration of the n-queens' placement, where `'Q'` and `'.'` both indicate a queen and an empty space, respectively.

Example 1:

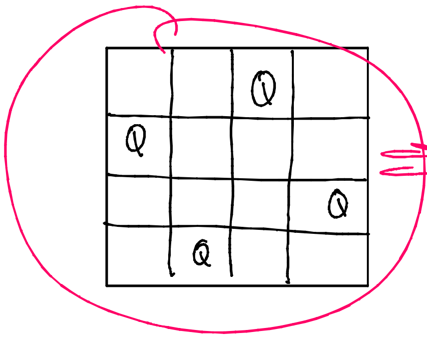


```
Input: n = 4
Output: [[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]
Explanation: There exist two distinct solutions to the 4-queens puzzle as
shown above
```

N=4



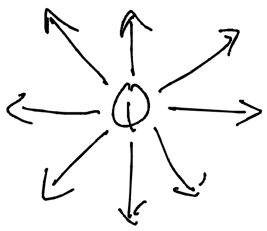
⇒ One solution



⇒ 2nd solution

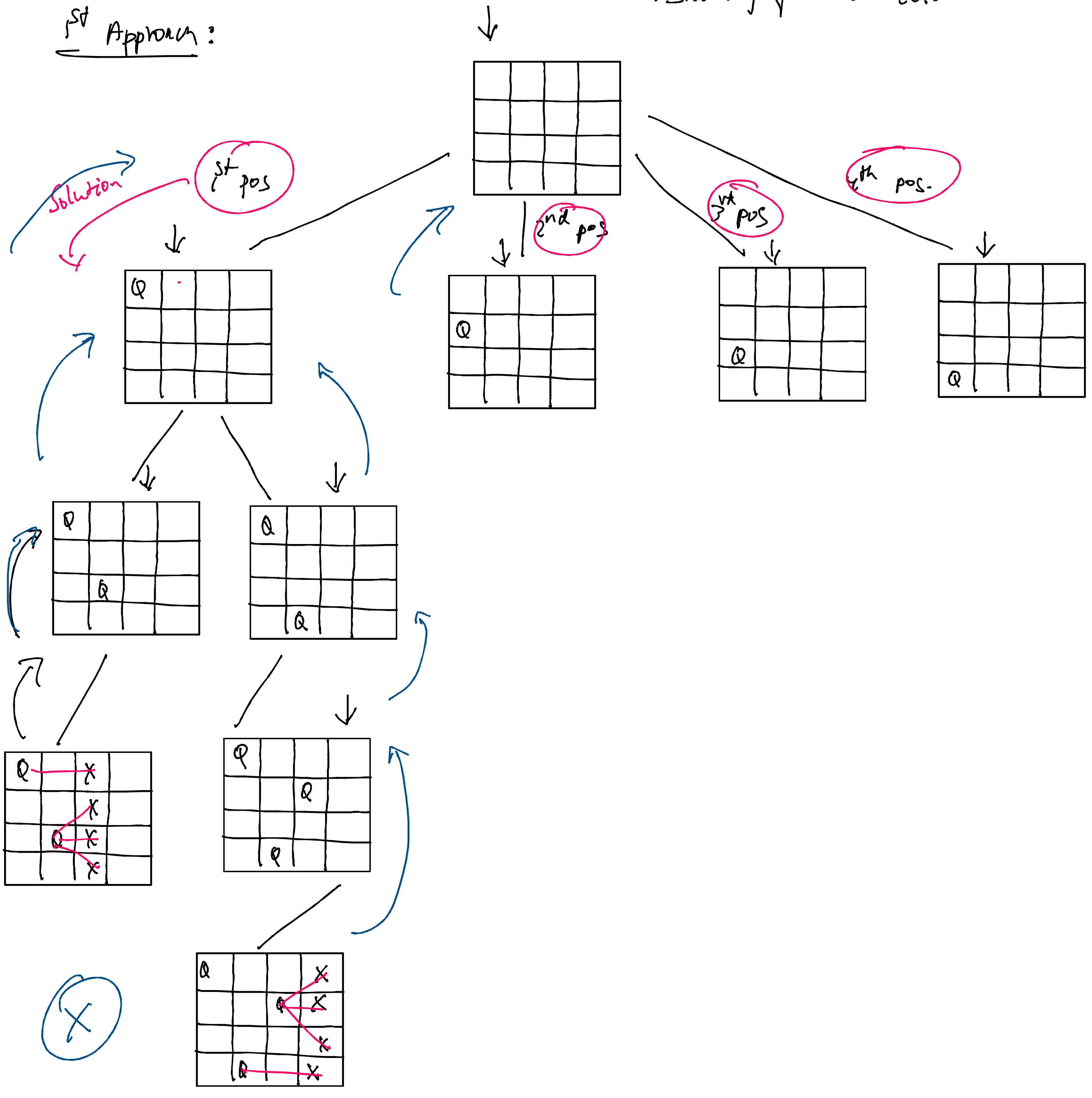
Three Rules

- → Every Columns → 1 Q
- → Every Rows → 1 Q
- → None of the 'Q' attack each other

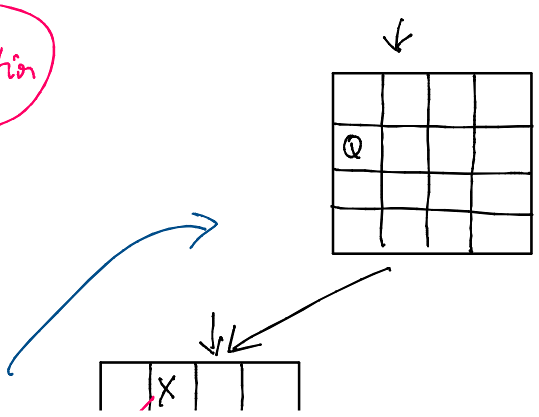


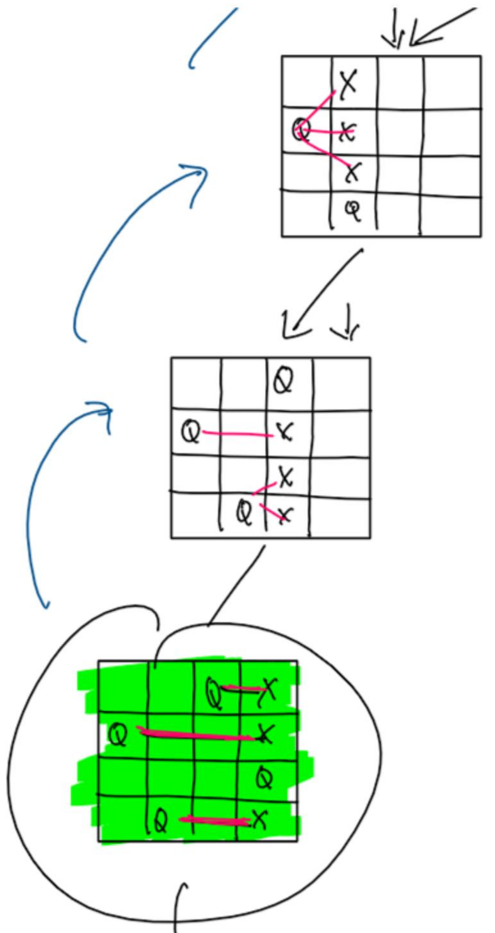
* Starting from 0th col.

