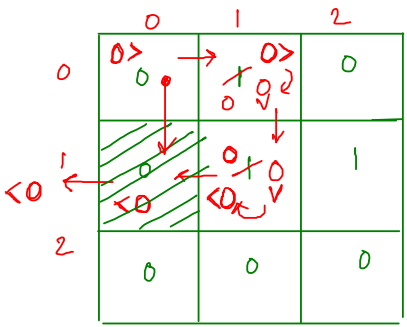
★ Find the way : [Get out of matrix]



op: (1, 0) [Last cell before coming out of matrix]

★ 0 → forward

1 → make it 0, turn right, move forward

0> (right) = 0

0 (down) = 1

<0 (Left) = 2

^0 (top) = 3

★ In order to move forward, we need to know our facing first, (Maintain facing variable) (0, 1, 2, 3)

(i, j) f=0 (i, j+1)

0> → 0>

0 (i, j)

f=1

0 (i+1, j)

f=2

<0 (i, j-1) ← <0 (i, j)

^0 (i-1, j)

f=3

^0 (i, j)

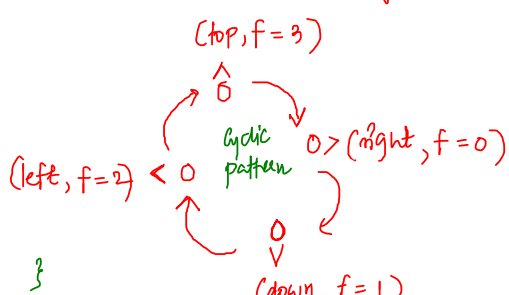
★ How to move forward based on facing.

* How to know the facing ?

Initially, you are at (0,0) facing right

```
let i=0;
let j=0;
let facing = 0;
while (true) {
```

* how to change facing



}

* 1st know your facing
then only you can decide on how to forward.

mat[i][j] = 0 → no change in facing

⇒ In this facing variable will directly give the direction.

mat[i][j] = 1 → facing will change because you need to turn right.

f = 0 1 2 3 0 1 2 3 0 1 2 3 . . . . .

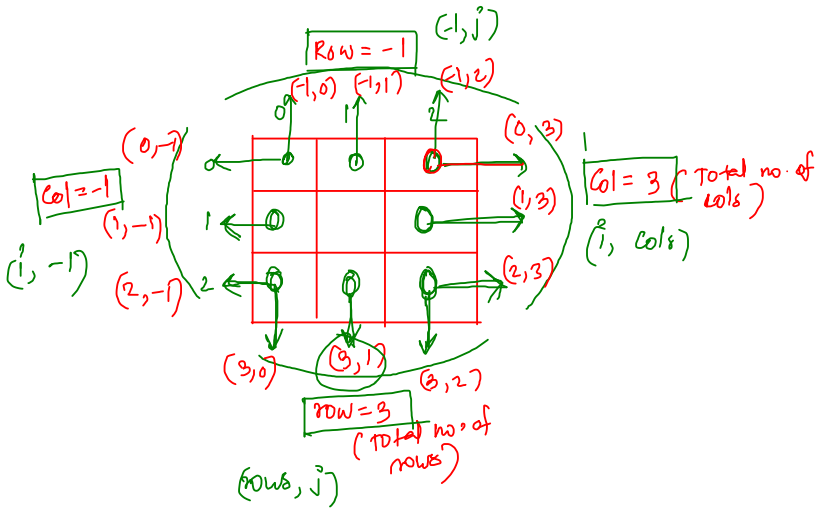
f = 0 1 2 3 4 (f = f + 1 will not work)

0

* cyclic pattern, °/o helpful

f = (f + 1) % 4 ;

★ How to check you are out of Matrix



① $(-1, j)$

last box ⇒ $(0, j)$

② $(i, cols)$

last box ⇒ $(i, cols-1)$

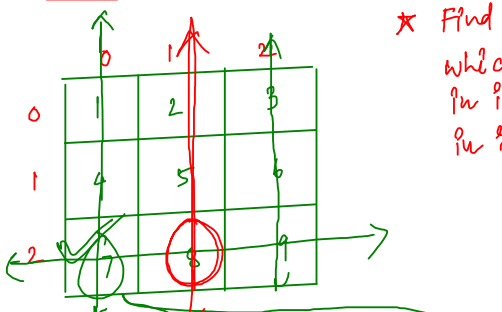
③ $(rows, j)$

last box ⇒ $(rows-1, j)$

④ $(i, -1)$

last box ⇒ $(i, 0)$

★ Maxima / Minima :



