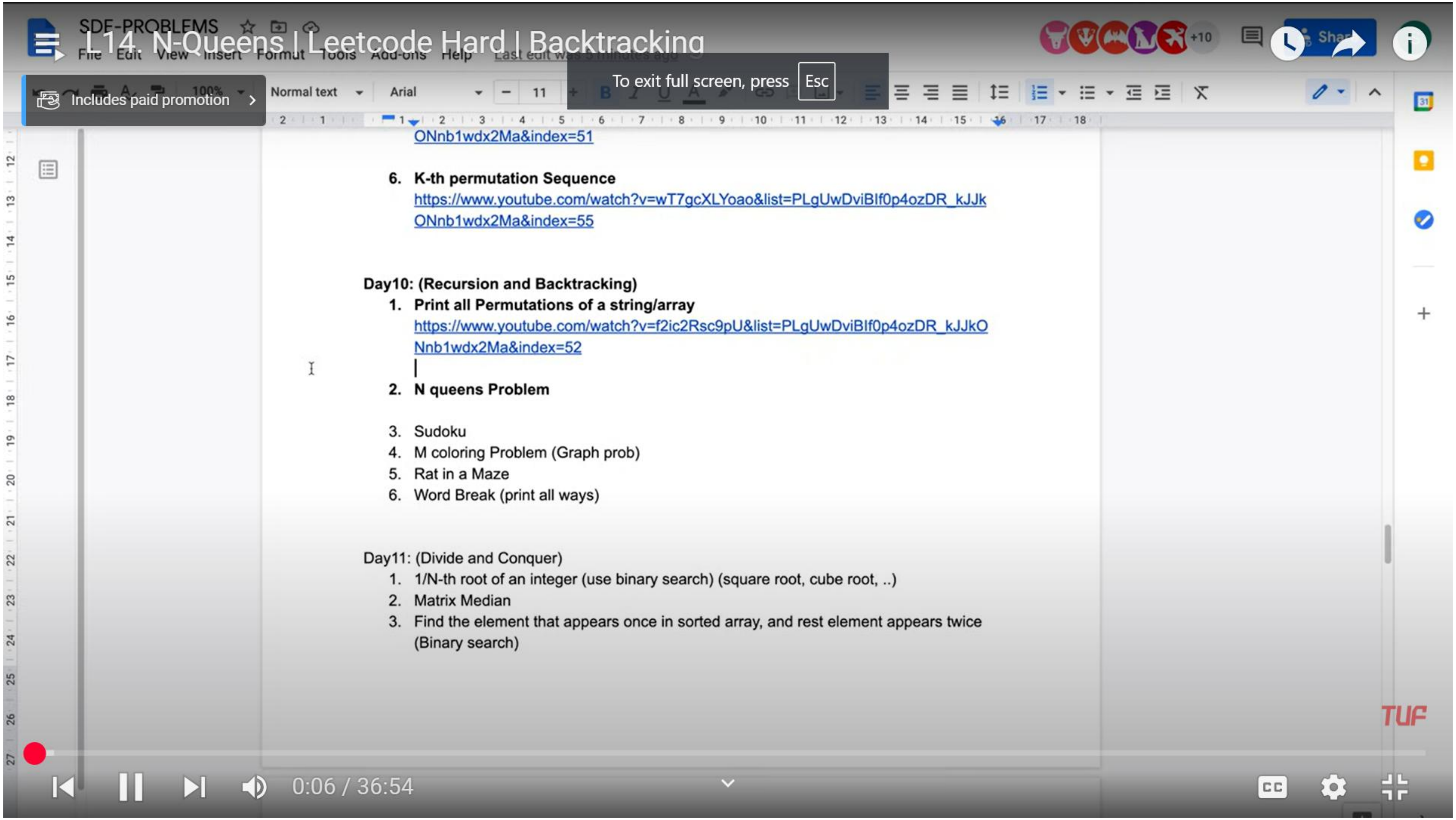

SDE-PROBLEMS
File Edit View Insert Format Tools Add-ons Help
L14. N-Queens | Leetcode Hard | Backtracking
Includes paid promotion
To exit full screen, press Esc
Normal text
Arial
11
0:06 / 36:54
TUF

ONnb1wdx2Ma&index=51

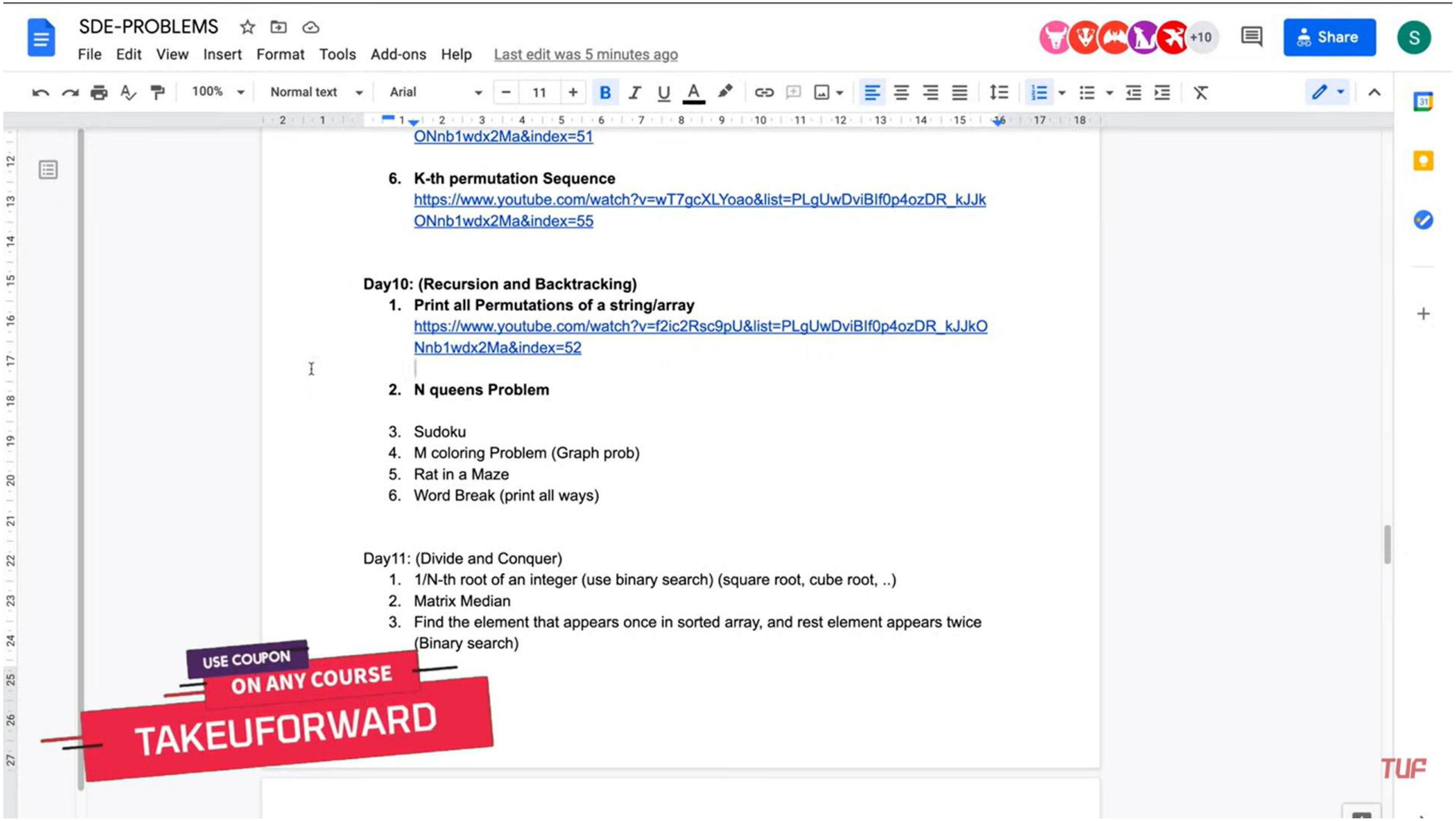
6. **K-th permutation Sequence**
   https://www.youtube.com/watch?v=wT7gcXLYoao&list=PLgUwDviBIf0p4ozDR_kJJkONnb1wdx2Ma&index=55

Day10: (Recursion and Backtracking)

1. **Print all Permutations of a string/array**
   https://www.youtube.com/watch?v=f2ic2Rsc9pU&list=PLgUwDviBIf0p4ozDR_kJJkONnb1wdx2Ma&index=52
2. **N queens Problem**
3. Sudoku
4. M coloring Problem (Graph prob)
5. Rat in a Maze
6. Word Break (print all ways)

Day11: (Divide and Conquer)

1. 1/N-th root of an integer (use binary search) (square root, cube root, ..)
2. Matrix Median
3. Find the element that appears once in sorted array, and rest element appears twice (Binary search)


SDE-PROBLEMS
File Edit View Insert Format Tools Add-ons Help Last edit was 5 minutes ago
Share
100% Normal text Arial 11

ONnb1wdx2Ma&index=51

6. **K-th permutation Sequence**
   https://www.youtube.com/watch?v=wT7gcXLYoao&list=PLgUwDviBIf0p4ozDR_kJJkONnb1wdx2Ma&index=55

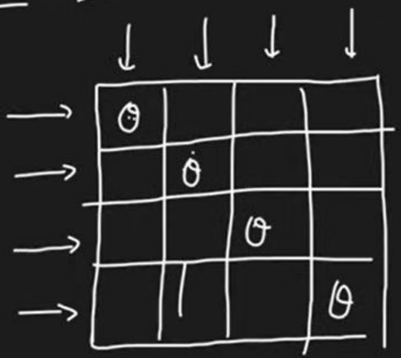
Day10: (Recursion and Backtracking)

1. **Print all Permutations of a string/array**
   https://www.youtube.com/watch?v=f2ic2Rsc9pU&list=PLgUwDviBIf0p4ozDR_kJJkONnb1wdx2Ma&index=52
2. **N queens Problem**
3. Sudoku
4. M coloring Problem (Graph prob)
5. Rat in a Maze
6. Word Break (print all ways)

Day11: (Divide and Conquer)

1. 1/N-th root of an integer (use binary search) (square root, cube root, ..)
2. Matrix Median
3. Find the element that appears once in sorted array, and rest element appears twice (Binary search)