1st Approach :



2nd position

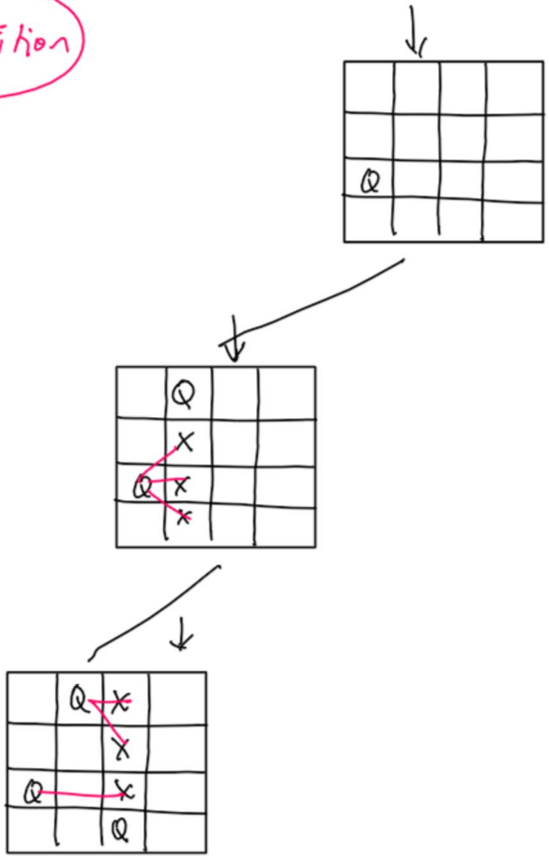


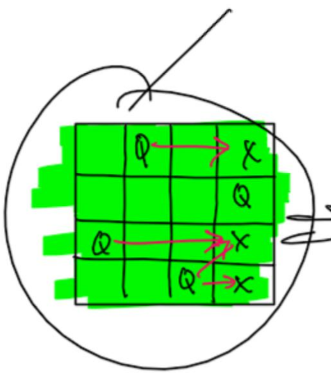


got a output

so store it.

