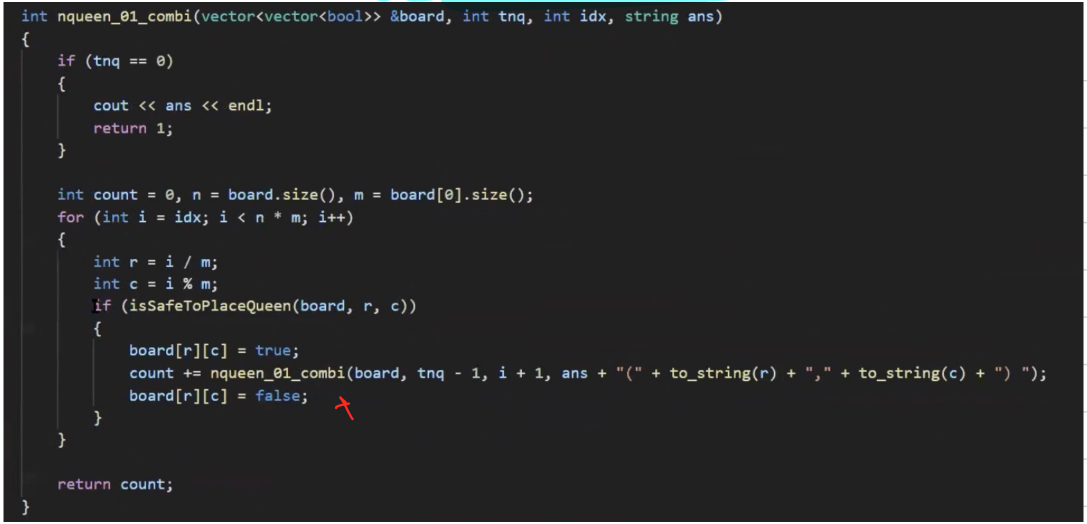
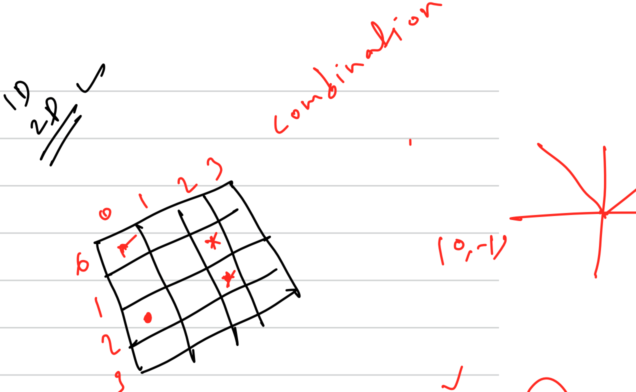
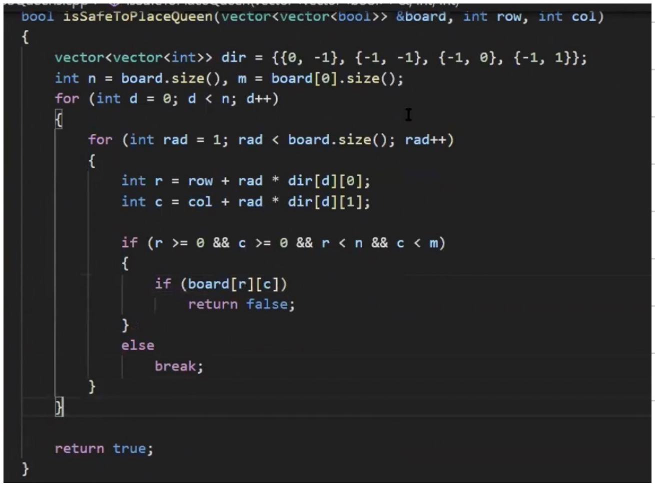
n queens :- (combination, direction arr + radius concept)