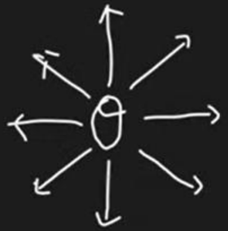
N = 4 Queens



→ Every row → 1 Q

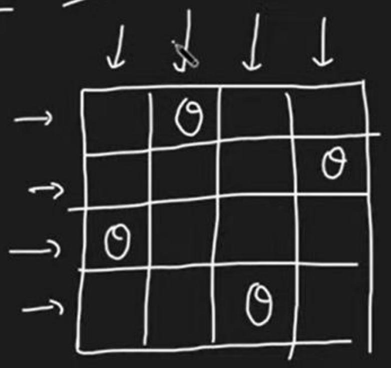
→ Every col → 1 Q

→ None of the Q attack each other



TUF

N = 4 Queens

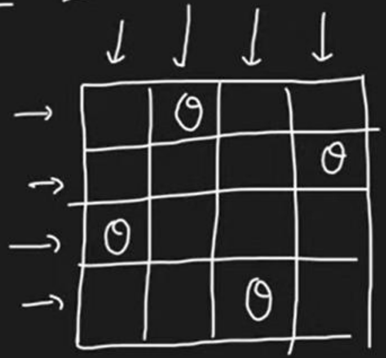


→ Every row → 1 Q

→ Every col → 1 Q

→ None of the Q attack each other

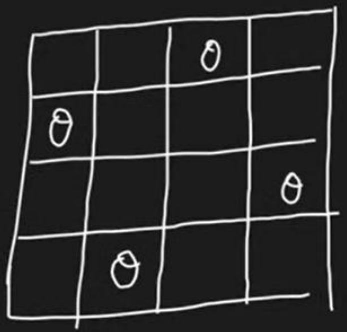
N = 4 Queens



→ Every row → 1 Q

→ Every col → 1 Q

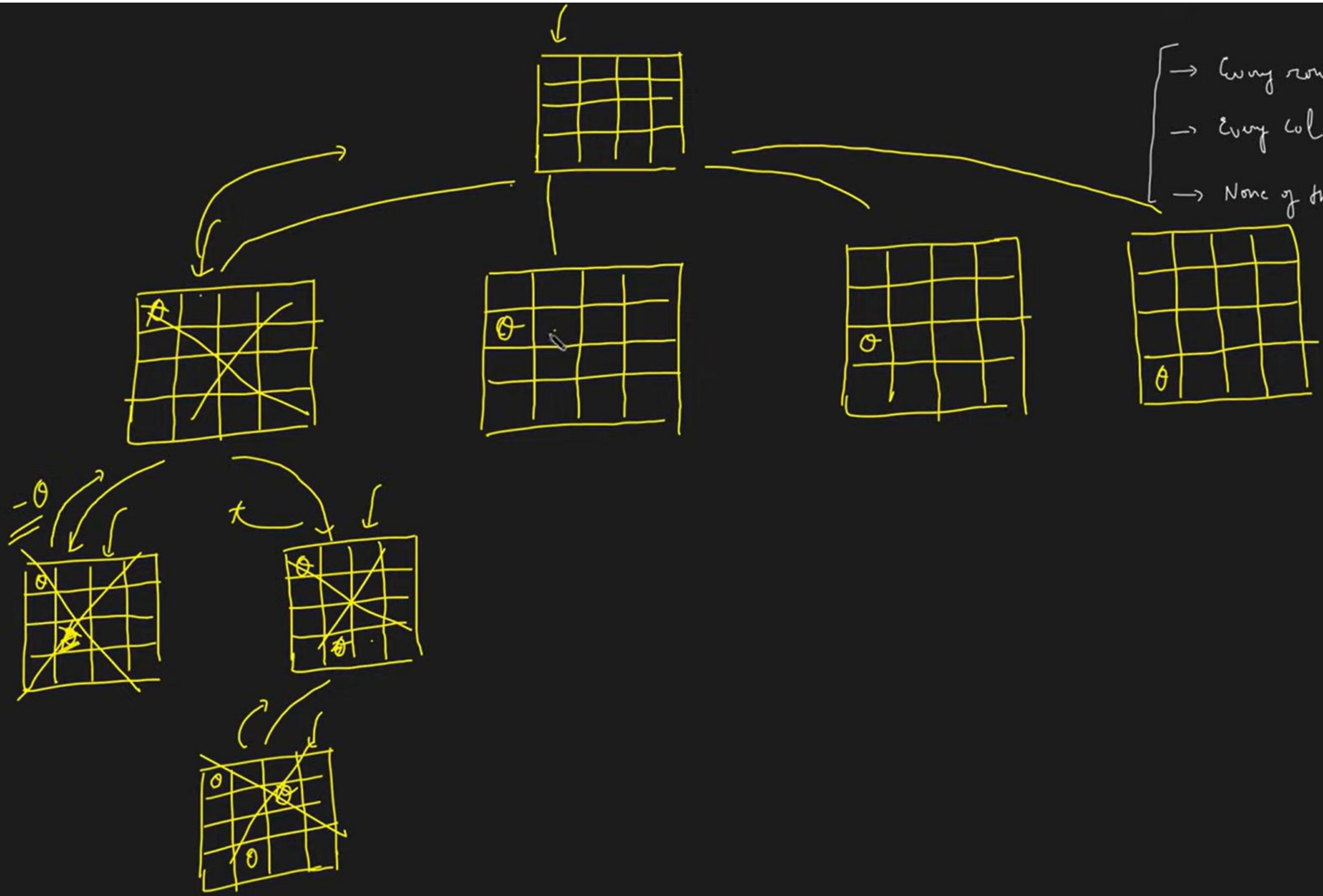
→ None of the Q attack each other

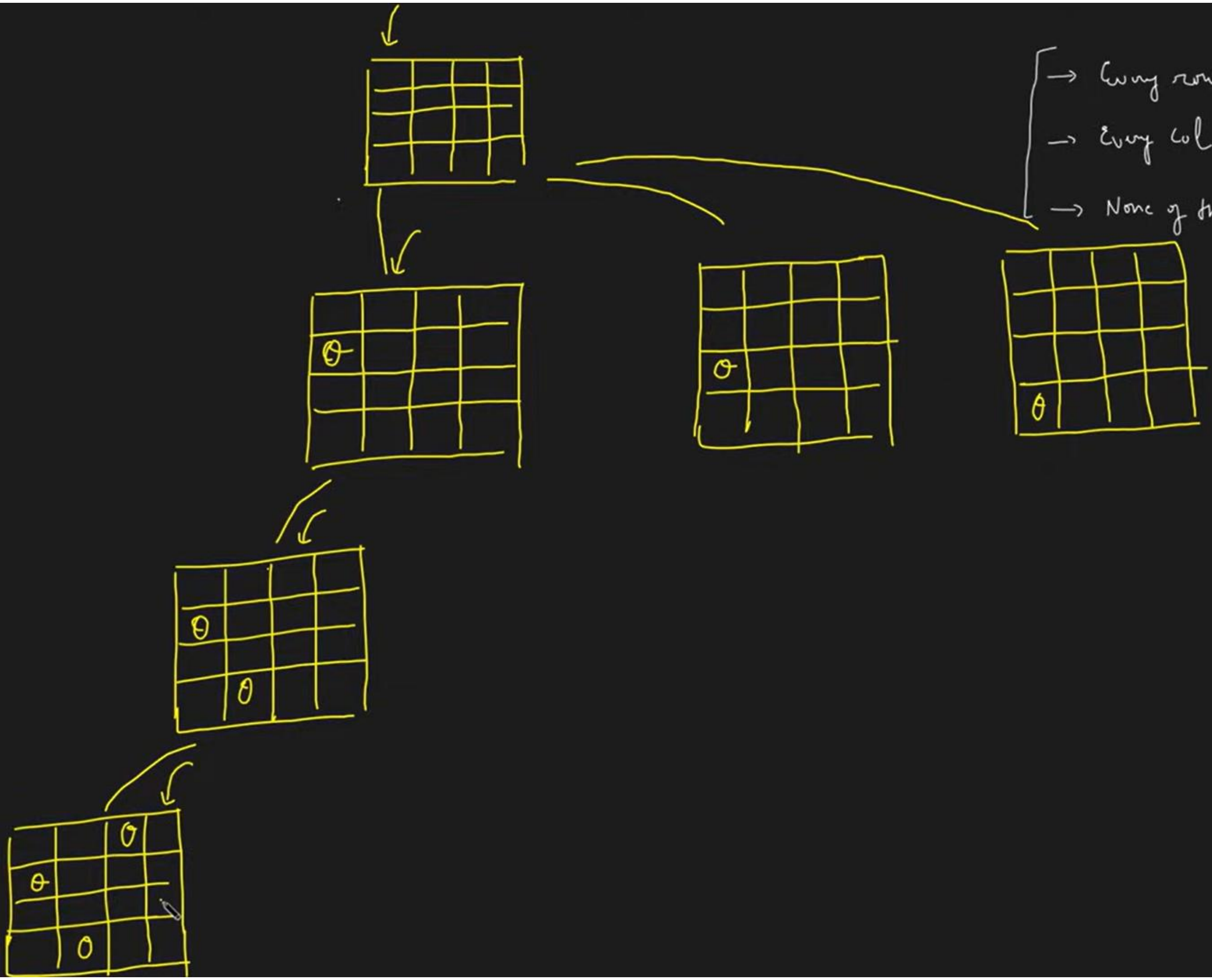


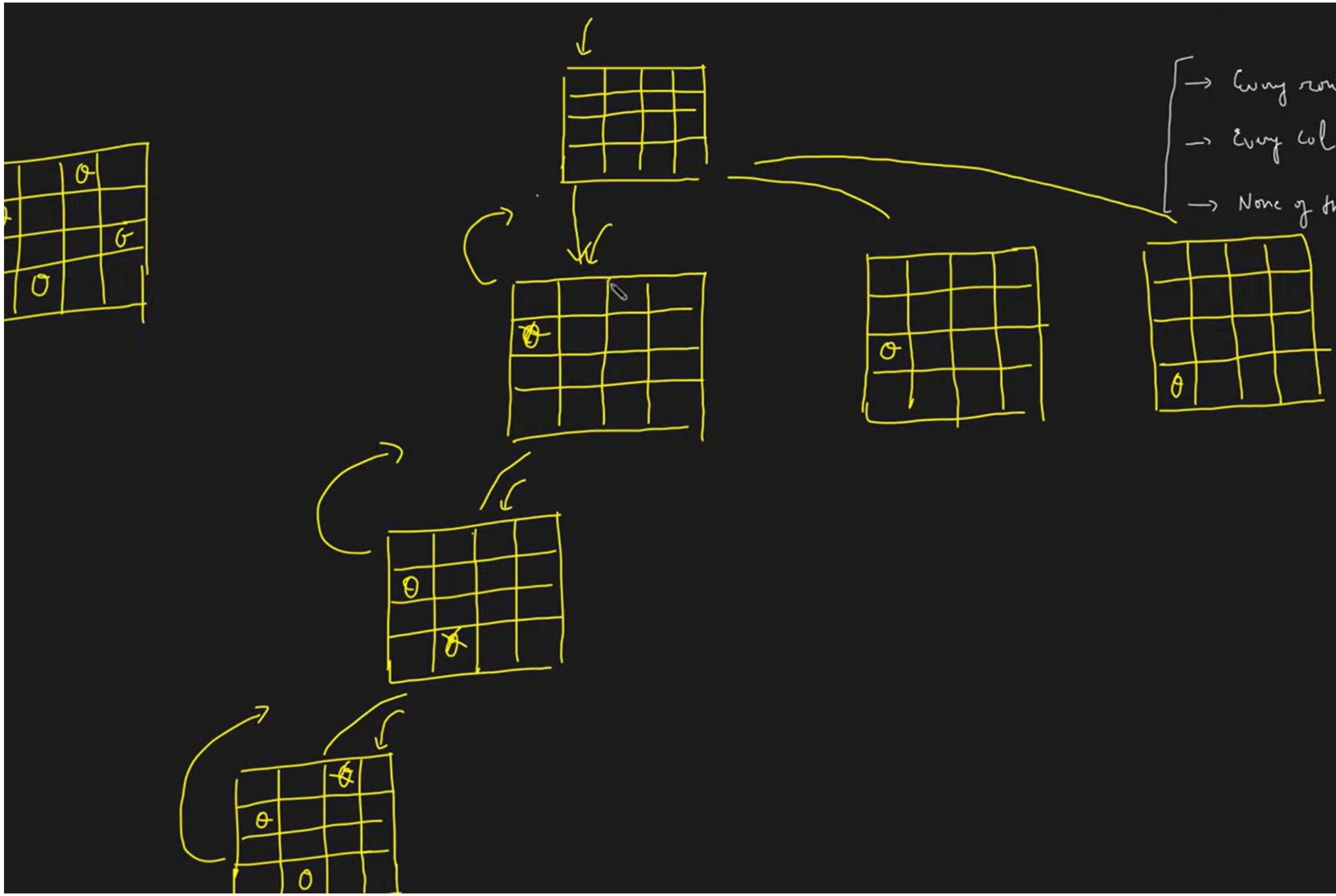
N.

N × N.

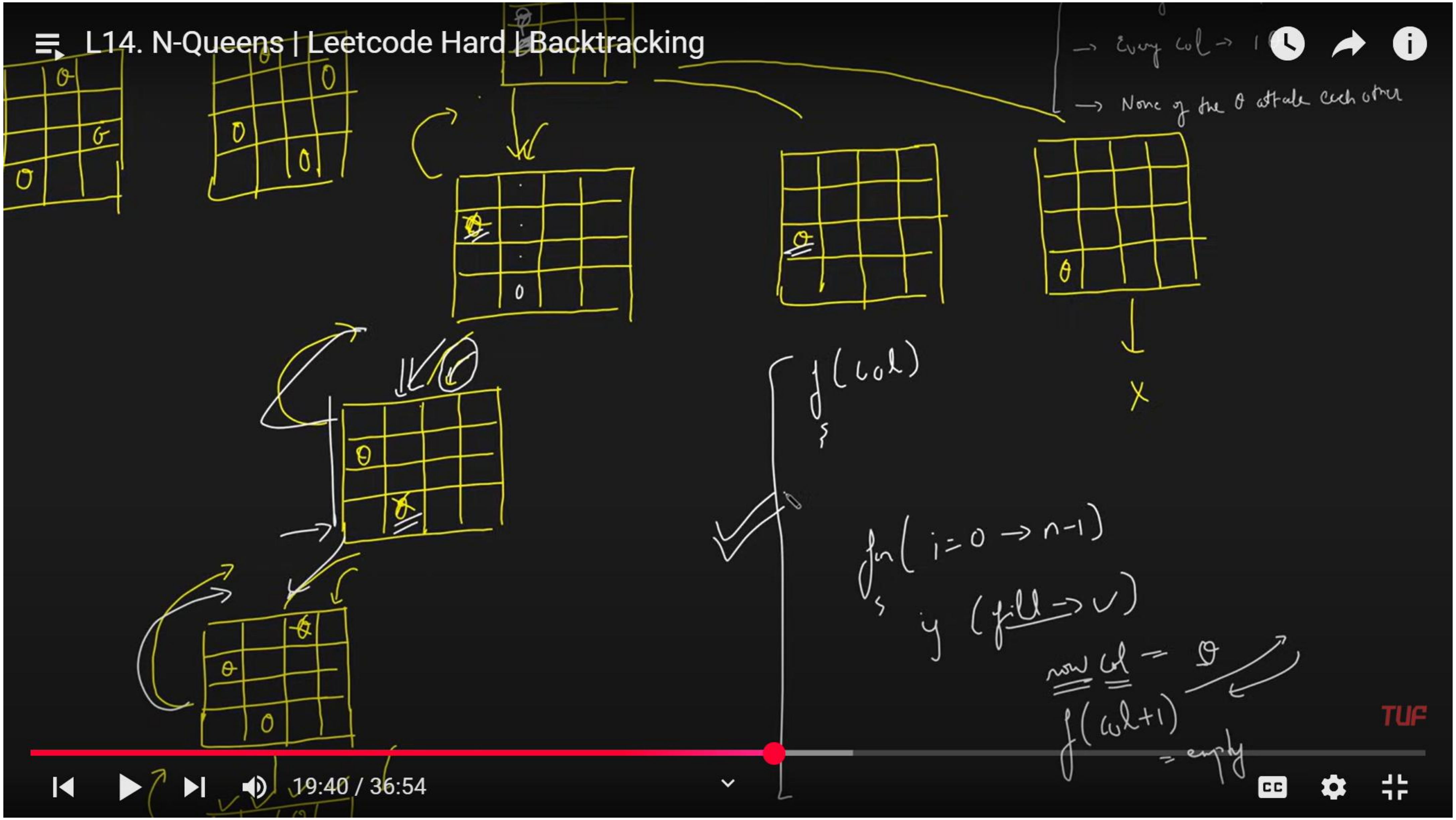
- → Every row → 1 Q
- → Every col → 1 Q
- → None of the Q attack each other




→ Every row → 1 Q
→ Every col → 1 Q
→ None of the Q attack each other
Q
Q
Q
Q
Q
Q
Q
Q



- → Every row → 1 Q
- → Every col → 1 Q
- → None of the Q attack each other



L14. N-Queens | Leetcode Hard | Backtracking

→ Every col → 1

→ None of the Q attack each other

f(col)
{

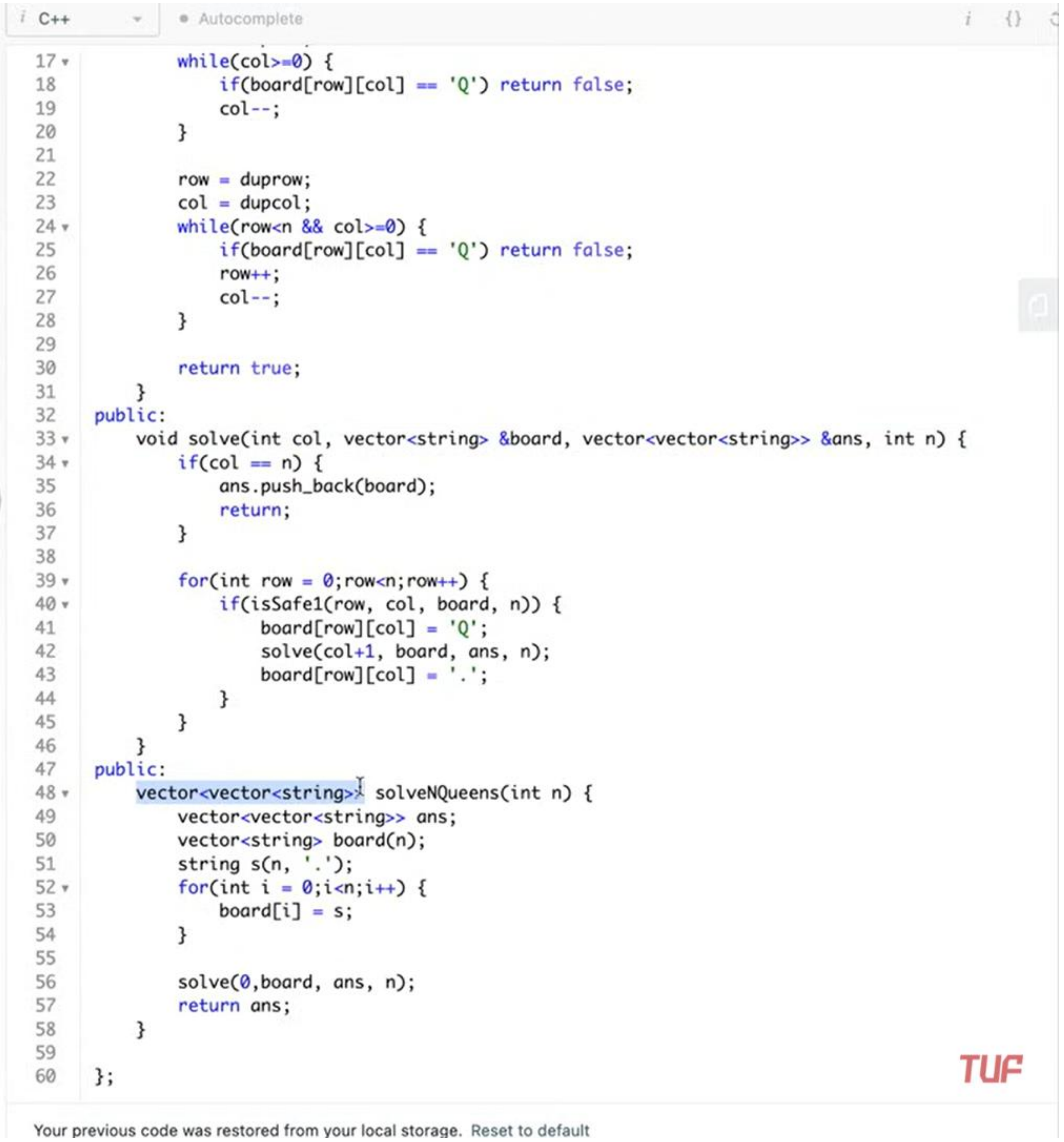
for(i=0 → n-1)
{
 if (fill → v)

 row col = Q

 f(col+1)