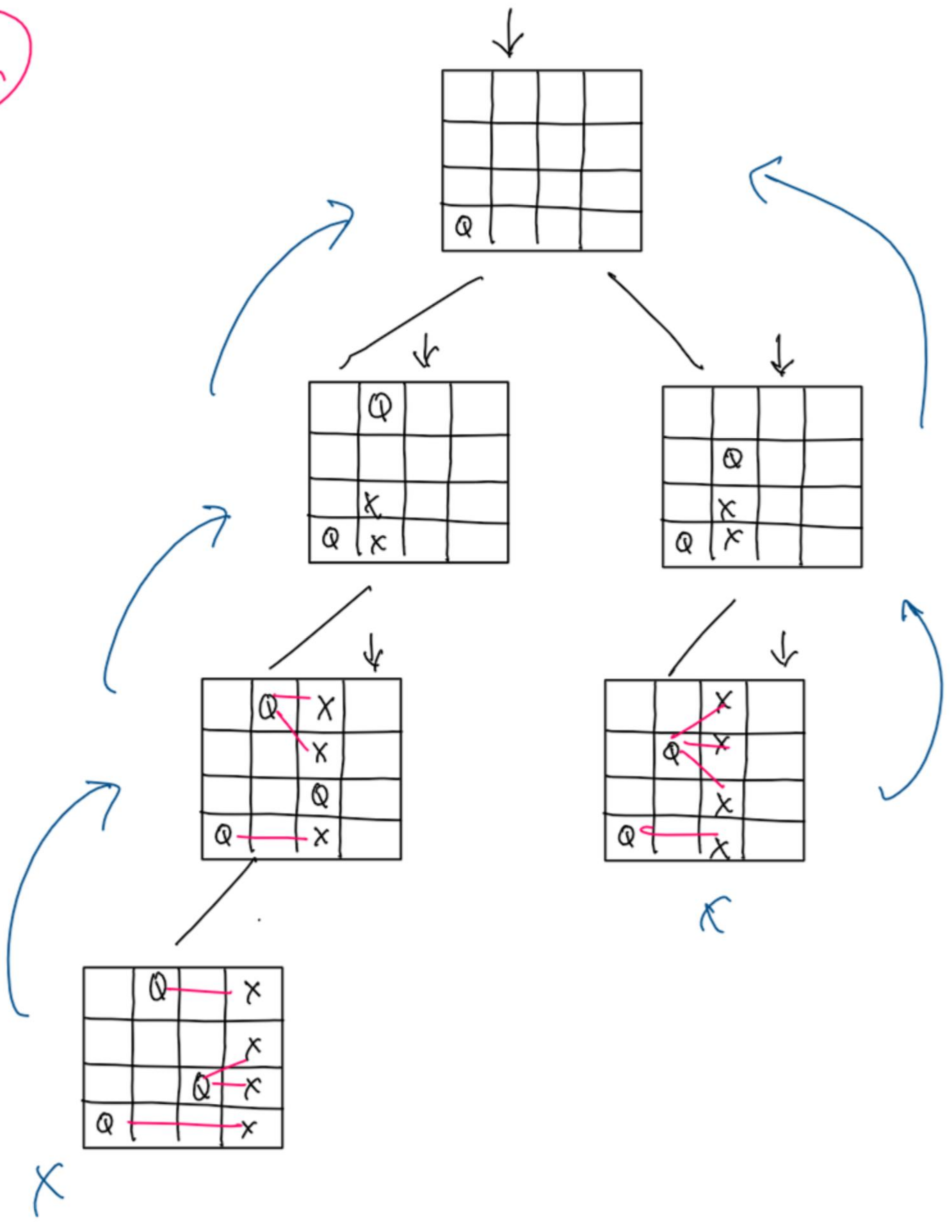
3rd position



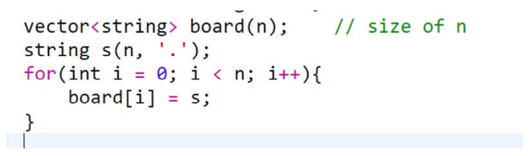


$\Rightarrow$ One more solution, so store it.

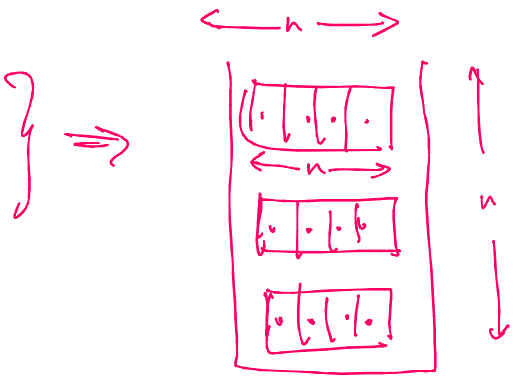
4th position



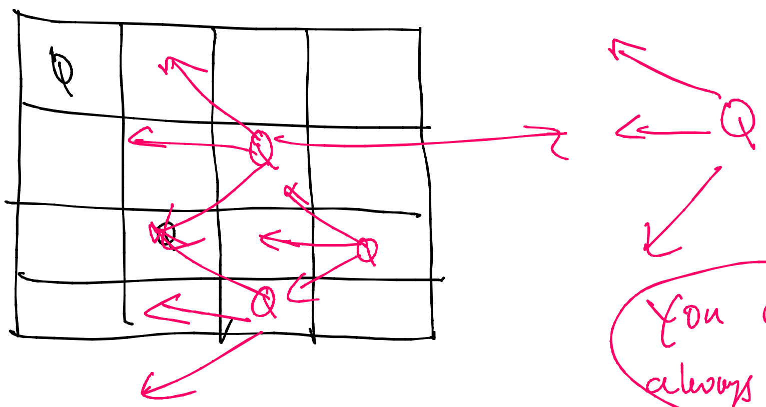
```
f(col)
{
    for(i = 0 -> n-1)
    { if(Lill -> v)
{
  if(fill → ✓)
        row col = Q
        f(col+1)
  }     row col = empty
}
```



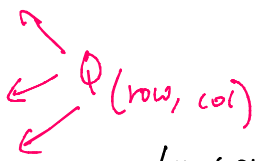
```
vector<string> board(n);    // size of n
string s(n, '.');
for(int i = 0; i < n; i++){
     board[i] = s;
}
```



is Safe function ()