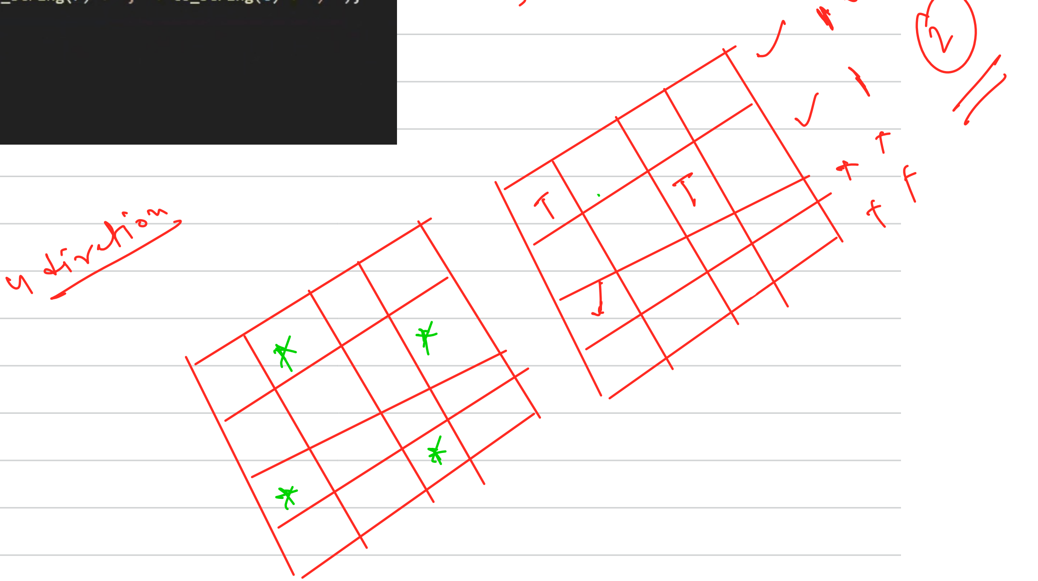


```
int nqueen_01_combi(vector<vector<bool>> &board, int tnq, int idx, string ans)
{
    if (tnq == 0)
    {
        cout << ans << endl;
        return 1;
    }

    int count = 0, n = board.size(), m = board[0].size();
    for (int i = idx; i < n * m; i++)
    {
        int r = i / m;
        int c = i % m;
        if (isSafeToPlaceQueen(board, r, c))
        {
            board[r][c] = true;
            count += nqueen_01_combi(board, tnq - 1, i + 1, ans + "(" + to_string(r) + "," + to_string(c) + ") ");
            board[r][c] = false;
        }
    }

    return count;
}
```

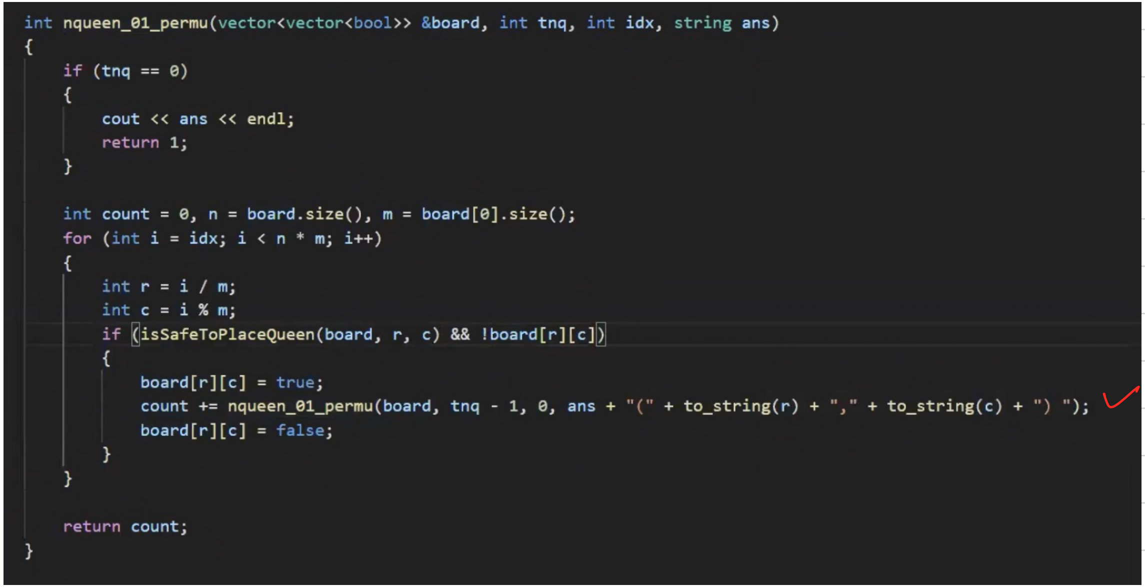




```
bool isSafeToPlaceQueen(vector<vector<bool>> &board, int row, int col)
{
    vector<vector<int>> dir = {{0, -1}, {-1, -1}, {-1, 0}, {-1, 1}};
    int n = board.size(), m = board[0].size();
    for (int d = 0; d < n; d++)
    {
        for (int rad = 1; rad < board.size(); rad++)
        {
            int r = row + rad * dir[d][0];
            int c = col + rad * dir[d][1];

            if (r >= 0 && c >= 0 && r < n && c < m)
            {
                if (board[r][c])
                    return false;
            }
            else
                break;
        }
    }

    return true;
}
```

