N Queens

BRUTE FORCE :

We consider choosing col one by one and looping through all rows to check we can place a queen at that row and col. If it is safe we consider it otherwise we move to next row.

IsSafe function checks whether a queen is placed in that row, upper diag before that col and low diag before that col. If this function returns true which means it is safe to place queen we place it otherwise we do not make a recursive call further for that chess board.

```
bool isSafe(int row,int col,vector<vector<int>> temp,int n)
{
    int rowSave=row;
    int colSave=col;
    while(row>=0 && col>=0)
    {
        if(temp[row][col]==1)
        {
            return false;
        }
        row--;
        col--;
    }
    row=rowSave;
    col=colSave;
    while(col>=0)
    {
        if(temp[row][col]==1)
        {
            return false;
        }
        col--;
    }
    row=rowSave;
    col=colSave;
    while(col>=0&& row<n)
    {
        if(temp[row][col]==1)
        {
            return false;
        }
        row++;
        col--;

    }
    return true;
}
void helper(int col,int n,vector<vector<int>> temp,vector<vector<int>>& ans)
{
    if(col==n)
    {
        vector<int> res;
        for(int i=0;i<temp.size();i++)
        {
            for(int j=0;j<temp[0].size();j++)
            {
                res.push_back(temp[i][j]);
            }
        }
        ans.push_back(res);
        return;
    }
    for(int row=0;row<n;row++)
    {
      if (isSafe(row, col, temp, n)) {
        temp[row][col] = 1;
        helper(col+1,n,temp,ans );
        temp[row][col]=0;
      }
    }
}
```

```
vector<vector<int>> solveNQueens(int n) {
    // Write your code here.
    vector<vector<int>> temp(n,vector<int>(n,0));
    vector<vector<int>> ans;
    helper(0,n,temp,ans);
    return ans;
}
```

- Time Complexity : O(N!*N)
- Space Complexity : O(N*N)

Optimal Approach :

Replacing isSafe function by hashig.

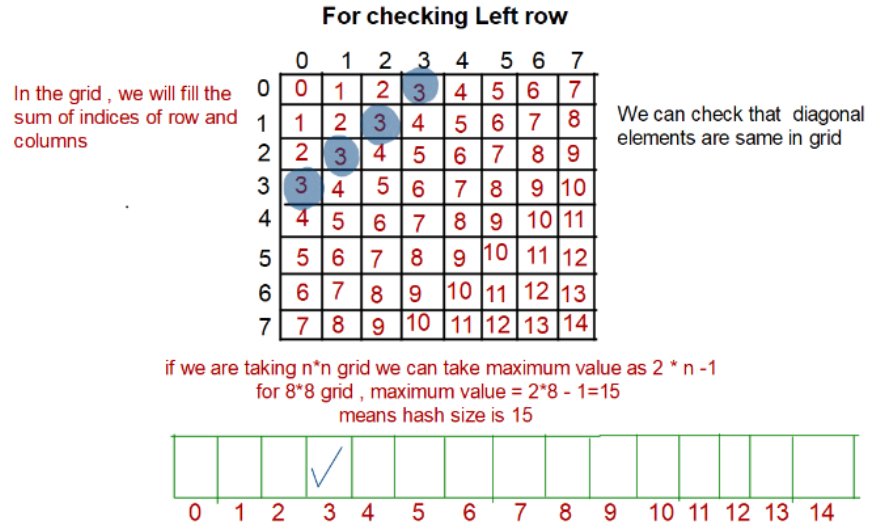
We maintain 3 vector one for row check ,another for lowDiag and other for upDiag.

for rowCheck a vector of length n is maintained.

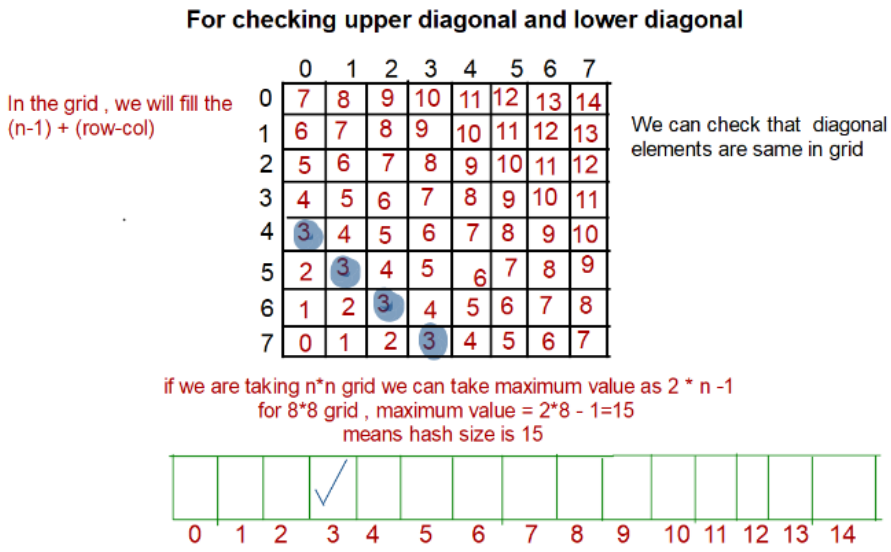
for lowDiag we observe a pattern that alllow diag elems are summation o row+col index.

for upDiag we use formula (n-1)+(col-row) .

lowDiag hash :



upDiag hash :




```
void helper(int col,int n,vector<vector<int>> temp,vector<int> rowHash,vector<int> lowDiag,vector<int> upDiag,vector<vector<int>>& ans)
{
    if(col==n)
    {
        vector<int> res;
        for(int i=0;i<temp.size();i++)
        {
            for(int j=0;j<temp[0].size();j++)
            {
                res.push_back(temp[i][j]);
            }
        }
        ans.push_back(res);
        return;
    }
    for(int row=0;row<n;row++)
    {
      if (rowHash[row]==0&& lowDiag[row+col]==0 && upDiag[(n-1)+(col-row)]==0) {
        rowHash[row]=1;
        lowDiag[row+col]=1;
        upDiag[(n-1)+(col-row)]=1;
        temp[row][col] = 1;
        helper(col+1,n,temp,rowHash,lowDiag,upDiag,ans );
        temp[row][col]=0;
        rowHash[row]=0;
        lowDiag[row+col]=0;
        upDiag[(n-1)+(col-row)]=0;
      }
    }
}

vector<vector<int>> solveNQueens(int n) {
    // Write your code here.
    vector<int> rowHash(n,0),upDiag(2*n-1,0),lowDiag(2*n-1,0);
    vector<vector<int>> temp(n,vector<int>(n,0));
    vector<vector<int>> ans;
    helper(0,n,temp,rowHash,upDiag,lowDiag,ans);
    return ans;
}
```

- Time Complexity : O(N!*N)
- Space Complexity : O(N)