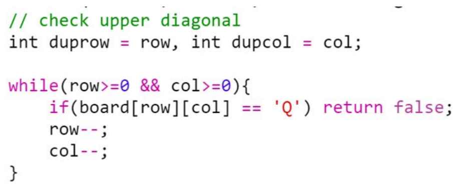


You check only those side always



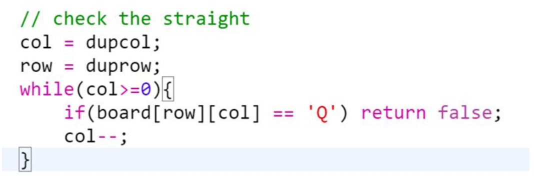
Q(row, col)

for upwards ↑ row-- , col--



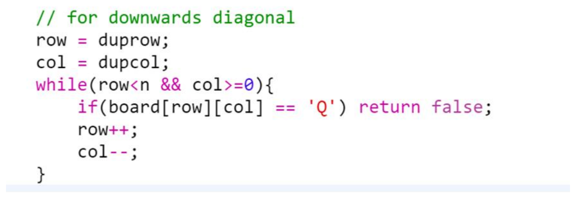
```
// check upper diagonal
int duprow = row, int dupcol = col;

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

for straghit ← col --

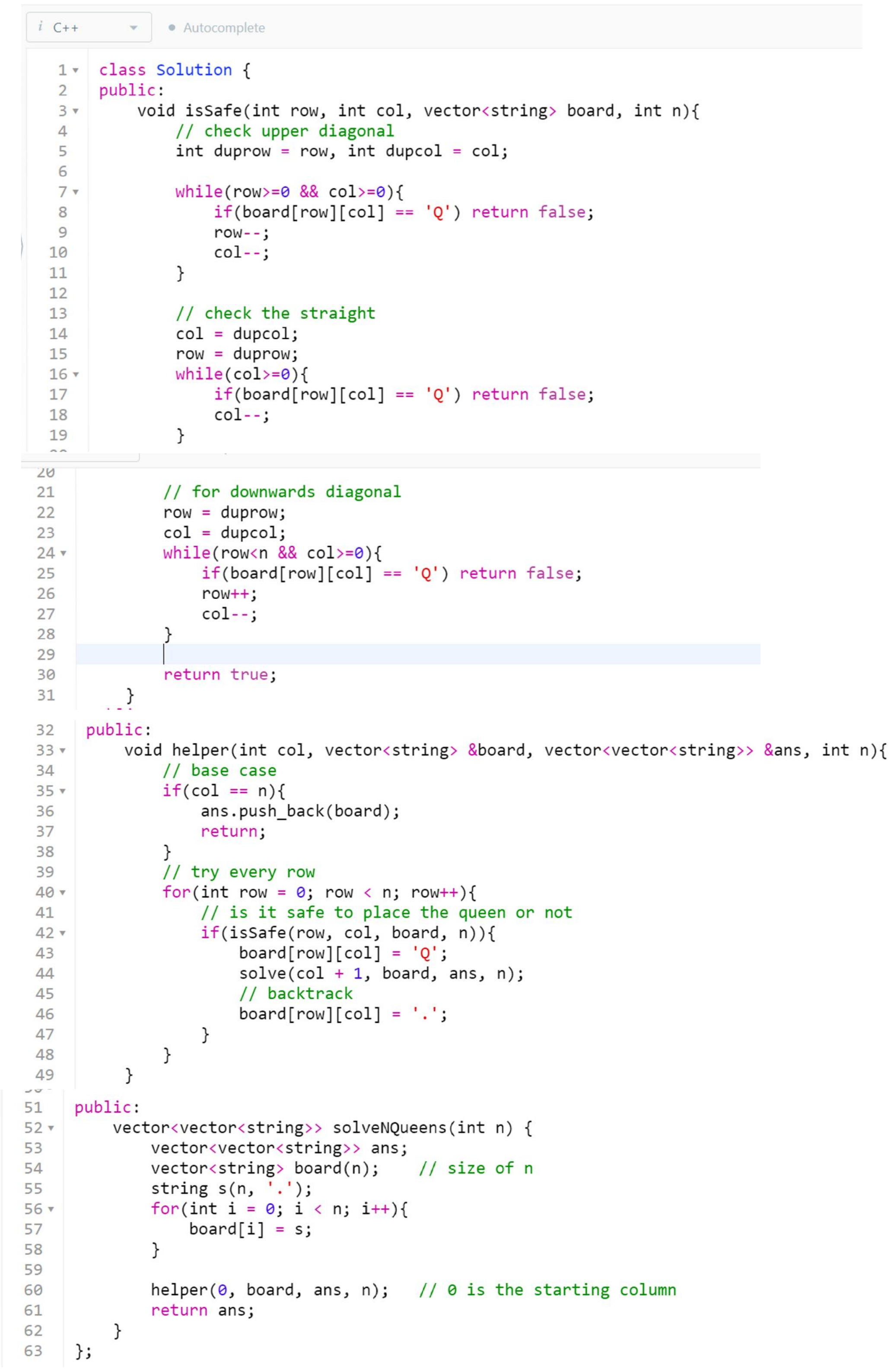