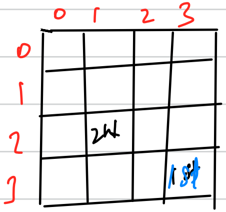


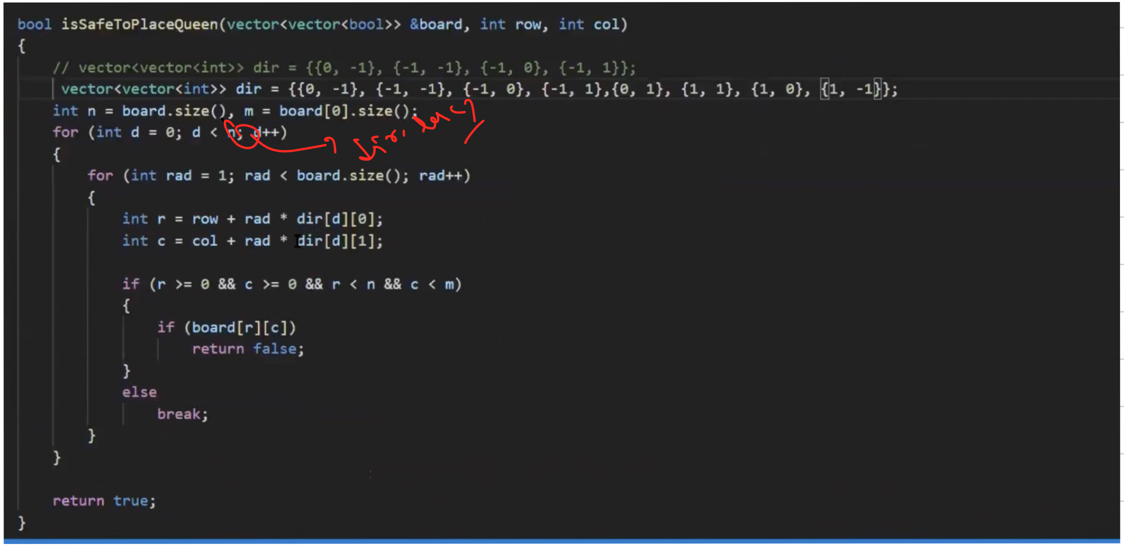
n queen using Permutation:-



```
int nqueen_01_permu(vector<vector<bool>> &board, int tnq, int idx, string ans)
{
    if (tnq == 0)
    {
        cout << ans << endl;
        return 1;
    }

    int count = 0, n = board.size(), m = board[0].size();
    for (int i = idx; i < n * m; i++)
    {
        int r = i / m;
        int c = i % m;
        if (isSafeToPlaceQueen(board, r, c) && !board[r][c])
        {
            board[r][c] = true;
            count += nqueen_01_permu(board, tnq - 1, 0, ans + "(" + to_string(r) + "," + to_string(c) + ") ");
            board[r][c] = false;
        }
    }

    return count;
}
```

1. check in all direction in issafemethod()