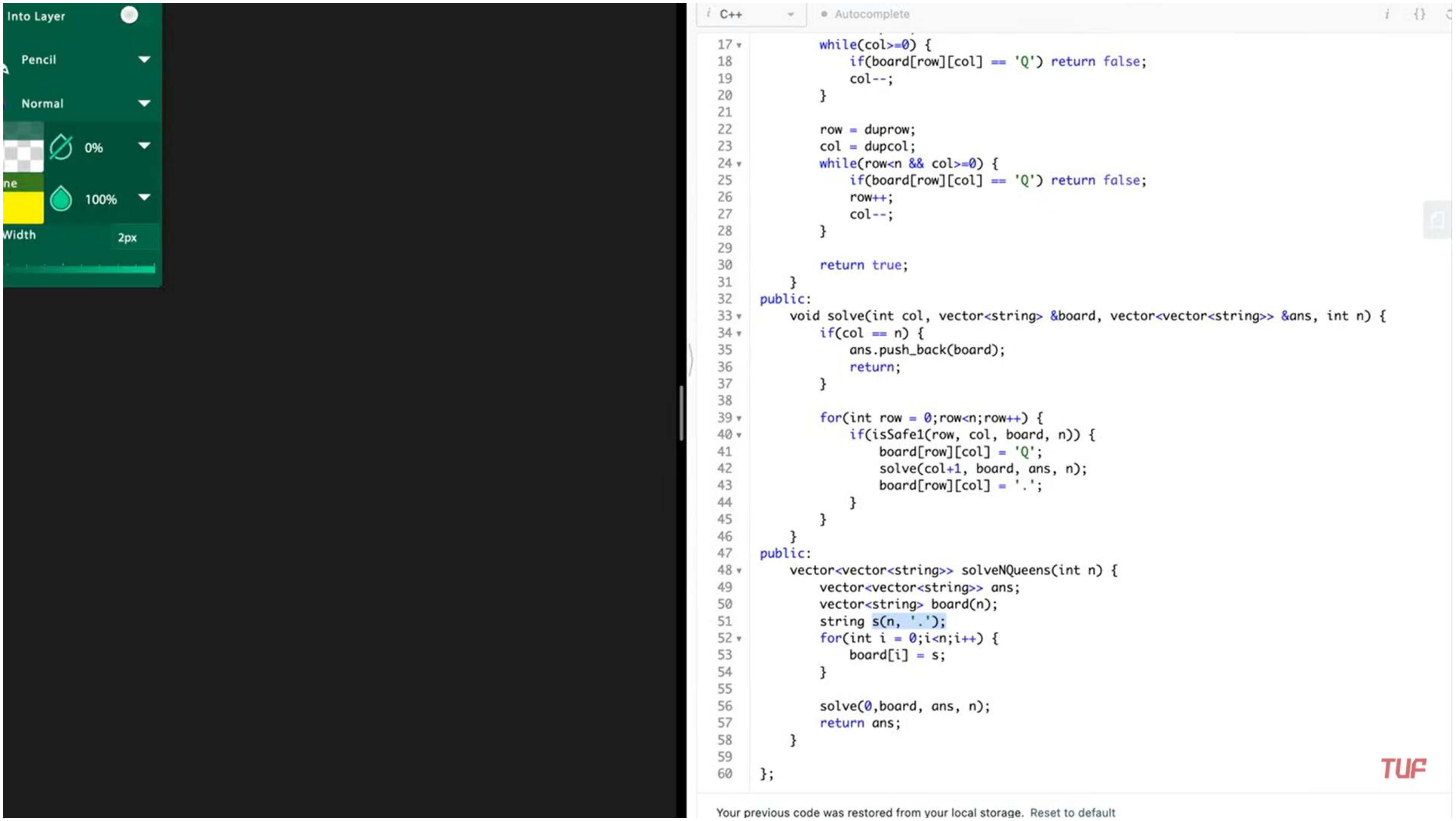
 = empty

19:40 / 36:54



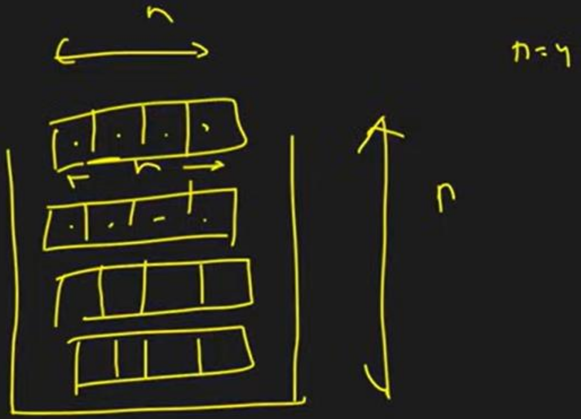
```cpp
        while(col>=0) {
            if(board[row][col] == 'Q') return false;
            col--;
        }

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

        return true;
    }
public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }

        for(int row = 0;row<n;row++) {
            if(isSafe1(row, col, board, n)) {
                board[row][col] = 'Q';
                solve(col+1, board, ans, n);
                board[row][col] = '.';
            }
        }
    }
public:
    vector<vector<string>> solveNQueens(int n) {
        vector<vector<string>> ans;
        vector<string> board(n);
        string s(n, '.');
        for(int i = 0;i<n;i++) {
            board[i] = s;
        }

        solve(0,board, ans, n);
        return ans;
    }

};
```

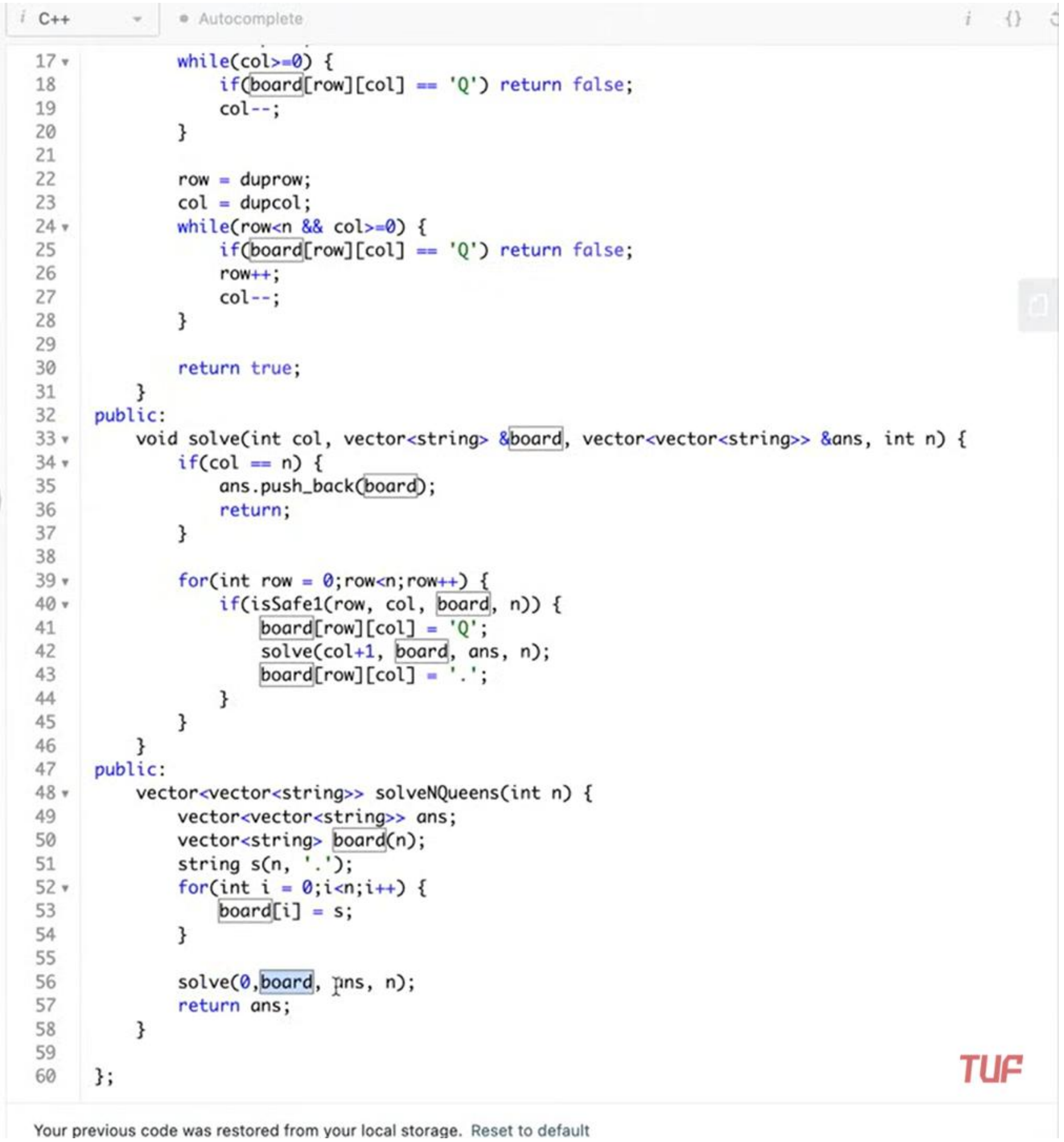
Your previous code was restored from your local storage. Reset to default



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

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

        return true;
    }
public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }

        for(int row = 0;row<n;row++) {
            if(isSafe1(row, col, board, n)) {
                board[row][col] = 'Q';
                solve(col+1, board, ans, n);
                board[row][col] = '.';
            }
        }
    }
public:
    vector<vector<string>> solveNQueens(int n) {
        vector<vector<string>> ans;
        vector<string> board(n);
        string s(n, '.');
        for(int i = 0;i<n;i++) {
            board[i] = s;
        }

        solve(0,board, ans, n);
        return ans;
    }

};
```

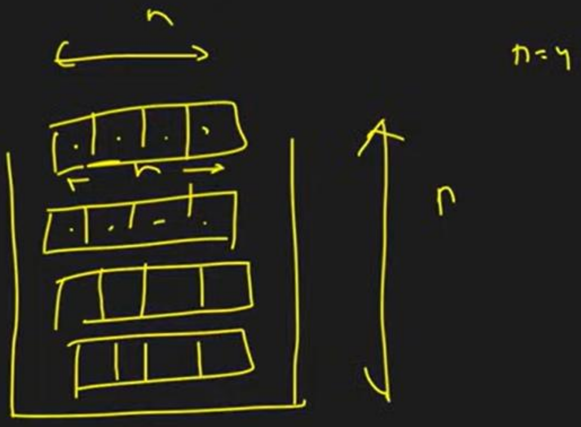
Your previous code was restored from your local storage. Reset to default

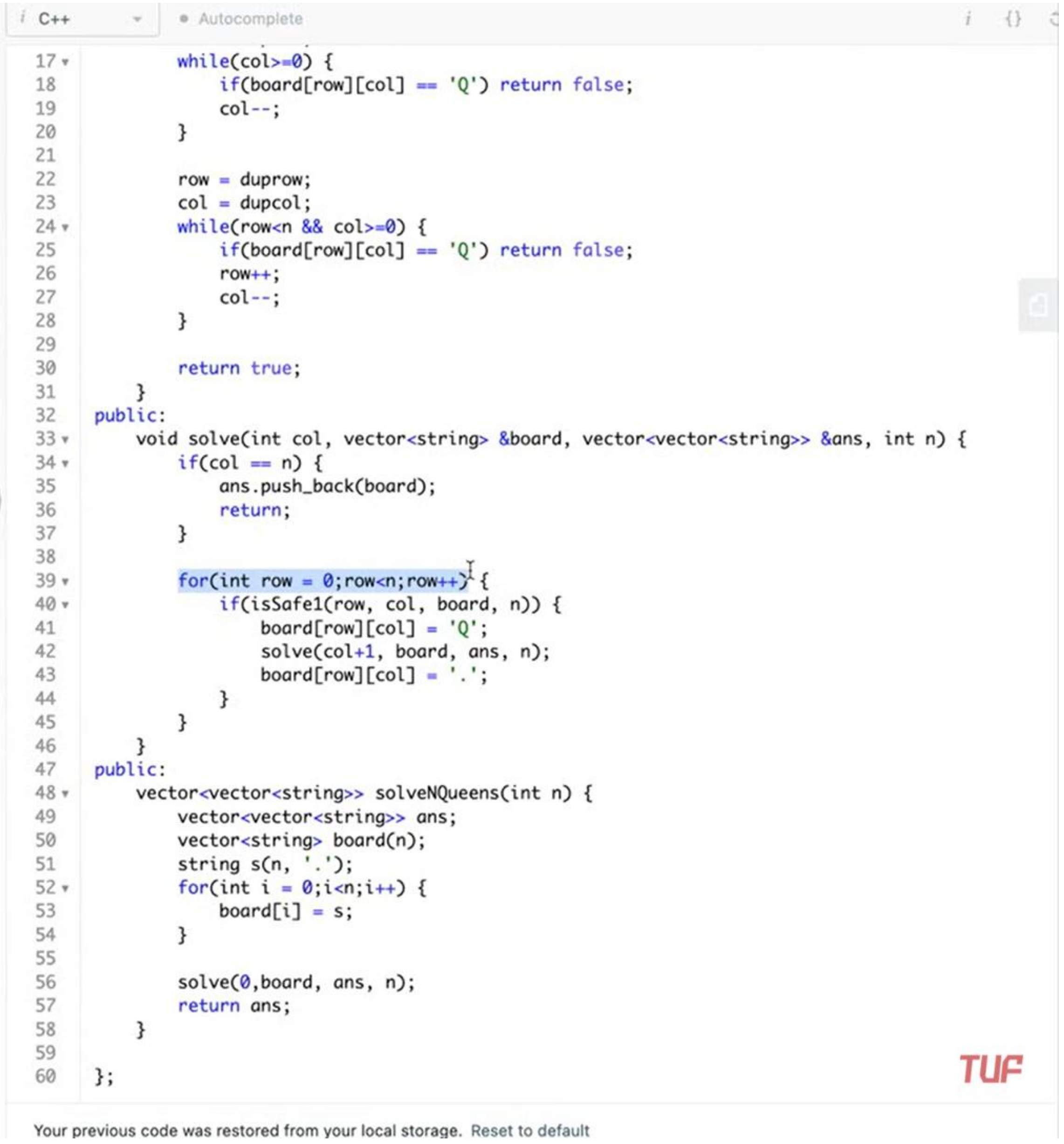




```
        while(col>=0) {
            if(board[row][col] == 'Q') return false;
            col--;
        }

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

        return true;
    }
public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }

        for(int row = 0;row<n;row++) {
            if(isSafe1(row, col, board, n)) {
                board[row][col] = 'Q';
                solve(col+1, board, ans, n);
                board[row][col] = '.';
            }
        }
    }
public:
    vector<vector<string>> solveNQueens(int n) {
        vector<vector<string>> ans;
        vector<string> board(n);
        string s(n, '.');
        for(int i = 0;i<n;i++) {
            board[i] = s;
        }

        solve(0,board, ans, n);
        return ans;
    }

};
```

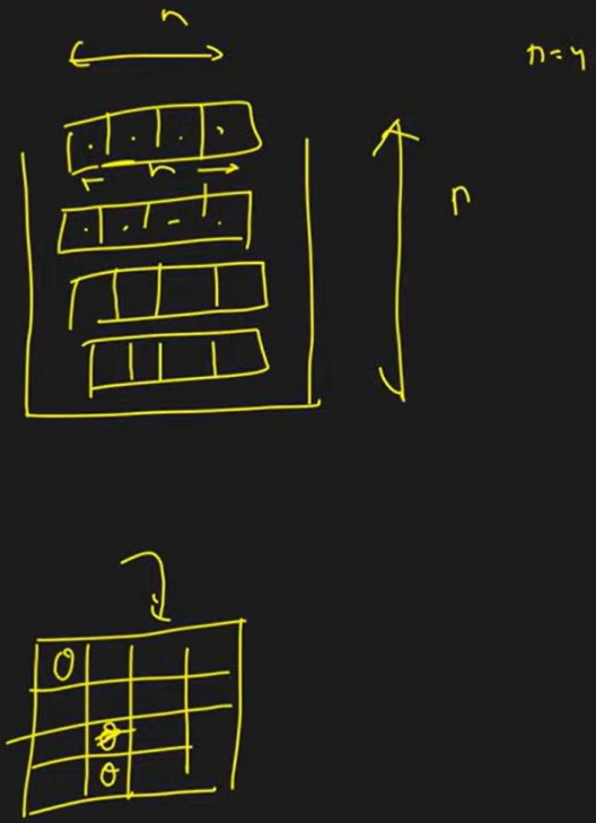
Your previous code was restored from your local storage. Reset to default

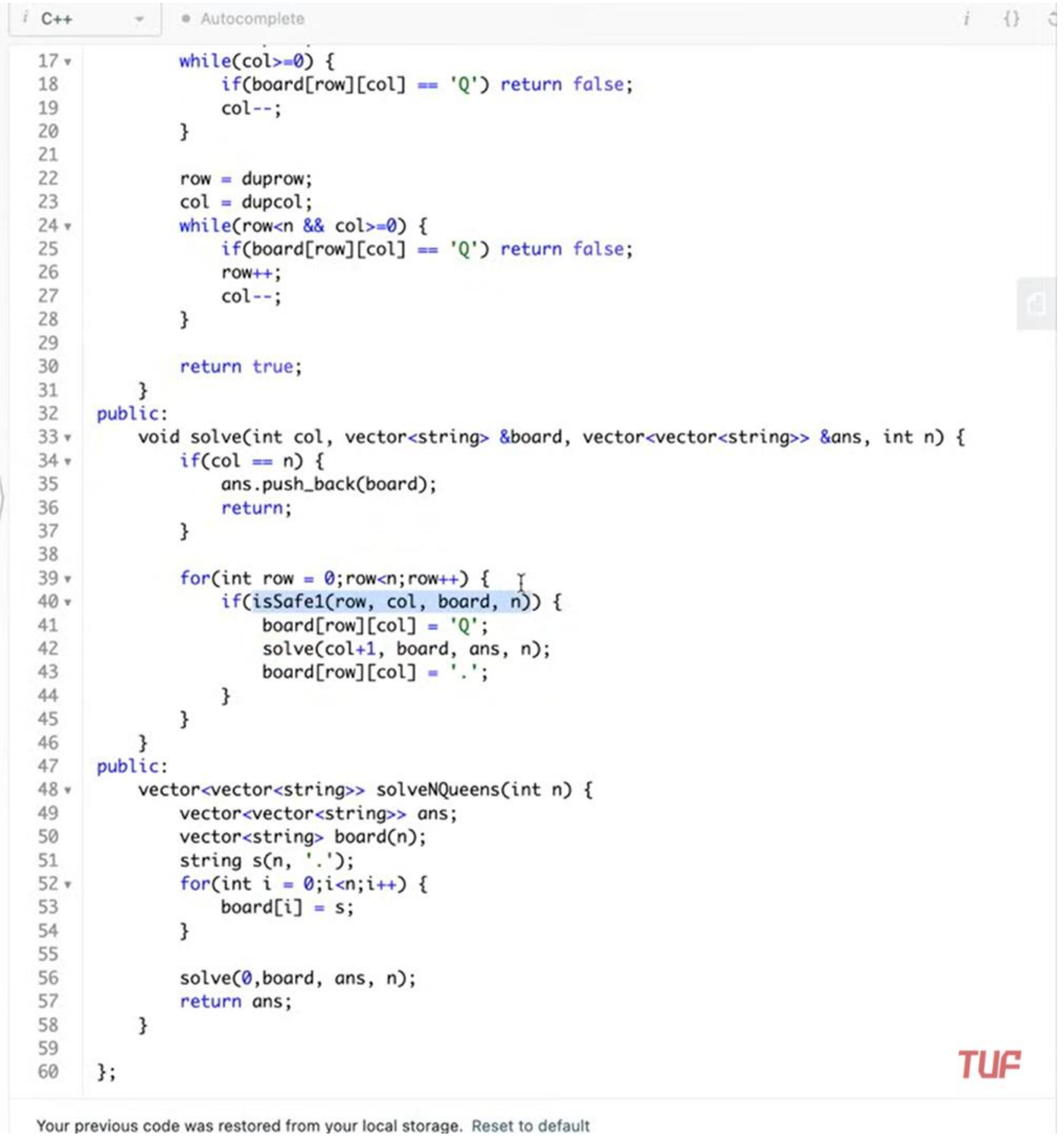