```
// check the straight
col = dupcol;
row = duprow;
while(col>=0){
    if(board[row][col] == 'Q') return false;
    col--;
}
```

for downwards ↙ row ++
col --



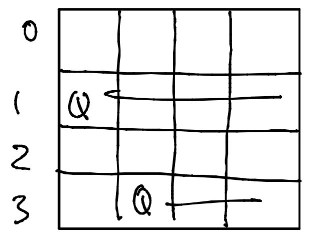
```
// for downwards diagonal
row = duprow;
col = dupcol;
while(row<n && col>=0){
    if(board[row][col] == 'Q') return false;
    row++;
    col--;
}
```

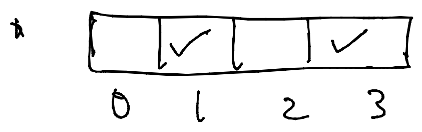


```
class Solution {
public:
    void isSafe(int row, int col, vector<string> board, int n){
        // check upper diagonal
        int duprow = row, int dupcol = col;

        while(row>=0 && col>=0){
            if(board[row][col] == 'Q') return false;
            row--;
            col--;
        }

        // check the straight
        col = dupcol;
        row = duprow;
        while(col>=0){
            if(board[row][col] == 'Q') return false;
            col--;
        }

        // for downwards diagonal
        row = duprow;
        col = dupcol;
        while(row<n && col>=0){
            if(board[row][col] == 'Q') return false;
            row++;
            col--;
        }

        return true;
    }
public:
    void helper(int col, vector<string> &board, vector<vector<string>> &ans, int n){
        // base case
        if(col == n){
            ans.push_back(board);
            return;
        }
        // try every row
        for(int row = 0; row < n; row++){
            // is it safe to place the queen or not
            if(isSafe(row, col, board, n)){
                board[row][col] = 'Q';
                solve(col + 1, board, ans, n);
                // backtrack
                board[row][col] = '.';
            }
        }
    }
public:
    vector<vector<string>> solveNQueens(int n) {
        vector<vector<string>> ans;
        vector<string> board(n);    // size of n
        string s(n, '.');
        for(int i = 0; i < n; i++){
            board[i] = s;
        }

