51. N-Queens

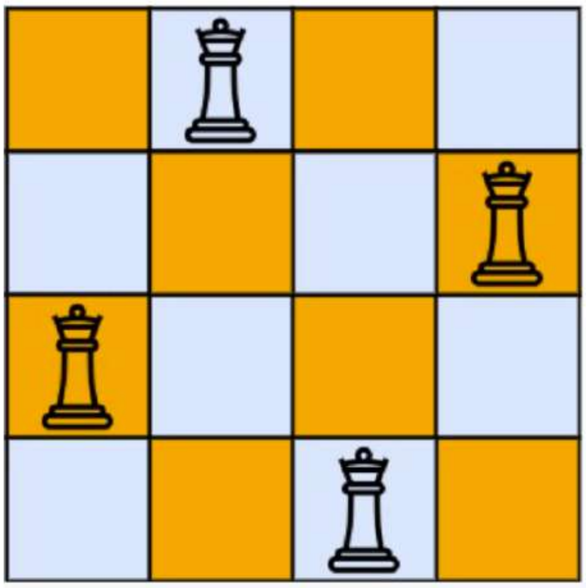
Hard 6383 159 Add to List Share

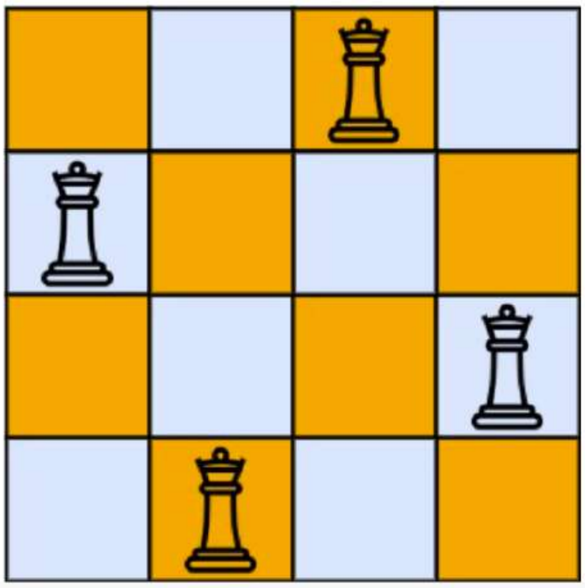
The **n-queens** puzzle is the problem of placing `n` queens on an `n x n` chessboard such that no two queens attack each other.

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:

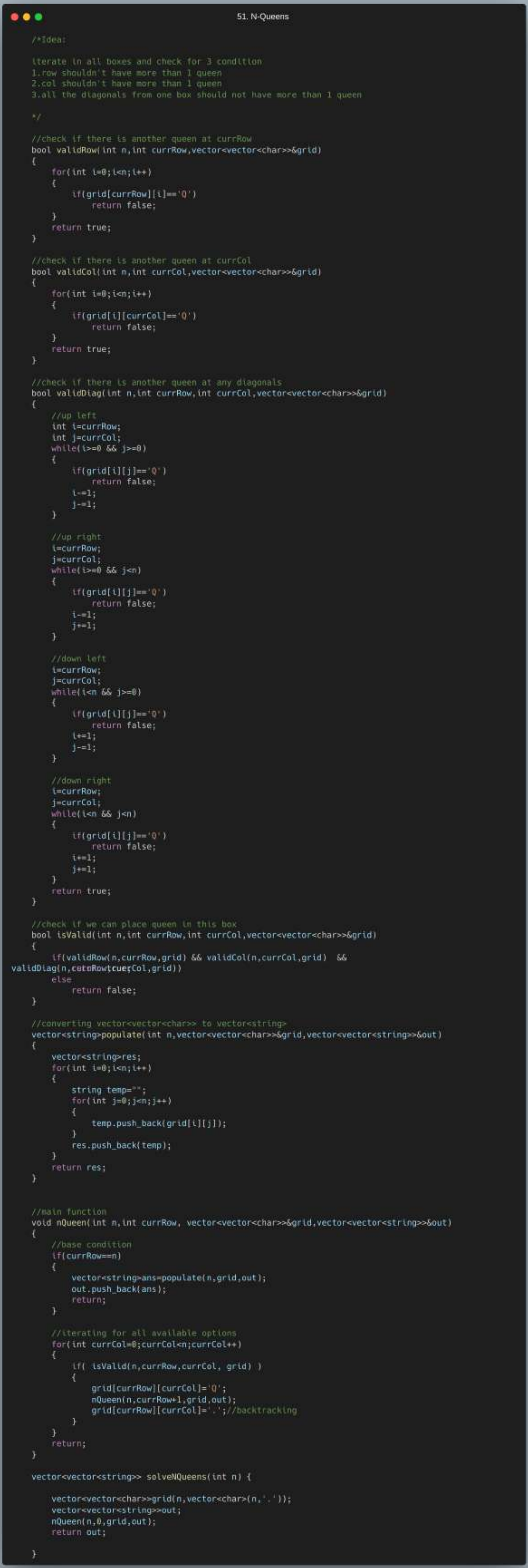




```
Input: n = 4
Output: [[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]
Explanation: There exist two distinct solutions to the 4-queens puzzle as shown above
```