```cpp
        while(col>=0) {
            if(board[row][col] == 'Q') return false;
            col--;
        }

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

        return true;
    }
public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }

        for(int row = 0;row<n;row++) {
            if(isSafe1(row, col, board, n)) {
                board[row][col] = 'Q';
                solve(col+1, board, ans, n);
                board[row][col] = '.';
            }
        }
    }
public:
    vector<vector<string>> solveNQueens(int n) {
        vector<vector<string>> ans;
        vector<string> board(n);
        string s(n, '.');
        for(int i = 0;i<n;i++) {
            board[i] = s;
        }

        solve(0,board, ans, n);
        return ans;
    }

};
```

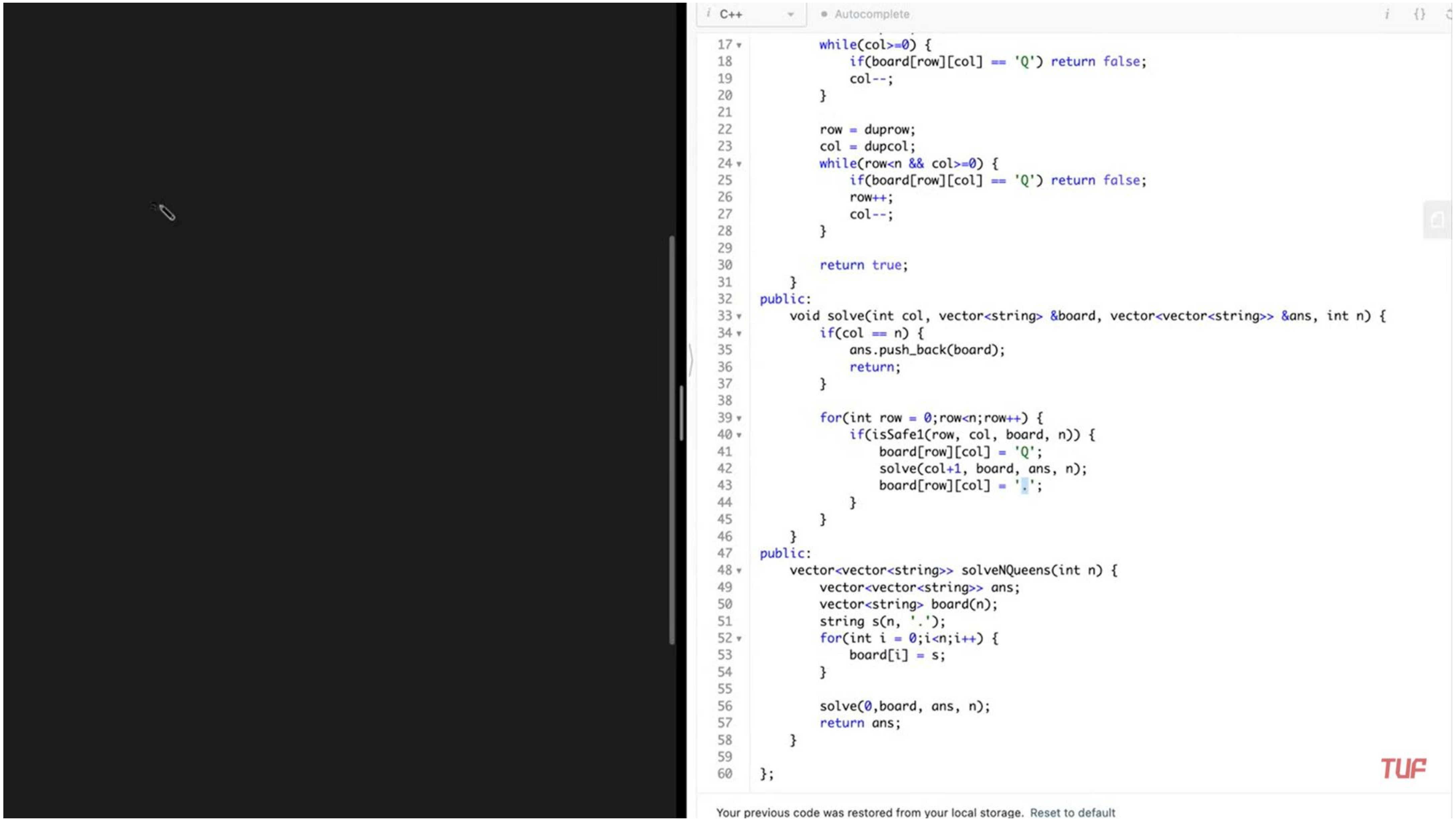
Your previous code was restored from your local storage. Reset to default





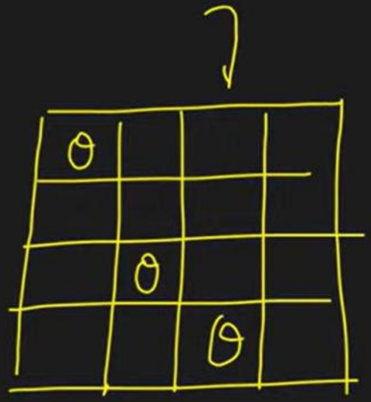
```
        while(col>=0) {
            if(board[row][col] == 'Q') return false;
            col--;
        }

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

        return true;
    }
public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }

        for(int row = 0;row<n;row++) {
            if(isSafe1(row, col, board, n)) {
                board[row][col] = 'Q';
                solve(col+1, board, ans, n);
                board[row][col] = '.';
            }
        }
    }
public:
    vector<vector<string>> solveNQueens(int n) {
        vector<vector<string>> ans;
        vector<string> board(n);
        string s(n, '.');
        for(int i = 0;i<n;i++) {
            board[i] = s;
        }

        solve(0,board, ans, n);
        return ans;
    }

};
```

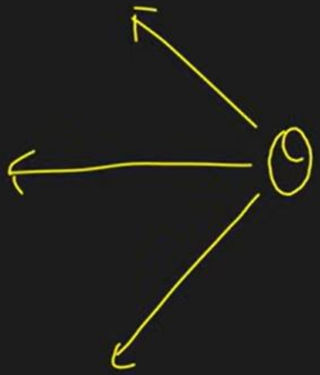
Your previous code was restored from your local storage. Reset to default

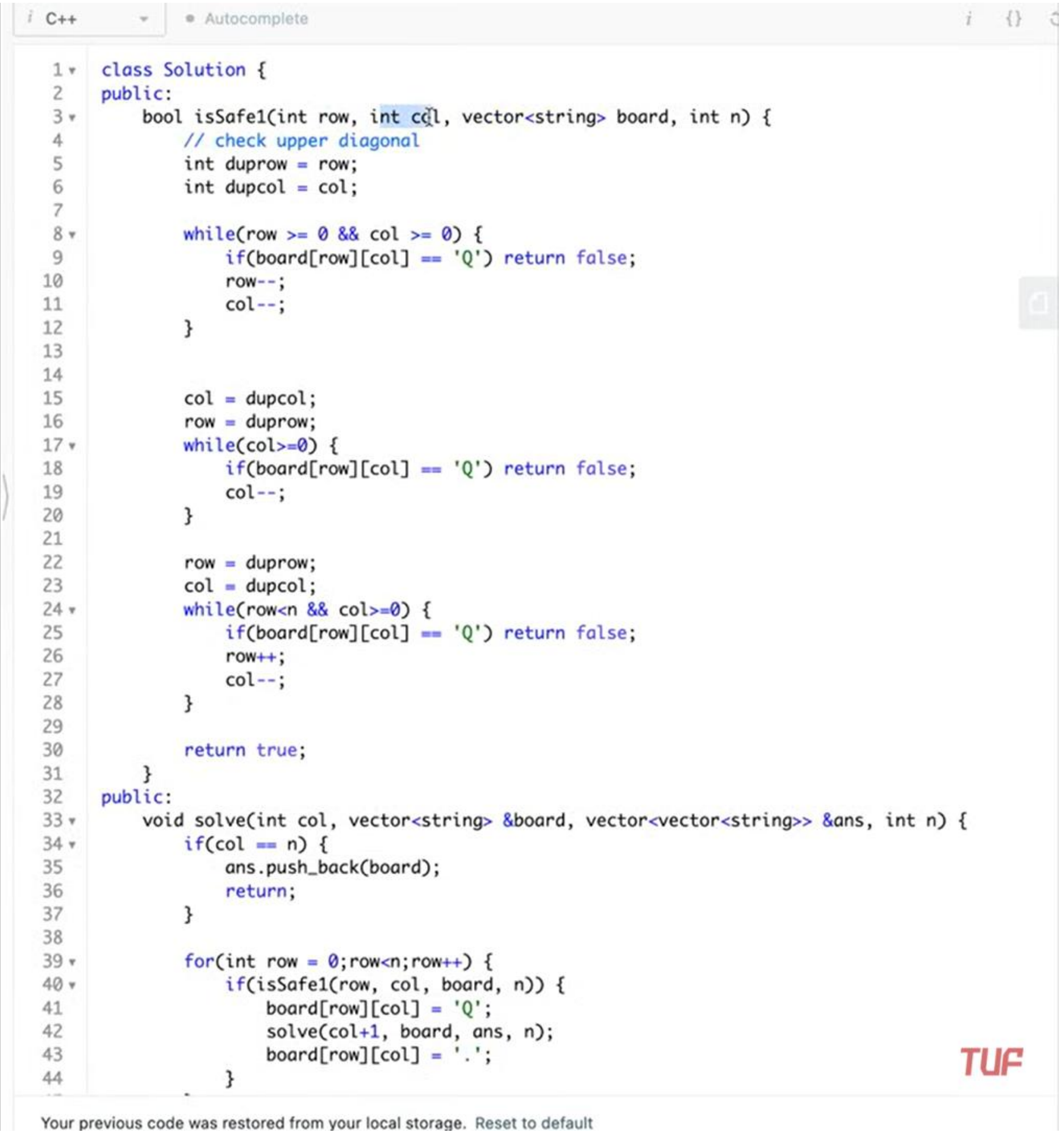

C++ Autocomplete

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

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

        return true;
    }
public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }

        for(int row = 0;row<n;row++) {
            if(isSafe1(row, col, board, n)) {
                board[row][col] = 'Q';
                solve(col+1, board, ans, n);
                board[row][col] = '.';
            }
        }
    }
public:
    vector<vector<string>> solveNQueens(int n) {
        vector<vector<string>> ans;
        vector<string> board(n);
        string s(n, '.');
        for(int i = 0;i<n;i++) {
            board[i] = s;
        }

        solve(0,board, ans, n);
        return ans;
    }

};
```

Your previous code was restored from your local storage. Reset to default

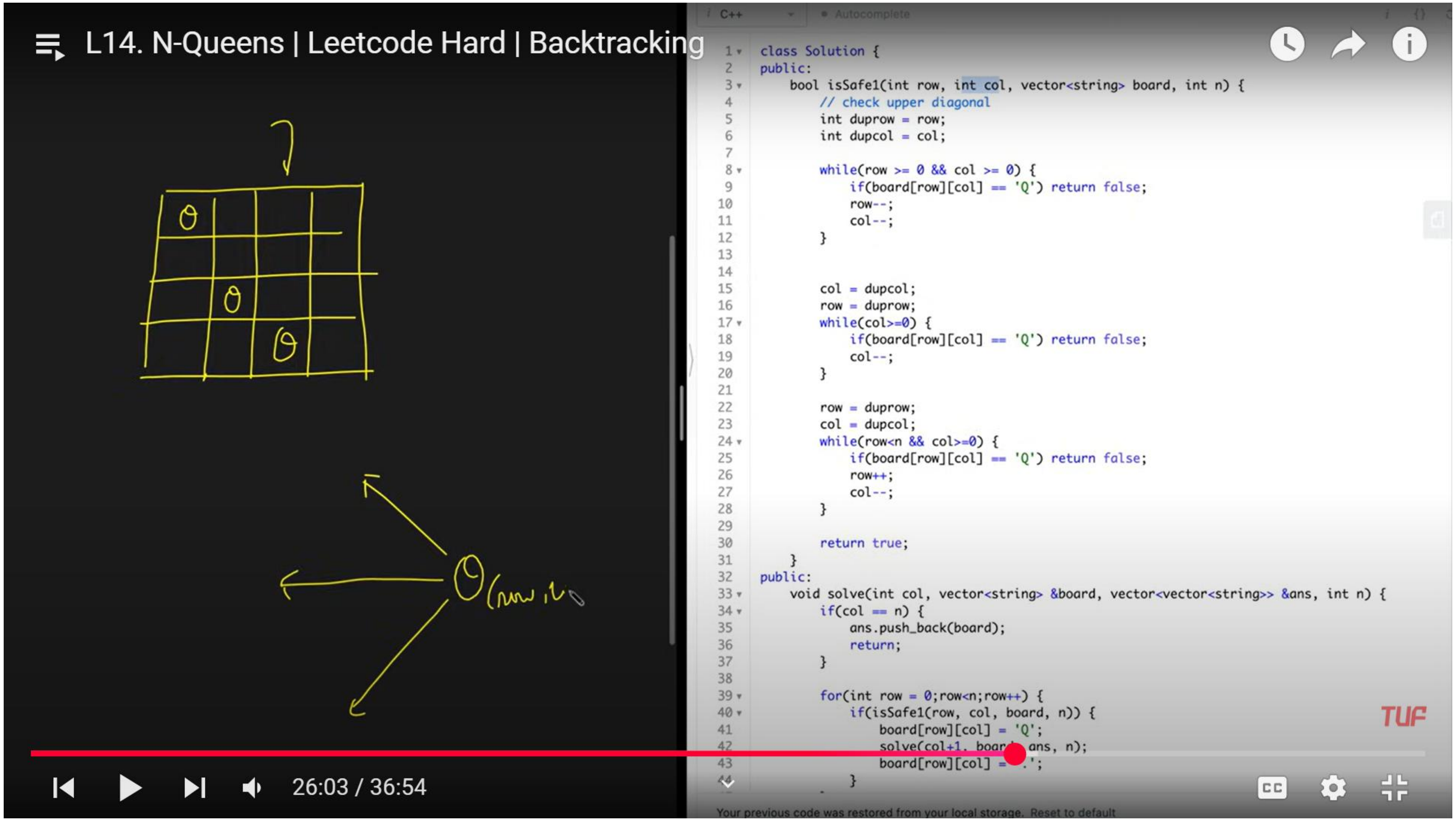






```cpp
class Solution {
public:
    bool isSafe1(int row, int col, vector<string> board, int n) {
        // check upper diagonal
        int duprow = row;
        int dupcol = col;

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


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

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

        return true;
    }
public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }

        for(int row = 0;row<n;row++) {
            if(isSafe1(row, col, board, n)) {
                board[row][col] = 'Q';
                solve(col+1, board, ans, n);
                board[row][col] = '.';
            }
```