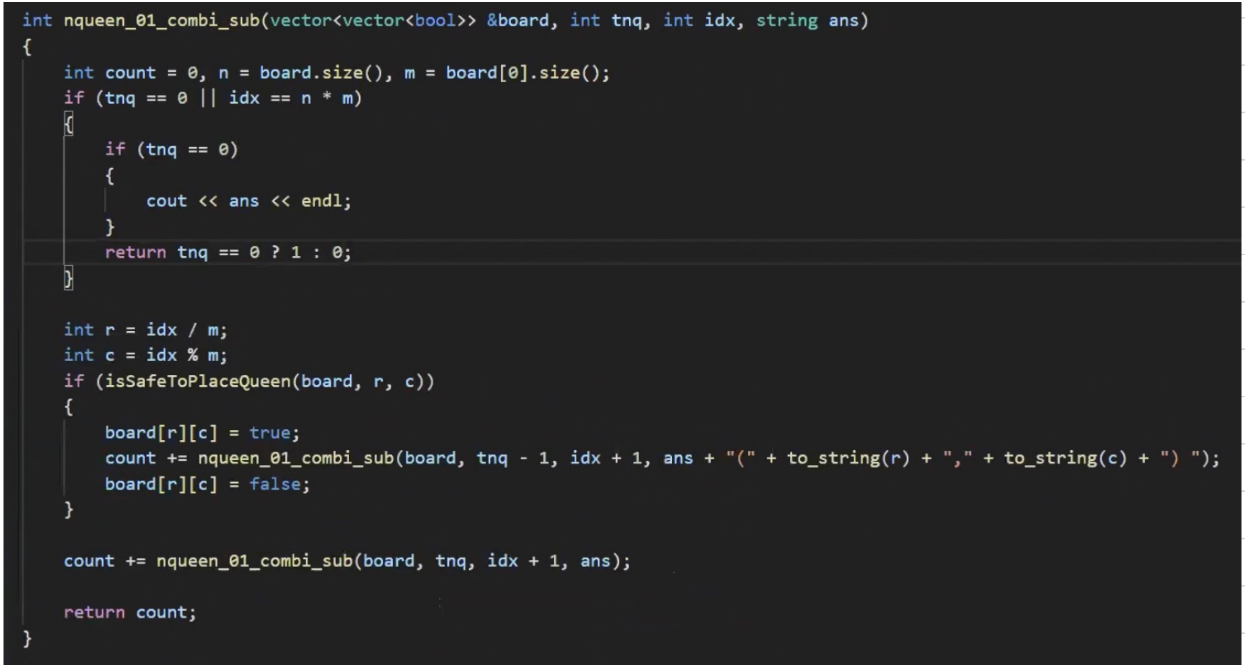




```cpp
bool isSafeToPlaceQueen(vector<vector<bool>> &board, int row, int col)
{
    // vector<vector<int>> dir = {{0, -1}, {-1, -1}, {-1, 0}, {-1, 1}};
    vector<vector<int>> dir = {{0, -1}, {-1, -1}, {-1, 0}, {-1, 1},{0, 1}, {1, 1}, {1, 0}, {1, -1}};
    int n = board.size(), m = board[0].size();
    for (int d = 0; d < n; d++)
    {
        for (int rad = 1; rad < board.size(); rad++)
        {
            int r = row + rad * dir[d][0];
            int c = col + rad * dir[d][1];

            if (r >= 0 && c >= 0 && r < n && c < m)
            {
                if (board[r][c])
                    return false;
            }
            else
                break;
        }
    }

    return true;
}
```

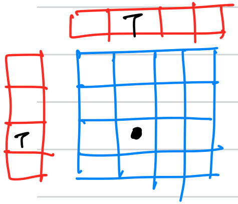
dir.size() ✓

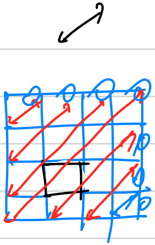
Subsequence:



```
int nqueen_01_combi_sub(vector<vector<bool>> &board, int tnq, int idx, string ans)
{
    int count = 0, n = board.size(), m = board[0].size();
    if (tnq == 0 || idx == n * m)
    {
        if (tnq == 0)
        {
            cout << ans << endl;
        }
        return tnq == 0 ? 1 : 0;
    }

    int r = idx / m;
    int c = idx % m;
    if (isSafeToPlaceQueen(board, r, c))
    {
        board[r][c] = true;
        count += nqueen_01_combi_sub(board, tnq - 1, idx + 1, ans + "(" + to_string(r) + "," + to_string(c) + ") ");
        board[r][c] = false;
    }

    count += nqueen_01_combi_sub(board, tnq, idx + 1, ans);

    return count;
}
```

