N-Queen Problem

51. N-Queens

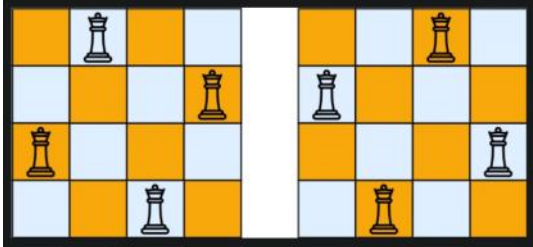
Hard Topics Companies

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.

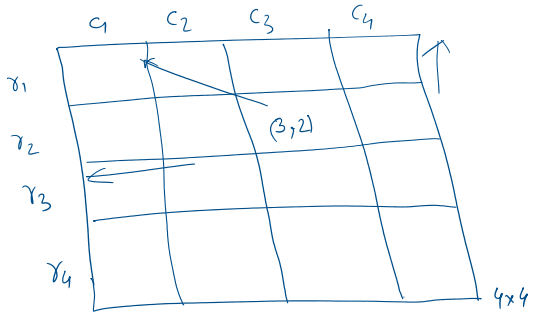
Example 1:



```
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"]]
```



① go row by row

② Try to place the queen at each cell of the row

③ Validate the queen placement
- ① if valid (move to next row)
  save the current configuration & move to next row (recursion
- ② remove the queen & continue its rating along same row

④ Base Case :-

if row == n;

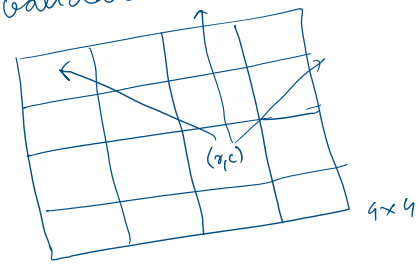
a) save current configuration in ans

b) return;

isValid function :-

Now to validate the current chess configuration

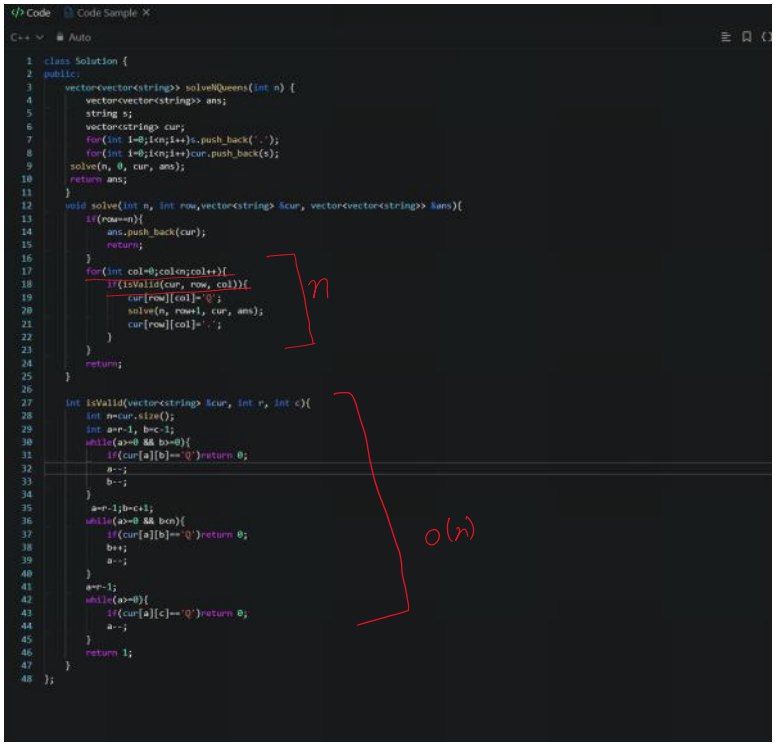
eg: in



check availability of a queen along 3 direction :-

1. ← diagonal 2. ↗ diagonal 3. ↑ col.

There is no need to check availability at same row as each row can have a single queen.



```cpp
class Solution {
public:
    vector<vector<string>> solveNQueens(int n) {
        vector<vector<string>> ans;
        string s;
        vector<string> cur;
        for(int i=0;i<n;i++)s.push_back('.');
        for(int i=0;i<n;i++)cur.push_back(s);
     solve(n, 0, cur, ans);
     return ans;
    }
    void solve(int n, int row,vector<string> &cur, vector<vector<string>> &ans){
        if(row==n){
            ans.push_back(cur);
            return;
        }
        for(int col=0;col<n;col++){
            if(isValid(cur, row, col)){
                cur[row][col]='Q';
                solve(n, row+1, cur, ans);
                cur[row][col]='.';
            }
        }
        return;
    }

    int isValid(vector<string> &cur, int r, int c){
        int n=cur.size();
        int a=r-1, b=c-1;
        while(a>=0 && b>=0){
            if(cur[a][b]=='Q')return 0;
            a--;
            b--;
        }
        a=r-1;b=c+1;
        while(a>=0 && b<n){
            if(cur[a][b]=='Q')return 0;
            b++;
            a--;
        }
        a=r-1;
        while(a>=0){
            if(cur[a][c]=='Q')return 0;
            a--;
        }
        return 1;
    }
};
```

n

o(n)

T.C. Analysis :-

$O(n!)$

S.C.

$O(n)$ ← recursion stack