Example 2:

```
Input: n = 1
Output: [["Q"]]
```


51. N-Queens

```cpp
    /*Idea:

    iterate in all boxes and check for 3 condition
    1.row shouldn't have more than 1 queen
    2.col shouldn't have more than 1 queen
    3.all the diagonals from one box should not have more than 1 queen

    */

    //check if there is another queen at currRow
    bool validRow(int n,int currRow,vector<vector<char>>&grid)
    {
        for(int i=0;i<n;i++)
        {
            if(grid[currRow][i]=='Q')
                return false;
        }
        return true;
    }

    //check if there is another queen at currCol
    bool validCol(int n,int currCol,vector<vector<char>>&grid)
    {
        for(int i=0;i<n;i++)
        {
            if(grid[i][currCol]=='Q')
                return false;
        }
        return true;
    }

    //check if there is another queen at any diagonals
    bool validDiag(int n,int currRow,int currCol,vector<vector<char>>&grid)
    {
        //up left
        int i=currRow;
        int j=currCol;
        while(i>=0 && j>=0)
        {
            if(grid[i][j]=='Q')
                return false;
            i-=1;
            j-=1;
        }

        //up right
        i=currRow;
        j=currCol;
        while(i>=0 && j<n)
        {
            if(grid[i][j]=='Q')
                return false;
            i-=1;
            j+=1;
        }

        //down left
        i=currRow;
        j=currCol;
        while(i<n && j>=0)
        {
            if(grid[i][j]=='Q')
                return false;
            i+=1;
            j-=1;
        }

        //down right
        i=currRow;
        j=currCol;
        while(i<n && j<n)
        {
            if(grid[i][j]=='Q')
                return false;
            i+=1;
            j+=1;
        }
        return true;
    }

    //check if we can place queen in this box
    bool isValid(int n,int currRow,int currCol,vector<vector<char>>&grid)
    {
        if(validRow(n,currRow,grid) && validCol(n,currCol,grid)  &&
validDiag(n,cetoRow;cuerCol,grid))
        else
            return false;
    }

    //converting vector<vector<char>> to vector<string>
    vector<string>populate(int n,vector<vector<char>>&grid,vector<vector<string>>&out)
    {
        vector<string>res;
        for(int i=0;i<n;i++)
        {
            string temp="";
            for(int j=0;j<n;j++)
            {
                temp.push_back(grid[i][j]);
            }
            res.push_back(temp);
        }
        return res;
    }


    //main function
    void nQueen(int n,int currRow, vector<vector<char>>&grid,vector<vector<string>>&out)
    {
        //base condition
        if(currRow==n)
        {
            vector<string>ans=populate(n,grid,out);
            out.push_back(ans);
            return;
        }

        //iterating for all available options
        for(int currCol=0;currCol<n;currCol++)
        {
            if( isValid(n,currRow,currCol, grid) )
            {
                grid[currRow][currCol]='Q';
                nQueen(n,currRow+1,grid,out);
                grid[currRow][currCol]='.';//backtracking
            }
        }
        return;
    }

    vector<vector<string>> solveNQueens(int n) {

        vector<vector<char>>grid(n,vector<char>(n,'.'));
        vector<vector<string>>out;
        nQueen(n,0,grid,out);
        return out;

    }
```