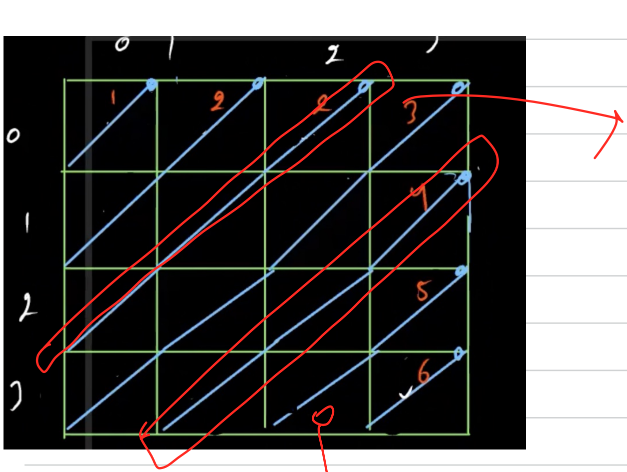
//optimised n-queen





no. of diagonals

(n+m-1)

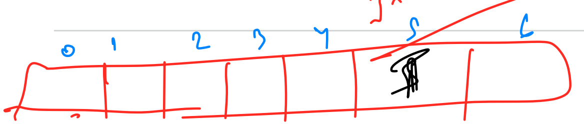


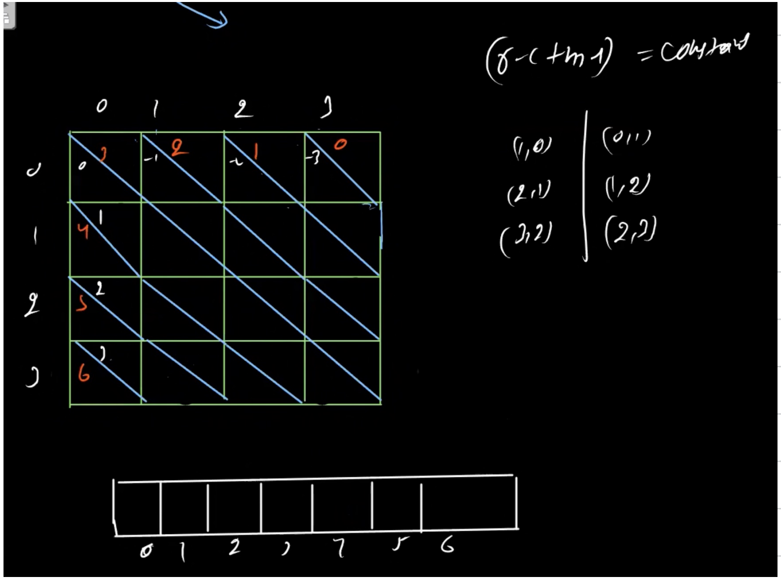
r+c = constant

(0,2)	(1,3)
(1,1)	(2,2)
(2,0)	(3,1)