        helper(0, board, ans, n);   // 0 is the starting column
        return ans;
    }
};
```

2nd Approach : Optimization of isSafe().

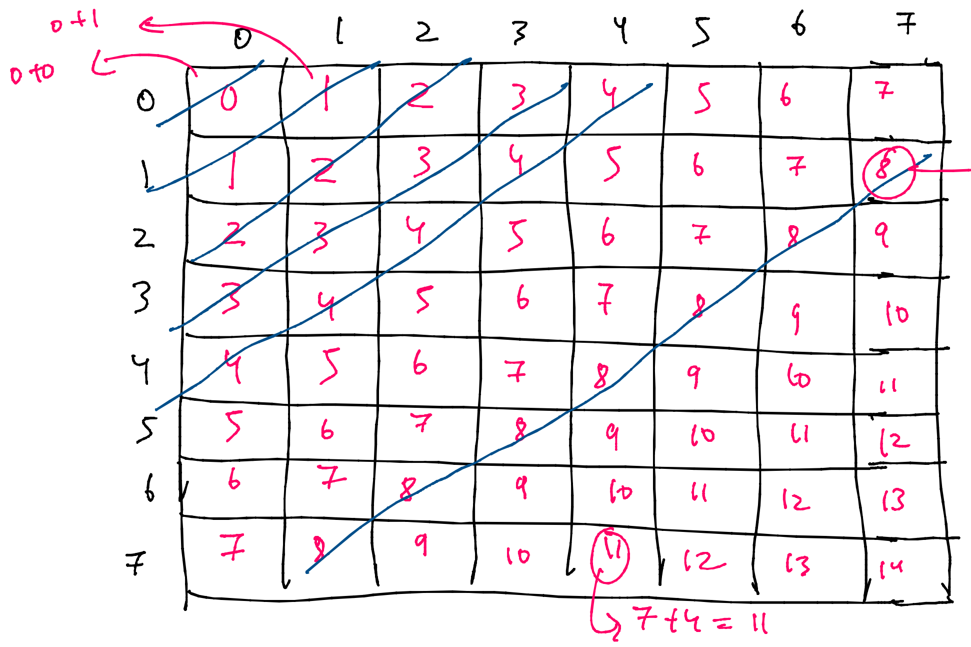
In previous approach we need O(N) for each row, col and diagonal. Can be optimized using hashing to maintain a list heck whether that position can be right one or not.



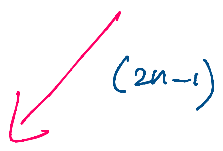


← only for this direction (left side cond).

for diagonal :



The moment you place the queen add r+c = 7+1 = 8, mark it in the hashtable.



lower diagonal

(2n-1) size

→ If you're taking n*n grid we can take minini val as (2n-1).

For 8*8, 2*8 - 1 = 5 (has size of 15)

upper diagonal

	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

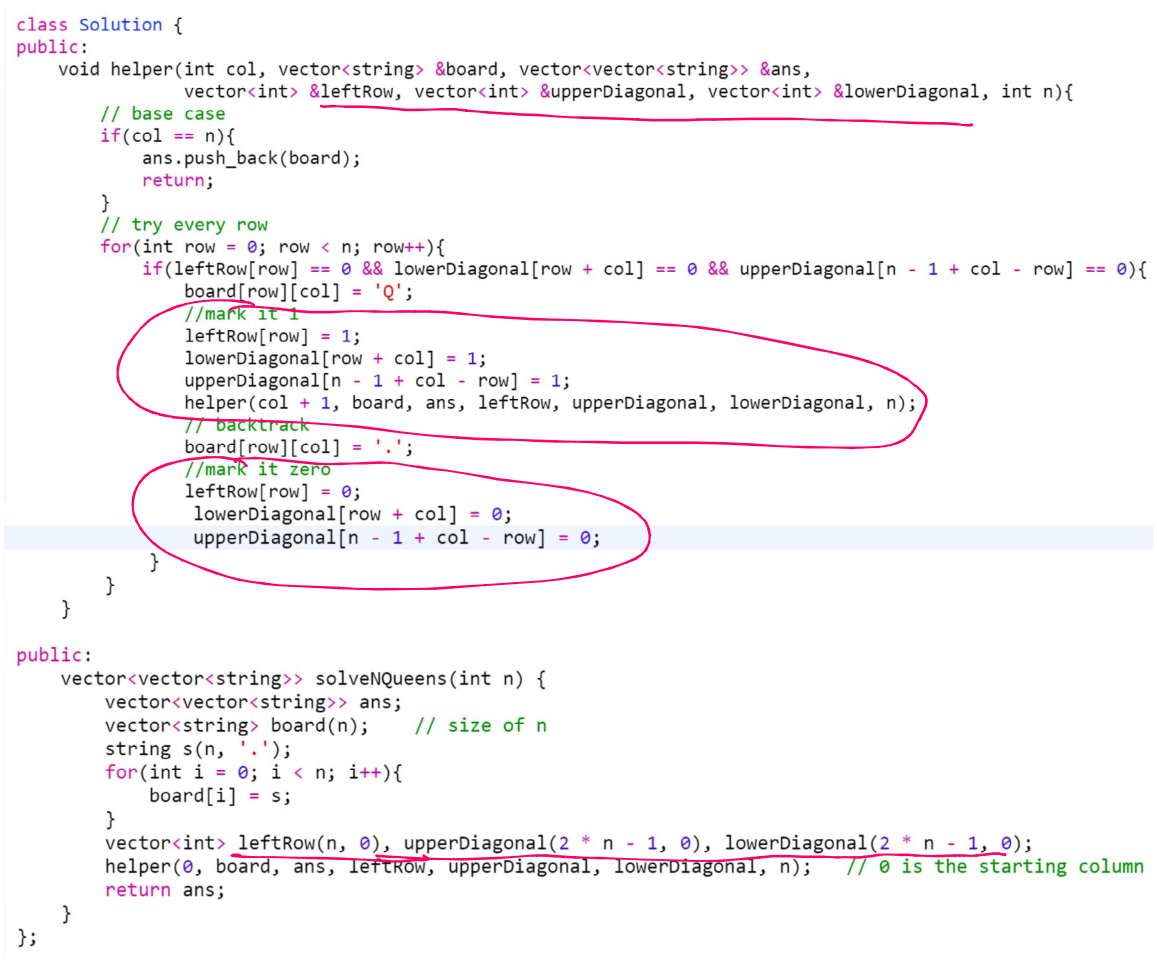
formula to fill

$$(n-1) + (col - row)$$

$$= 7-0 + (7-0) = \boxed{14}$$

2n-1

has[n-1 + (col - row)] = 1