★ Find an element which is minimum in its Row and Maximum in the column

(i, j) → belongs to Row i, belongs to col j

1. goto every Element (i, j)
2. check mat[i][i] == min in Row(i) && mat[i][j] == max col(j)

mat(i)(j) == min Row(i) && mat(i)(j) == max col(j)

```
for (let i = 0; i < rows; i++) {
    for (let j = 0; j < cols; j++) {
        const minR = findMinRow(mat, i);      // O(cols)
        const maxC = findMaxCol(mat, j);      // O(rows)
        if (mat[i][j] == minR && mat[i][j] == maxC) {
            return mat[i][j];
        }
    }
}
return -1;
```

O(rows * cols)

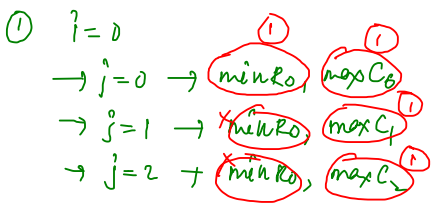
```
function findMinRow(mat, row) {
    /* Code */
}

function findMaxCol(mat, col) {
    /* Code */
}
```

TC : O(rows * cols * (row + cols)) ⇒ O(N * N * 2N)
⇒ O(N^3)

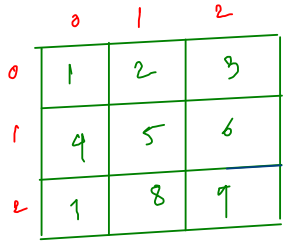
SC : O(1)

optimize / Improve :

3x3 matrix,

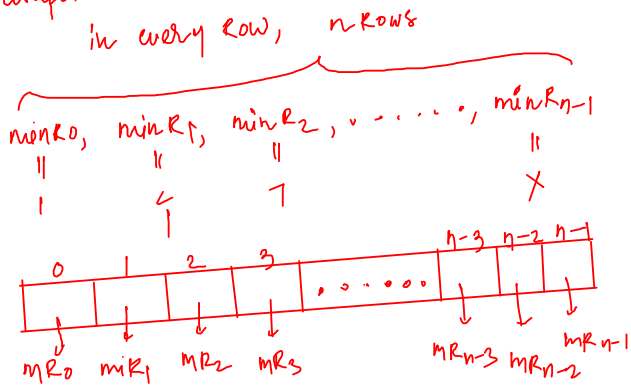


* we are calculating always which is redundant / duplicate, calculate once and store it some where.



② Compute all the maximas in every col (m-cols)

① Compute all the minimas in every Row, n Rows



findMinRow(mat, 2)

⇓

minR[2]

```
function maximaMinima(mat) {
  const rows = mat.length;
  const cols = mat[0].length;

  // 1. Precompute, calculate minR, maxC beforehand
  const minR = [];
  const maxC = [];
  for (let r = 0; r < rows; r++) {
    const minEle = findMinRow(mat, r);
    minR.push(minEle);
  }

  for (let c = 0; c < cols; c++) {
    const maxEle = findMaxCol(mat, c);
    maxC.push(maxEle);
  }

  for (let i = 0; i < rows; i++) {
    for (let j = 0; j < cols; j++) {
      if (mat[i][j] == minR[i] && mat[i][j] == maxC[j]) {
        return mat[i][j];
      }
    }
  }

  return -1;
}
```

O(rows * cols)

O(cols * rows)

⇒ O(rows * cols)

findMinRow → O(cols)

findMaxCol → O(rows)

* Precomputation,
(when you repetitive or redundant calculation, use this technique)

O(3 * rows * cols)

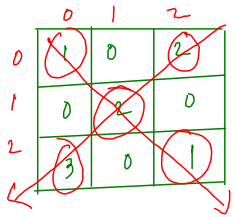
⇒ O(3N²)

⇒ O(N²) : TC

SC: O(rows + cols) ⇒ O(2N) ⇒ O(N).