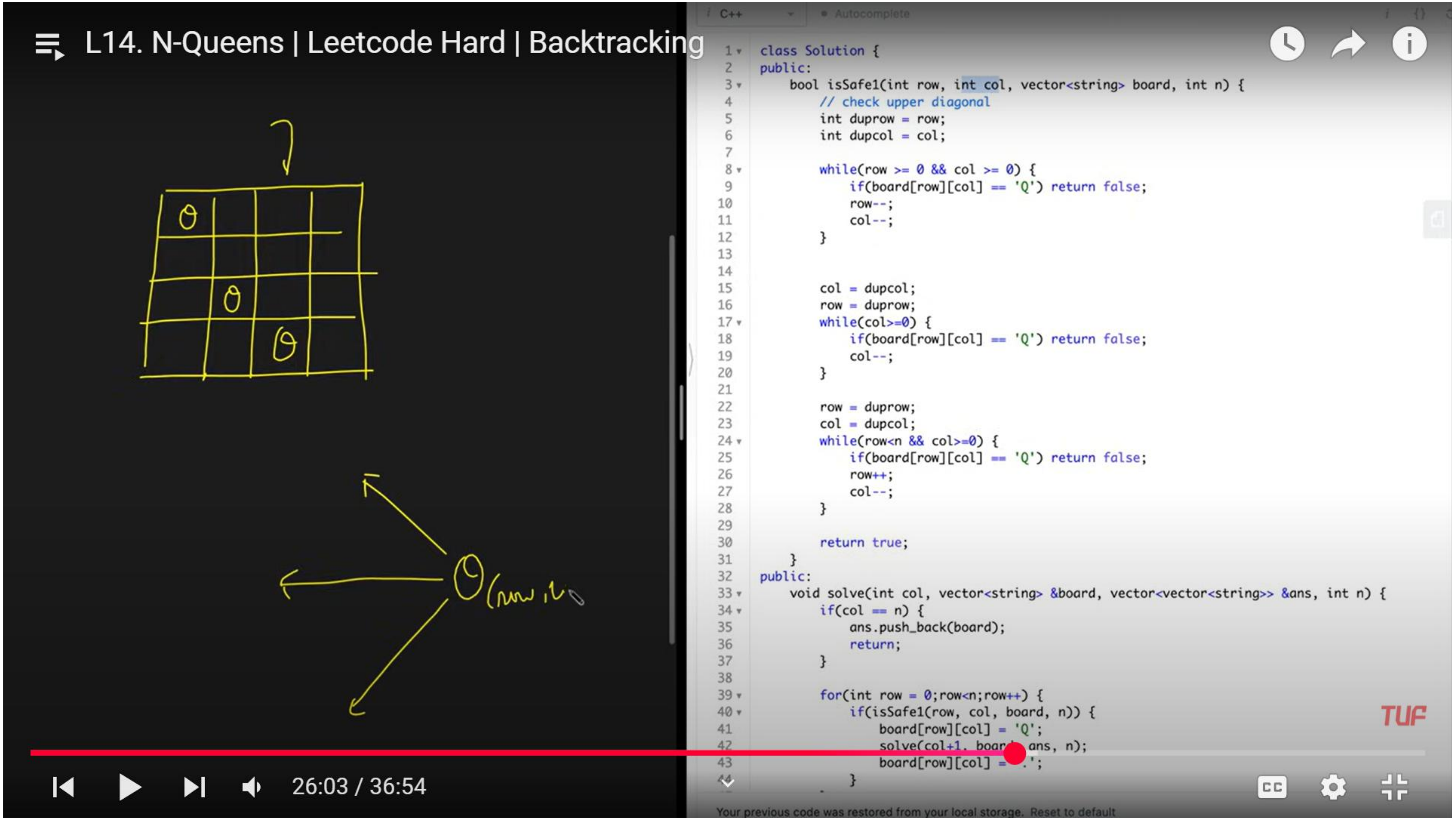
Your previous code was restored from your local storage. Reset to default



L14. N-Queens | Leetcode Hard | Backtracking

```
class Solution {
public:
    bool isSafe1(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--;
        }


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

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

        return true;
    }
public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }

        for(int row = 0;row<n;row++) {
            if(isSafe1(row, col, board, n)) {
                board[row][col] = 'Q';
                solve(col+1, board, ans, n);
                board[row][col] = '.';
            }
```

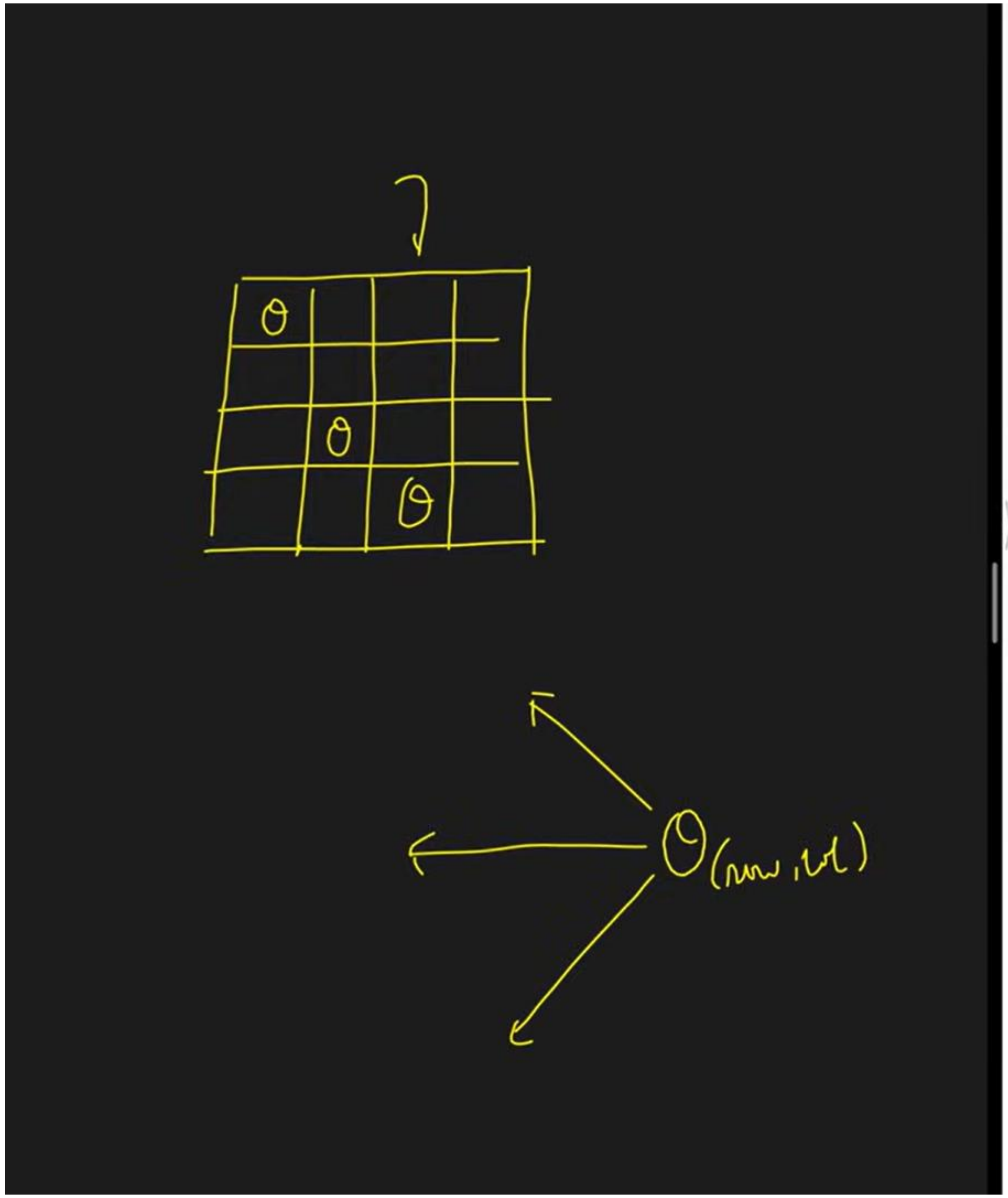
Your previous code was restored from your local storage. Reset to default

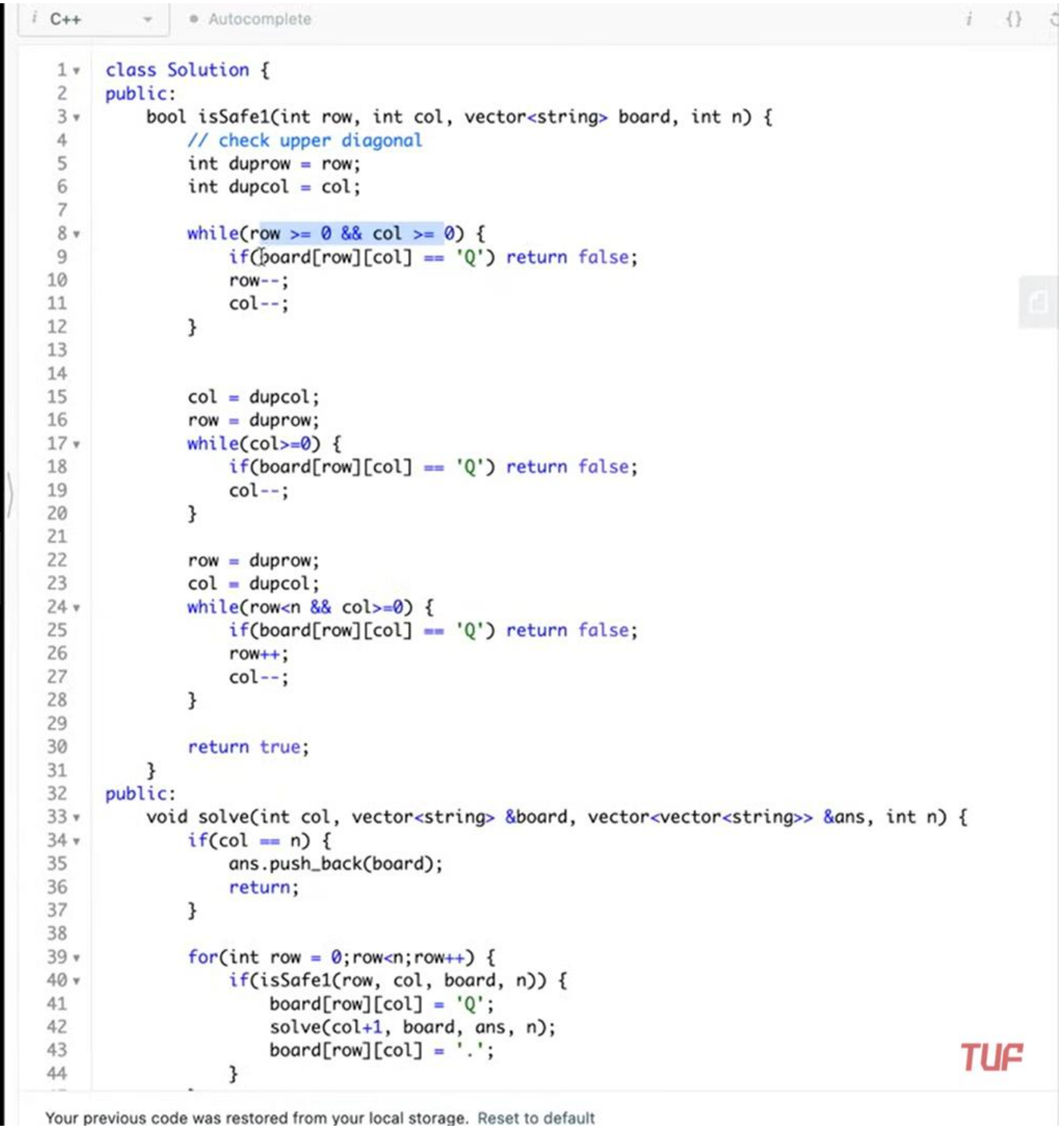


L14. N-Queens | Leetcode Hard | Backtracking

```
class Solution {
public:
    bool isSafe1(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--;
        }


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

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

        return true;
    }
public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }

        for(int row = 0;row<n;row++) {
            if(isSafe1(row, col, board, n)) {
                board[row][col] = 'Q';
                solve(col+1, board, ans, n);
                board[row][col] = '.';
            }
```

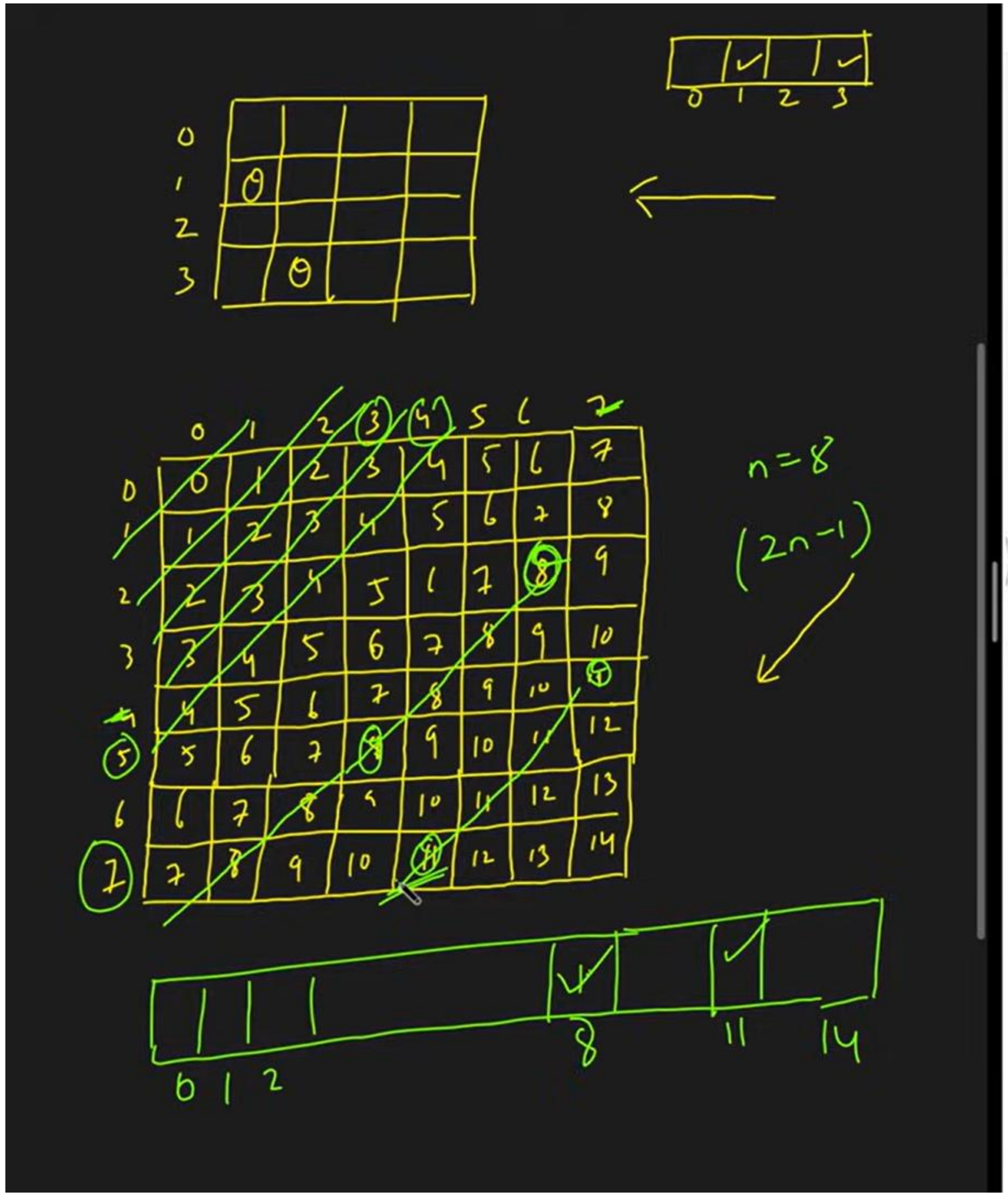
Your previous code was restored from your local storage. Reset to default