```
class Solution {
public:
    void helper(int col, vector<string> &board, vector<vector<string>> &ans,
                vector<int> &leftRow, vector<int> &upperDiagonal, vector<int> &lowerDiagonal, int n){
        // base case
        if(col == n){
            ans.push_back(board);
            return;
        }
        // try every row
        for(int row = 0; row < n; row++){
            if(leftRow[row] == 0 && lowerDiagonal[row + col] == 0 && upperDiagonal[n - 1 + col - row] == 0){
                board[row][col] = 'Q';
                //mark it 1
                leftRow[row] = 1;
                lowerDiagonal[row + col] = 1;
                upperDiagonal[n - 1 + col - row] = 1;
                helper(col + 1, board, ans, leftRow, upperDiagonal, lowerDiagonal, n);
                // backtrack
                board[row][col] = '.';
                //mark it zero
                leftRow[row] = 0;
                 lowerDiagonal[row + col] = 0;
                 upperDiagonal[n - 1 + col - row] = 0;
            }
        }
    }

public:
    vector<vector<string>> solveNQueens(int n) {
        vector<vector<string>> ans;
        vector<string> board(n);    // size of n
        string s(n, '.');
        for(int i = 0; i < n; i++){
            board[i] = s;
        }
        vector<int> leftRow(n, 0), upperDiagonal(2 * n - 1, 0), lowerDiagonal(2 * n - 1, 0);
        helper(0, board, ans, leftRow, upperDiagonal, lowerDiagonal, n);   // 0 is the starting column
        return ans;
    }
};
```