↓ minR[] ↓ maxC[]

* Special Matrix :



① If the ele belongs to diag / adiag != 0

② all others ele = 0

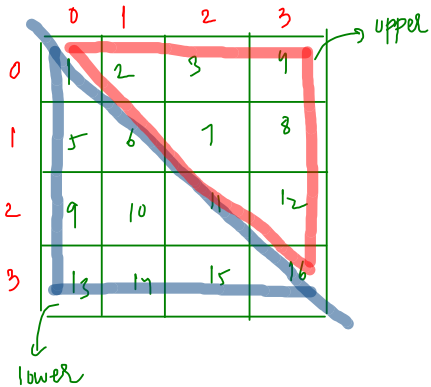
* given(r, c)
how do you know if is a diag/adiag ele ?

⇒ In diag, (r == c)
(0,0) (1,1) (2,2)

In adiag, (r) n-r-1)
c = n-r-1

```
for (let r = 0; r < rows; r++) {
    for (let c = 0; c < cols; c++) {
        if ( r == c || c == n-r-1 ) {
            if (mat[r][c] == 0) {
                return false;
            }
        }
        else {
            if (mat[r][c] != 0) {
                return false;
            }
        }
    }
}
return true;
```

★ upper ad lower triangular Sum :



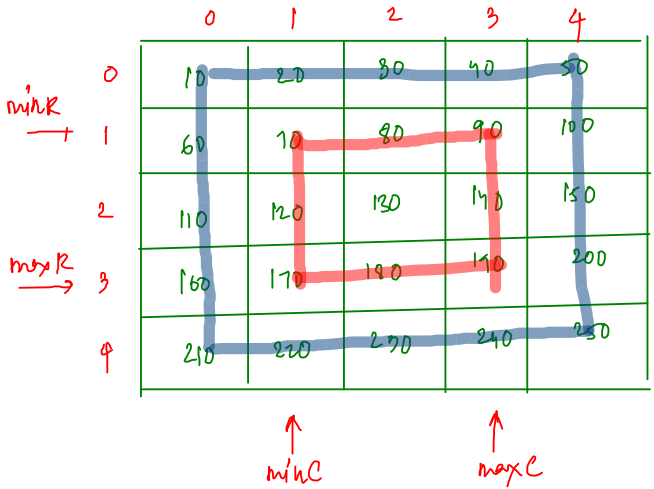
lower : (r, c) ⇒ (r >= c) ✓

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

upper : (r, c) ⇒ (r <= c) ✓

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

* Spiral Matrix 2 :



res []

① leftwall

```
for(let r = minR; r <= maxR; r++) {
    psw(mat[r][minC]);
}
minC++;
```

store in our res. push(mat(r)(minC))

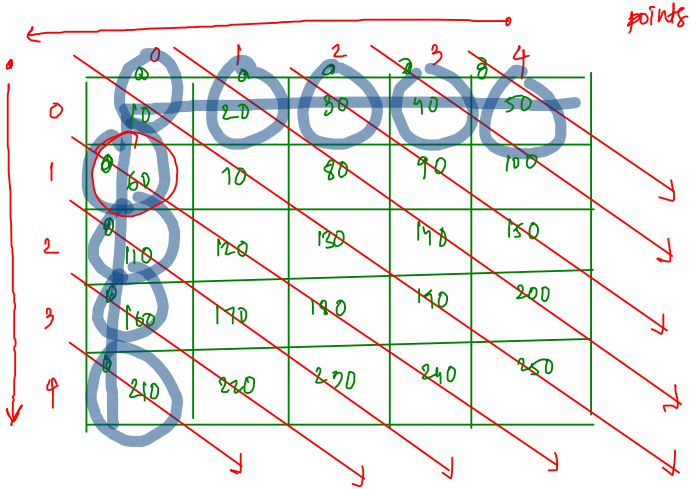
② bottomwall

```
for(let c = minC; c <= maxC; c++) {
    psw(mat[maxR][c]);
}
maxR--;
```

③ rightwall

④ top wall

★ Diagonal Traversal Different !



★ find the starting points and print every diagonal with starting point

start points

(0, 4)
(0, 3) (1, 4)
(0, 2) (1, 3) (2, 4)
(0, 1) (1, 2) (2, 3) (3, 4)
(0, 0) (1, 1) (2, 2) (3, 3) (4, 4)
(1, 0) (2, 1) (3, 2) (4, 3)

(0, 1) (+1, +1)
→ (1, 2) (+1, +1)
→ (2, 3) (+1, +1)
→ (3, 4)

• style diagonal traversal

(1, 0) → (2, 1) → (3, 2) → (4, 3)
(+1, +1) (+1, +1) (+1, +1)