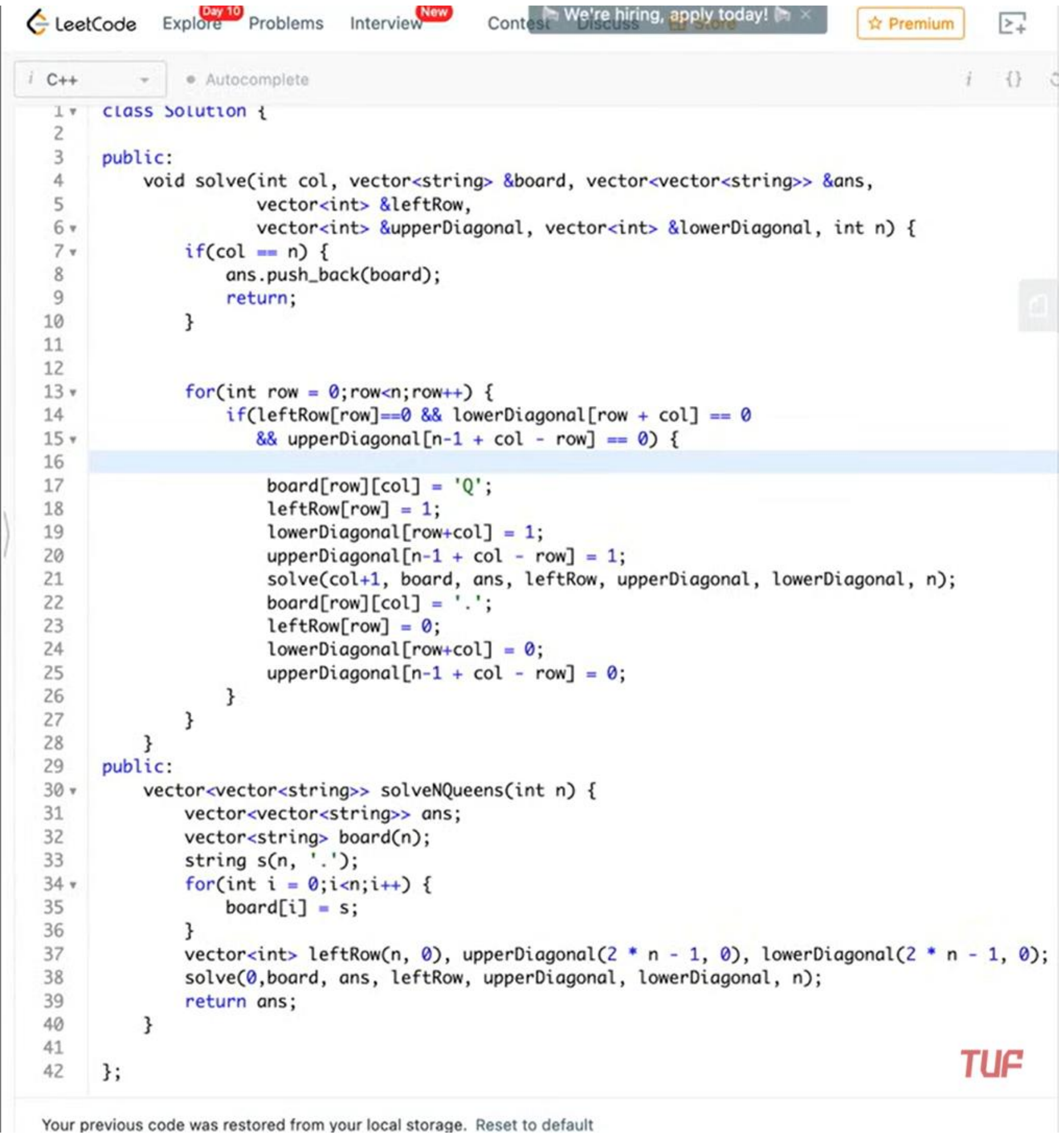




```cpp
class Solution {
public:
    bool isSafe1(int row, int col, vector<string> board, int n) {
        // check upper diagonal
        int duprow = row;
        int dupcol = col;

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


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

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

        return true;
    }
public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }

        for(int row = 0;row<n;row++) {
            if(isSafe1(row, col, board, n)) {
                board[row][col] = 'Q';
                solve(col+1, board, ans, n);
                board[row][col] = '.';
            }
```

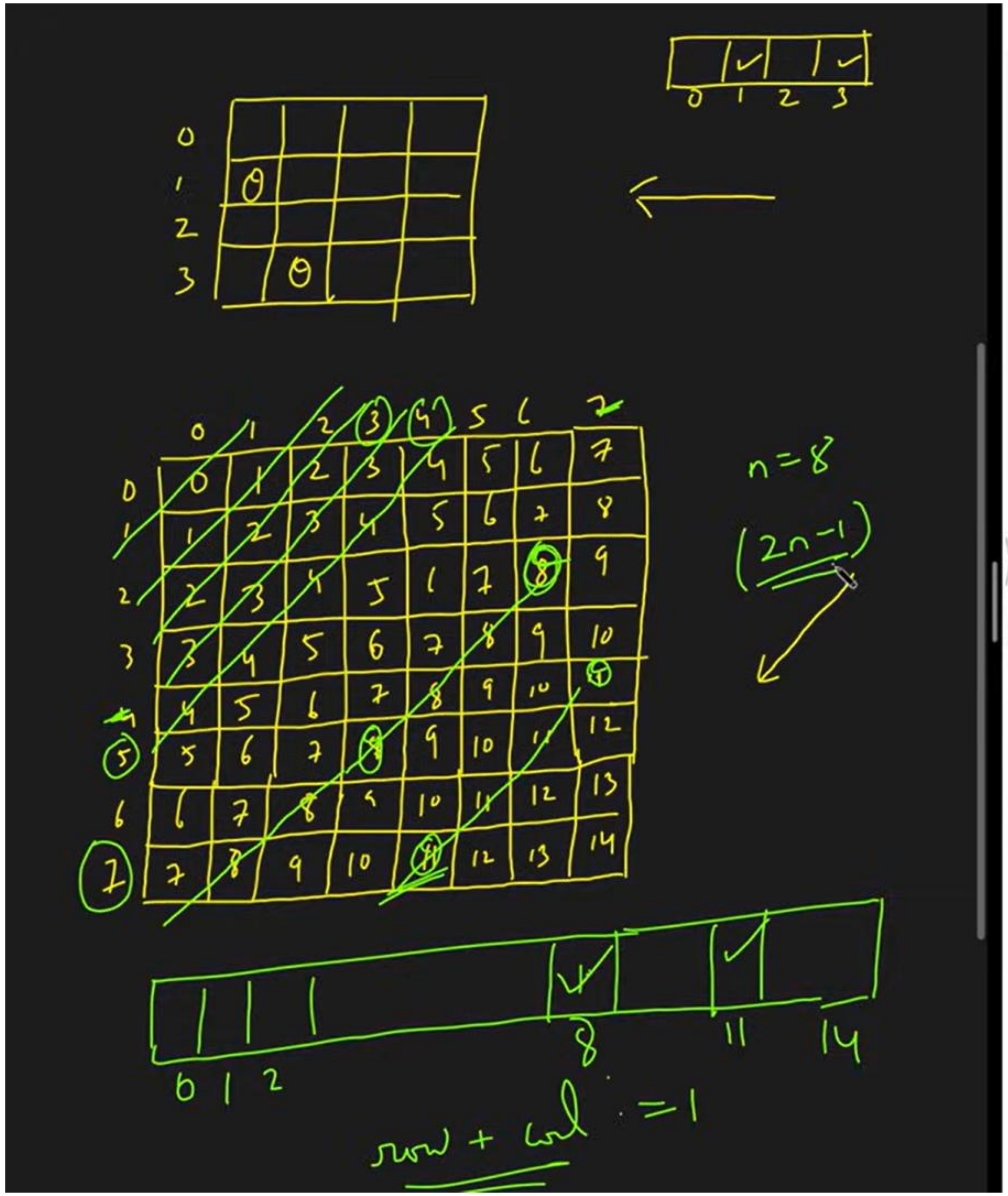
Your previous code was restored from your local storage. Reset to default

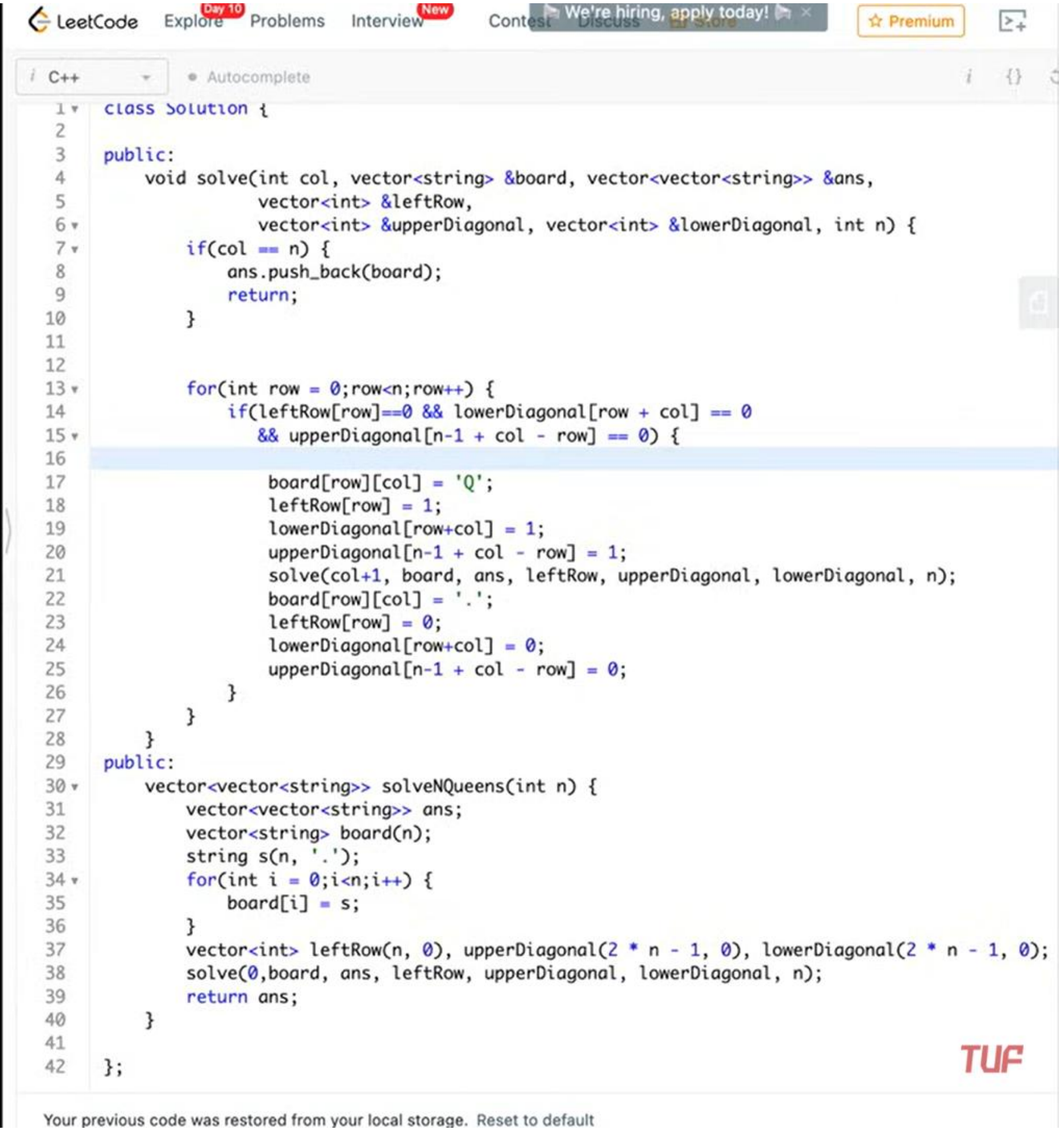




```cpp
class Solution {

public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans,
               vector<int> &leftRow,
               vector<int> &upperDiagonal, vector<int> &lowerDiagonal, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }


        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';
                leftRow[row] = 1;
                lowerDiagonal[row+col] = 1;
                upperDiagonal[n-1 + col - row] = 1;
                solve(col+1, board, ans, leftRow, upperDiagonal, lowerDiagonal, n);
                board[row][col] = '.';
                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);
        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);
        solve(0,board, ans, leftRow, upperDiagonal, lowerDiagonal, n);
        return ans;
    }

};
```

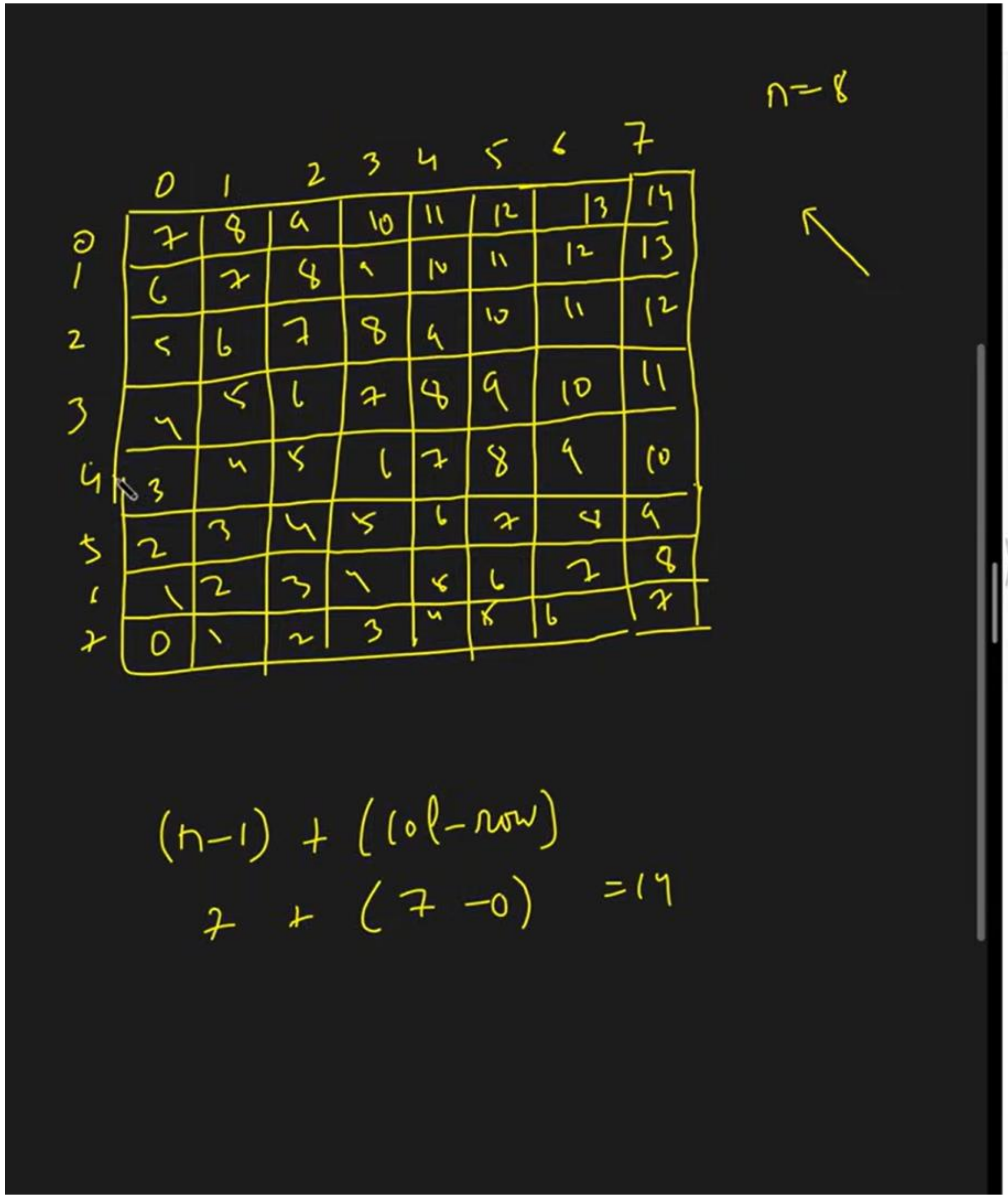
Your previous code was restored from your local storage. Reset to default





```cpp
class Solution {

public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans,
               vector<int> &leftRow,
               vector<int> &upperDiagonal, vector<int> &lowerDiagonal, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }


        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';
                leftRow[row] = 1;
                lowerDiagonal[row+col] = 1;
                upperDiagonal[n-1 + col - row] = 1;
                solve(col+1, board, ans, leftRow, upperDiagonal, lowerDiagonal, n);
                board[row][col] = '.';
                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);
        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);
        solve(0,board, ans, leftRow, upperDiagonal, lowerDiagonal, n);
        return ans;
    }

};
```

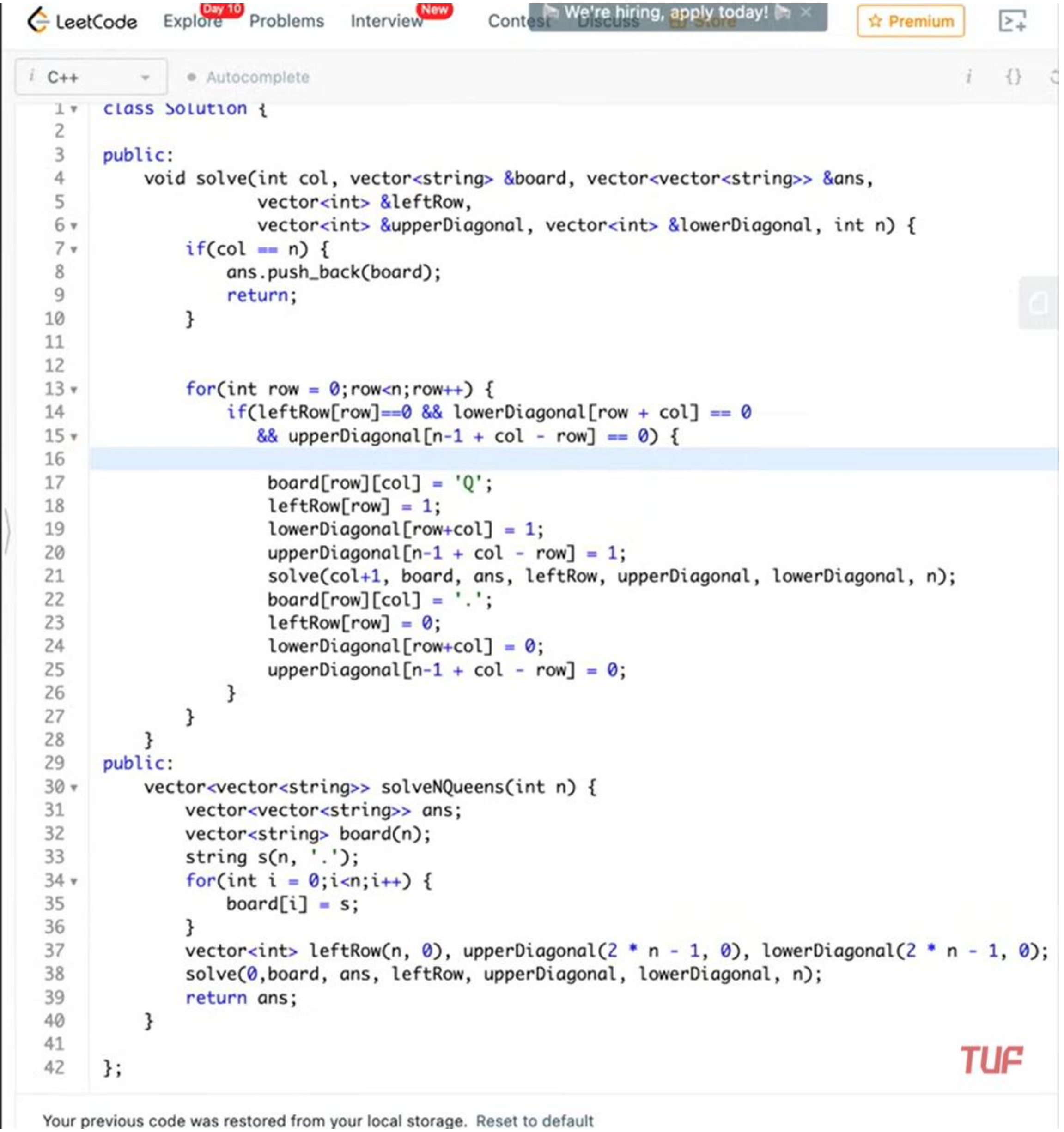
Your previous code was restored from your local storage. Reset to default