7

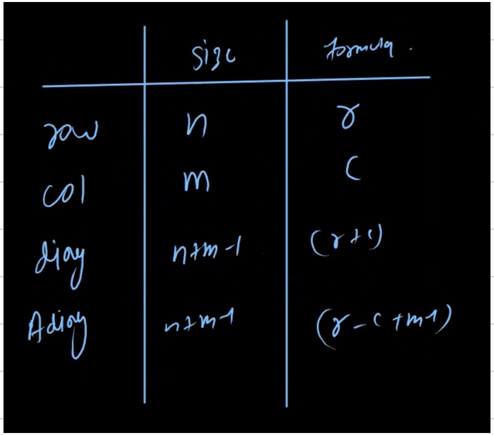
3+2 = 5



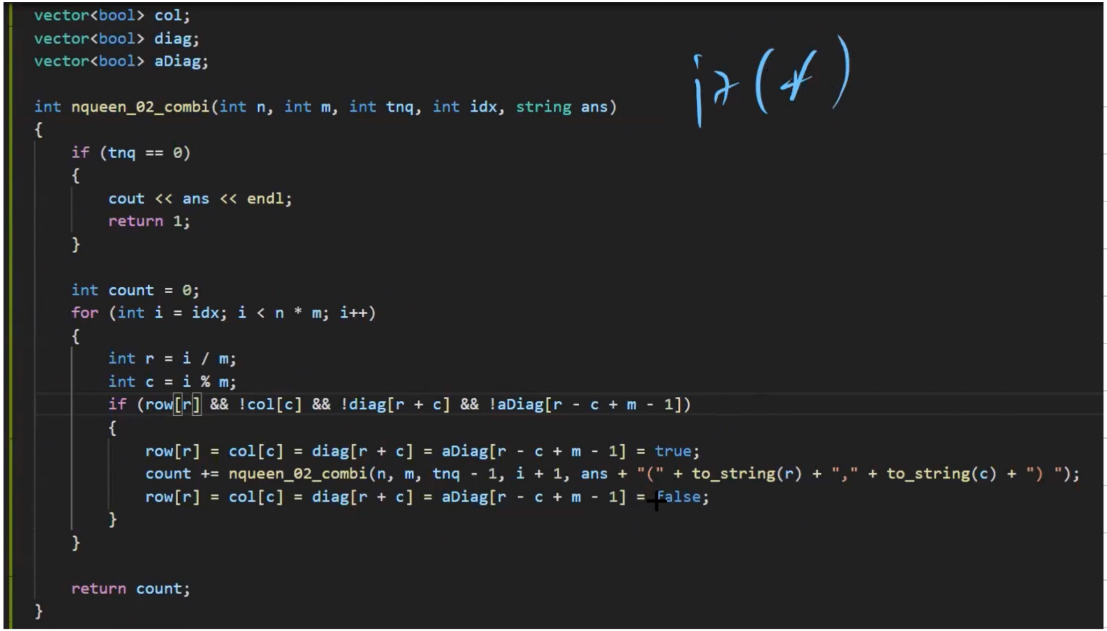


$(r-c+m-1) = \text{constant}$

(1,0)	(0,1)
(2,1)	(1,2)
(3,2)	(2,3)

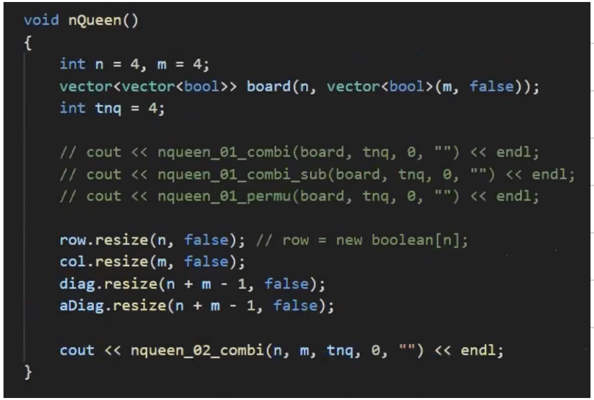


	size	formula
row	n	r
col	m	c
diag	n+m-1	(r+c)
Adiag	n+m-1	(r-c+m-1)



```
vector<bool> col;
vector<bool> diag;
vector<bool> aDiag;

int nqueen_02_combi(int n, int m, int tnq, int idx, string ans)
{
    if (tnq == 0)
    {
        cout << ans << endl;
        return 1;
    }

    int count = 0;
    for (int i = idx; i < n * m; i++)
    {
        int r = i / m;
        int c = i % m;
        if (row[r] && !col[c] && !diag[r + c] && !aDiag[r - c + m - 1])
        {
            row[r] = col[c] = diag[r + c] = aDiag[r - c + m - 1] = true;
            count += nqueen_02_combi(n, m, tnq - 1, i + 1, ans + "(" + to_string(r) + "," + to_string(c) + ") ");
            row[r] = col[c] = diag[r + c] = aDiag[r - c + m - 1] = false;
        }
    }

    return count;
}
```

i7(4)



```
void nQueen()
{
    int n = 4, m = 4;
    vector<vector<bool>> board(n, vector<bool>(m, false));
    int tnq = 4;

    // cout << nqueen_01_combi(board, tnq, 0, "") << endl;
    // cout << nqueen_01_combi_sub(board, tnq, 0, "") << endl;
    // cout << nqueen_01_permu(board, tnq, 0, "") << endl;

    row.resize(n, false); // row = new boolean[n];
    col.resize(m, false);
    diag.resize(n + m - 1, false);
    aDiag.resize(n + m - 1, false);

    cout << nqueen_02_combi(n, m, tnq, 0, "") << endl;
}
```