$n=8$

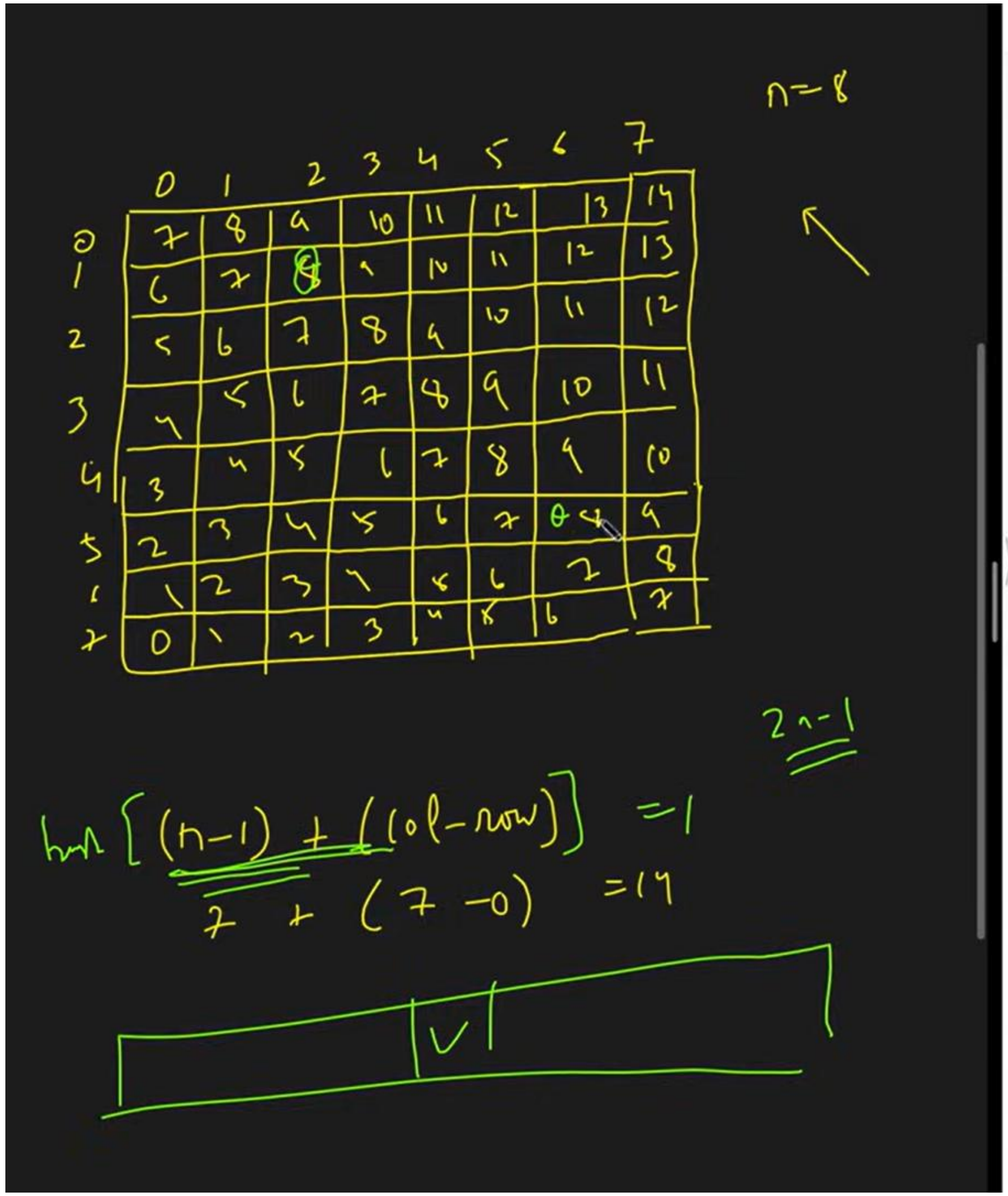
	0	1	2	3	4	5	6	7
0	7	8	9	10	11	12	13	14
1	6	7	8	9	10	11	12	13
2	5	6	7	8	9	10	11	12
3	4	5	6	7	8	9	10	11
4	3	4	5	6	7	8	9	10
5	2	3	4	5	6	7	8	9
6	1	2	3	4	5	6	7	8
7	0	1	2	3	4	5	6	7

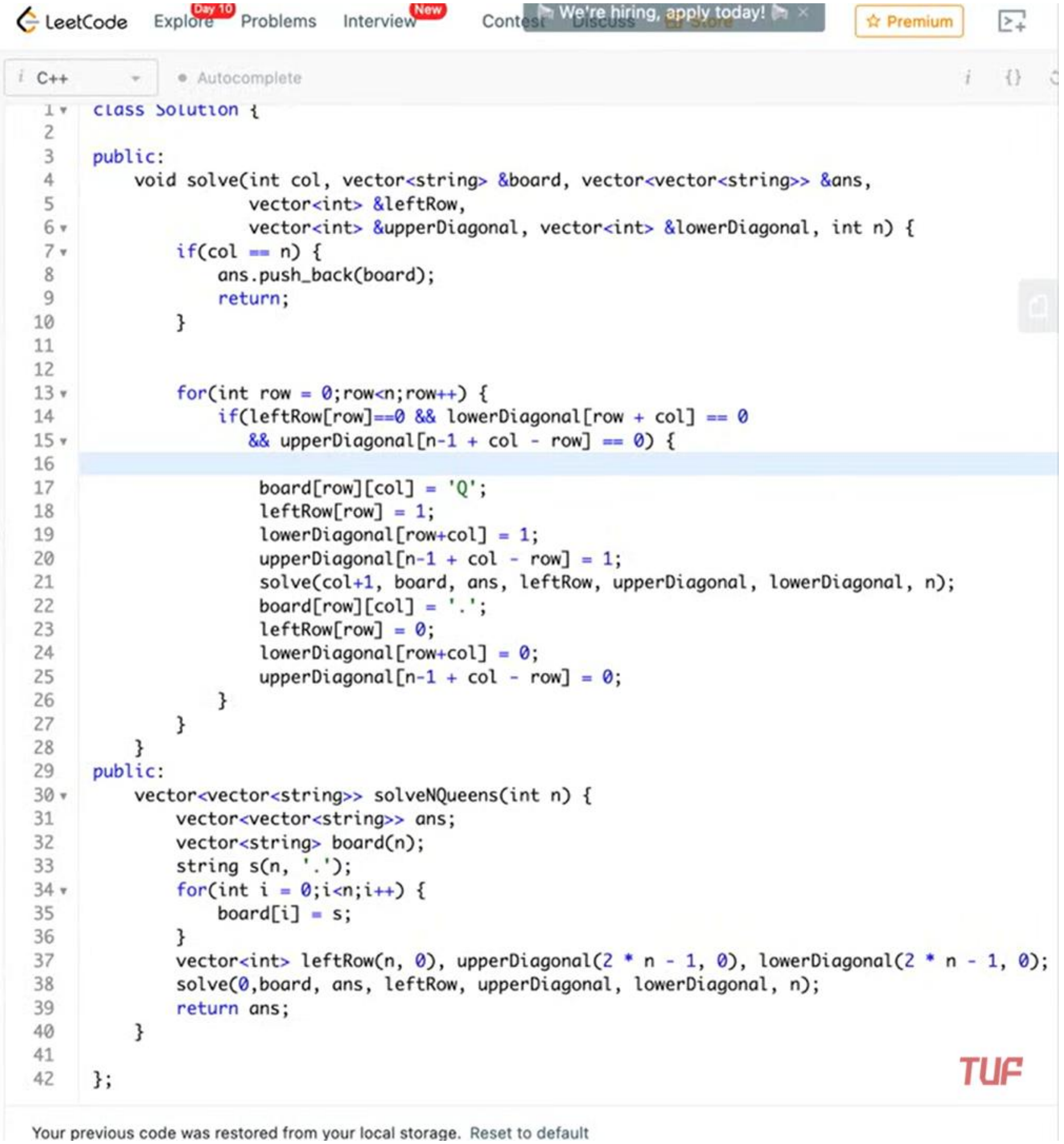
$$(n-1) + (col - row)$$
$$7 + (7 - 0) = 14$$



```cpp
class Solution {

public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans,
               vector<int> &leftRow,
               vector<int> &upperDiagonal, vector<int> &lowerDiagonal, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }


        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';
                leftRow[row] = 1;
                lowerDiagonal[row+col] = 1;
                upperDiagonal[n-1 + col - row] = 1;
                solve(col+1, board, ans, leftRow, upperDiagonal, lowerDiagonal, n);
                board[row][col] = '.';
                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);
        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);
        solve(0,board, ans, leftRow, upperDiagonal, lowerDiagonal, n);
        return ans;
    }

};
```

Your previous code was restored from your local storage. Reset to default

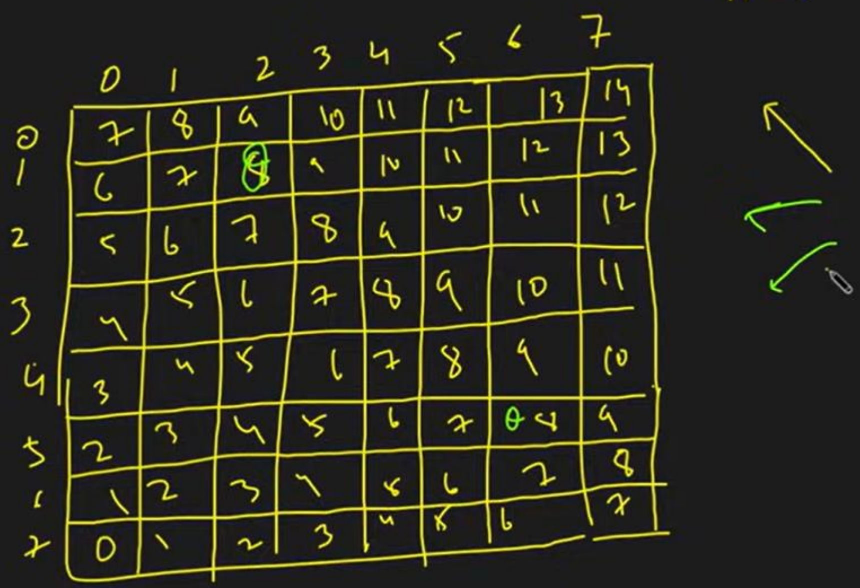




```cpp
class Solution {

public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans,
               vector<int> &leftRow,
               vector<int> &upperDiagonal, vector<int> &lowerDiagonal, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }


        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';
                leftRow[row] = 1;
                lowerDiagonal[row+col] = 1;
                upperDiagonal[n-1 + col - row] = 1;
                solve(col+1, board, ans, leftRow, upperDiagonal, lowerDiagonal, n);
                board[row][col] = '.';
                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);
        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);
        solve(0,board, ans, leftRow, upperDiagonal, lowerDiagonal, n);
        return ans;
    }

};
```

Your previous code was restored from your local storage. Reset to default

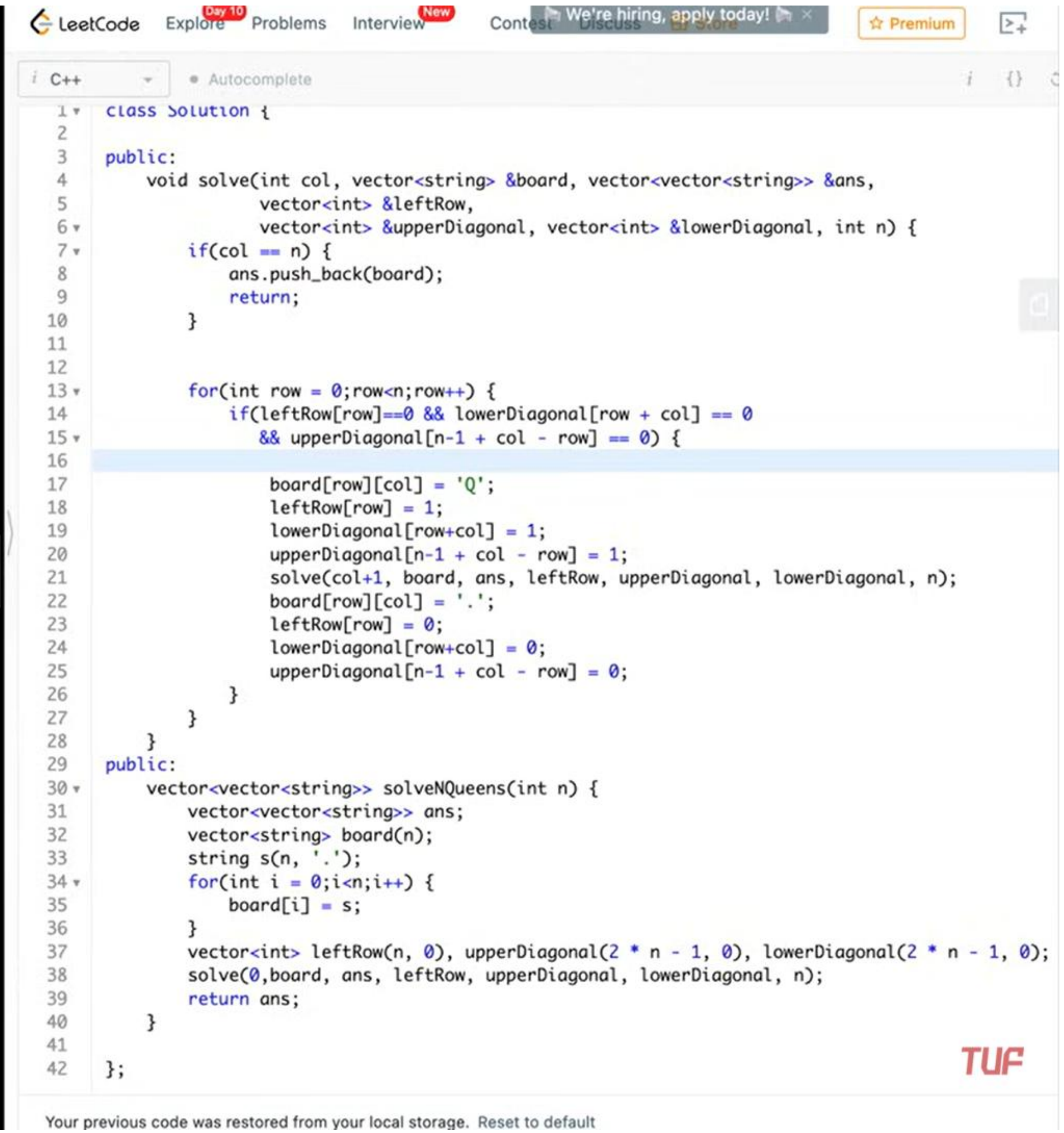
$n=8$



$$\text{hmh}\left[\underline{(n-1)}+(col-row)\right]=1$$

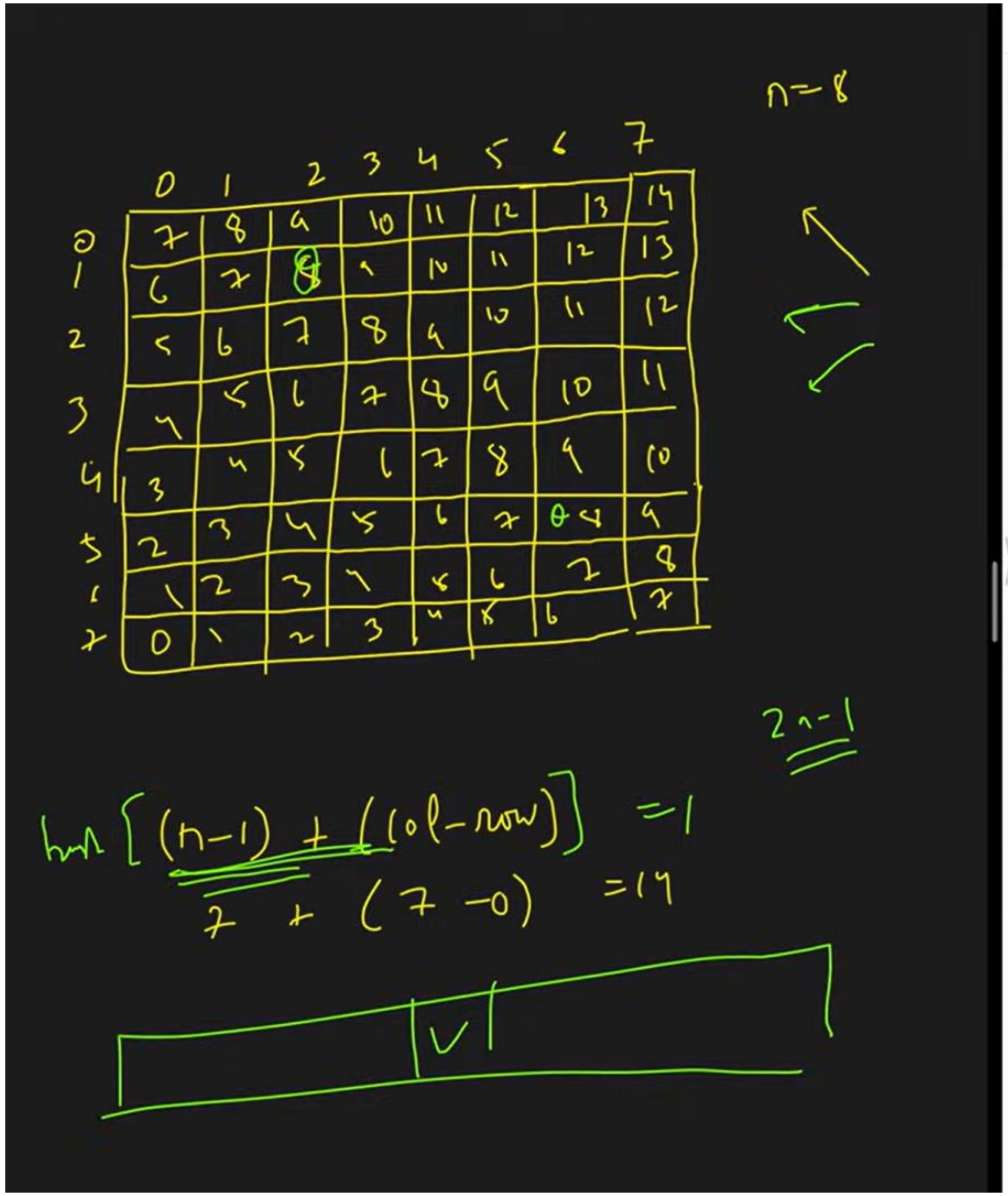
$$7+(7-0)=14$$

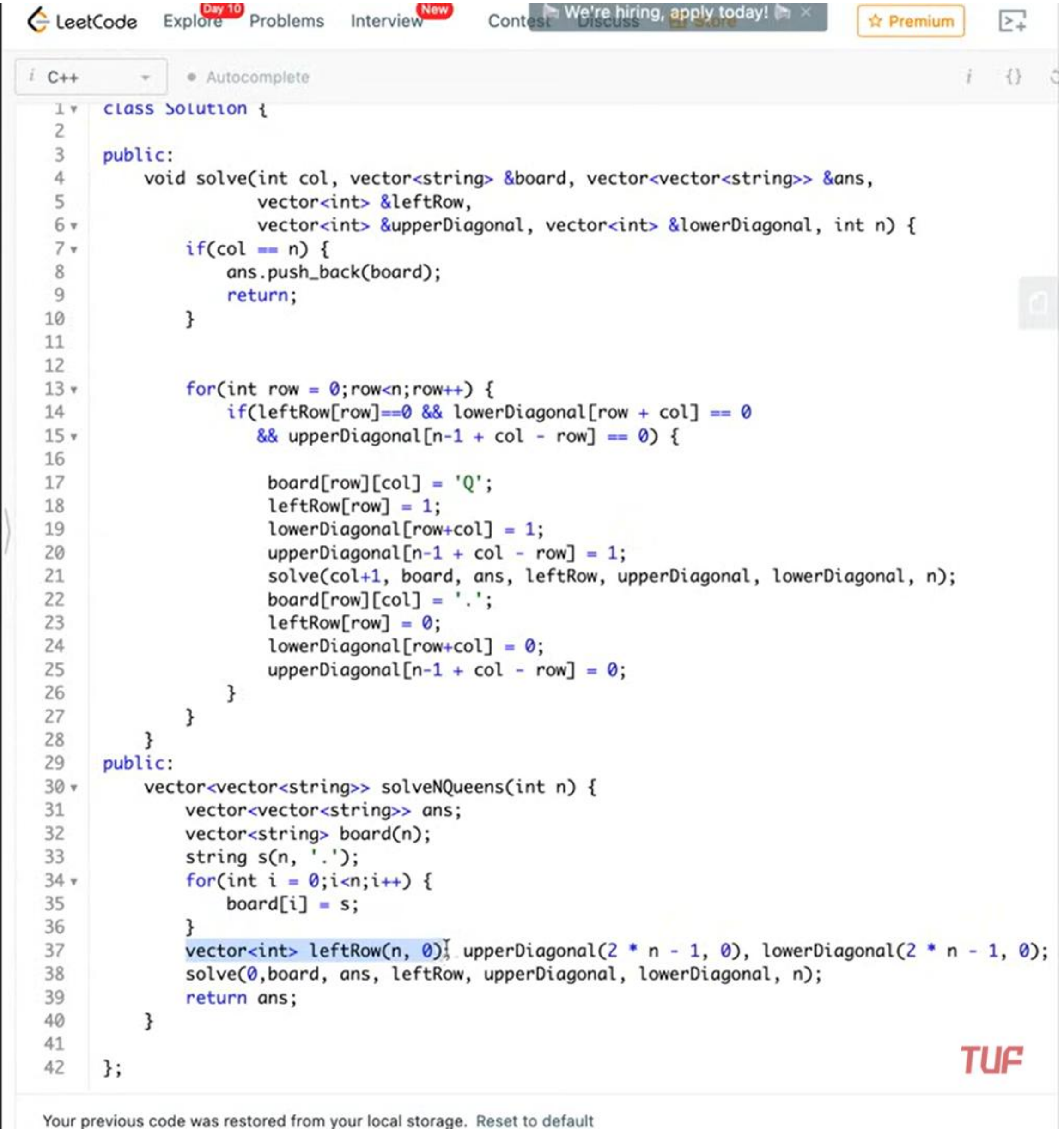
$$\underline{\underline{2n-1}}$$



```cpp
class Solution {

public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans,
               vector<int> &leftRow,
               vector<int> &upperDiagonal, vector<int> &lowerDiagonal, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }


        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';
                leftRow[row] = 1;
                lowerDiagonal[row+col] = 1;
                upperDiagonal[n-1 + col - row] = 1;
                solve(col+1, board, ans, leftRow, upperDiagonal, lowerDiagonal, n);
                board[row][col] = '.';
                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);
        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);
        solve(0,board, ans, leftRow, upperDiagonal, lowerDiagonal, n);
        return ans;
    }

};
```

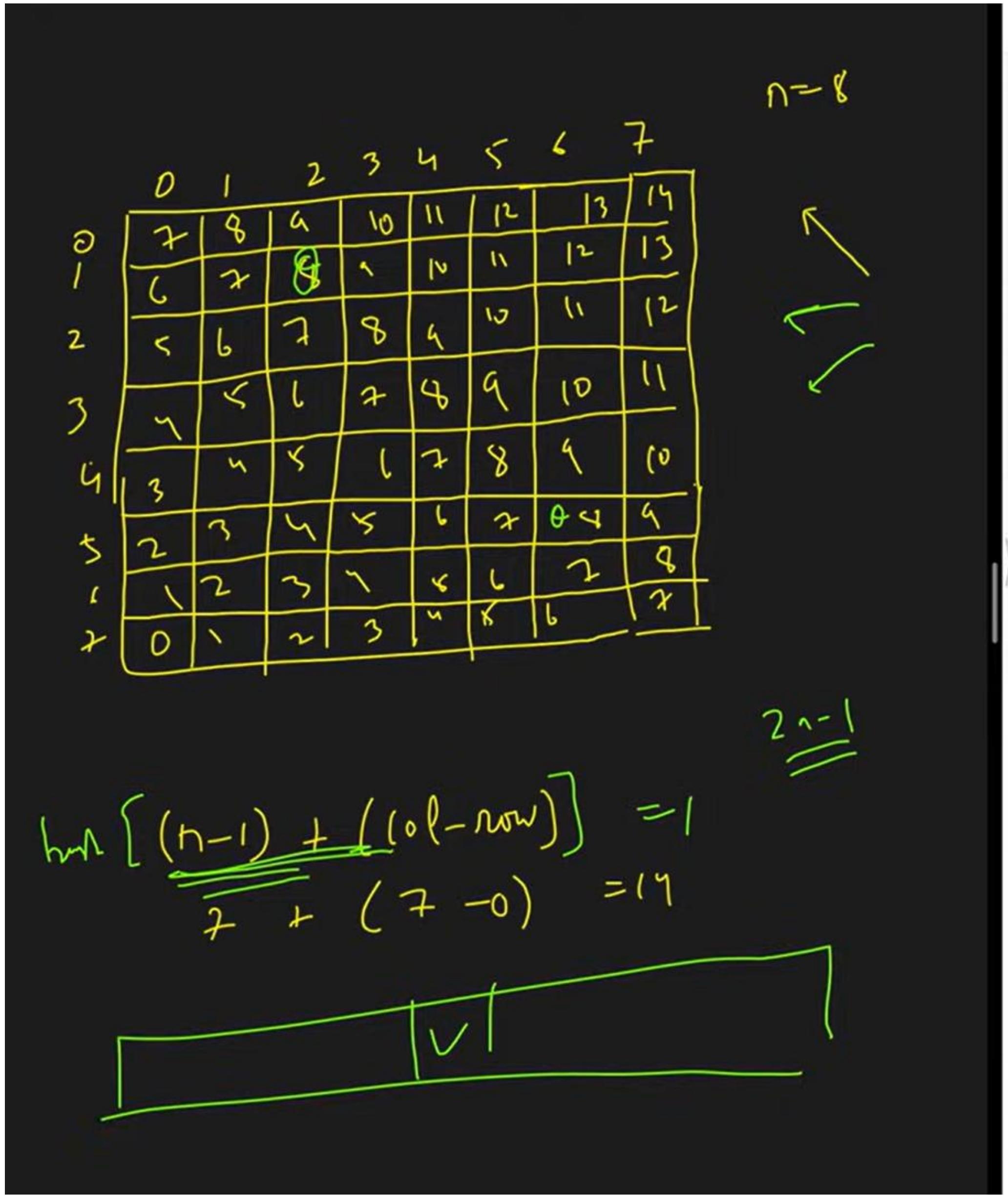
Your previous code was restored from your local storage. Reset to default

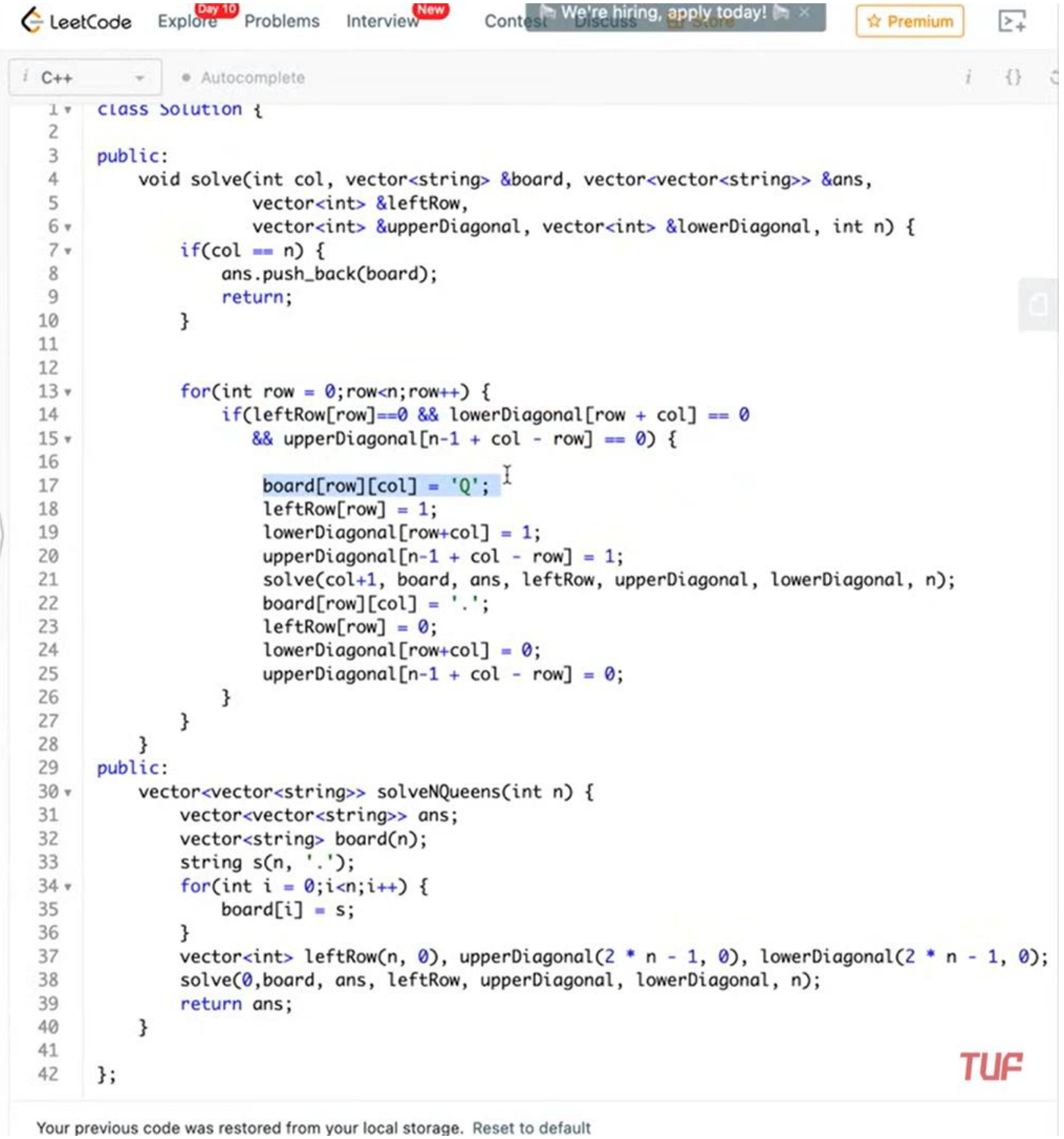




```cpp
class Solution {

public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans,
               vector<int> &leftRow,
               vector<int> &upperDiagonal, vector<int> &lowerDiagonal, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }


        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';
                leftRow[row] = 1;
                lowerDiagonal[row+col] = 1;
                upperDiagonal[n-1 + col - row] = 1;
                solve(col+1, board, ans, leftRow, upperDiagonal, lowerDiagonal, n);
                board[row][col] = '.';
                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);
        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);
        solve(0,board, ans, leftRow, upperDiagonal, lowerDiagonal, n);
        return ans;
    }

};
```

Your previous code was restored from your local storage. Reset to default





```cpp
class Solution {

public:
    void solve(int col, vector<string> &board, vector<vector<string>> &ans,
               vector<int> &leftRow,
               vector<int> &upperDiagonal, vector<int> &lowerDiagonal, int n) {
        if(col == n) {
            ans.push_back(board);
            return;
        }


        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';
                leftRow[row] = 1;
                lowerDiagonal[row+col] = 1;
                upperDiagonal[n-1 + col - row] = 1;
                solve(col+1, board, ans, leftRow, upperDiagonal, lowerDiagonal, n);
                board[row][col] = '.';
                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);
        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);
        solve(0,board, ans, leftRow, upperDiagonal, lowerDiagonal, n);
        return ans;
    }

};
```

Your previous code was restored from your local storage. Reset to default