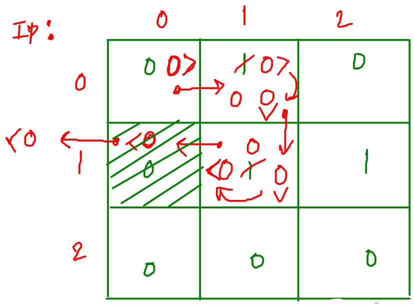
* Find the Way :

Ip :



* 0 → forward
1 → make it 0, turn right, move forward.

0> (right) = 0
0 with v below (down) = 1
<0 (left) = 2
^ above 0 (top) = 3

op : (1, 0)

the last cell before coming out of matrix

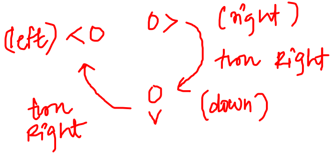
* maintain facing variable (0, 1, 2, 3)

(i, j) 0> — f=0 → (i, j+1) 0>

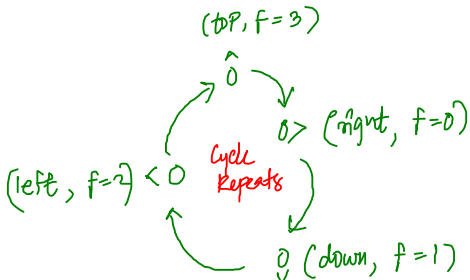
(i, j-1) <0 ← f=2 — (i, j) <0

0 with v below (i, j) — f=1 ↓ → 0 with v below (i+1, j)

^ above 0 (i-1, j) ↑ f=3 — ^ above 0 (i, j)



How to turn right :



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

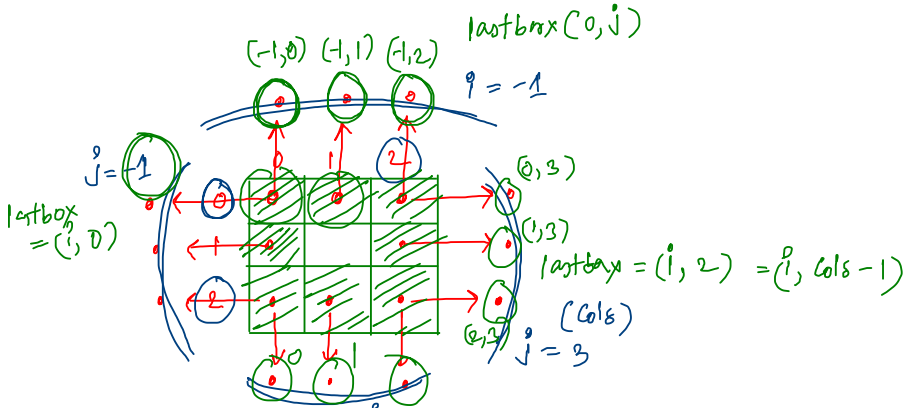
when, mat[i][j] == 1,

a. mat[i][j] = 0

b. turn right, f = f + 1, f = 0 1 2 3 4 5 6 → 0 1 2

★ f = (f + 1) % 4 f = 0 1 2 3 0

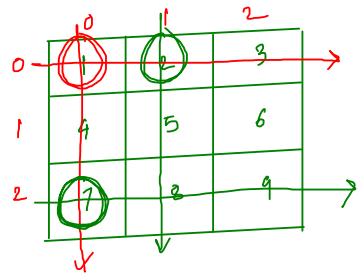
How to check you are out !



4 Boundaries

★ Maxima / minima :

Ip:



★ ele which minimum in its row and maximum in its col

Op: 7

$minR_0 = 1$

$maxC_0 = 7$

$1 == minR_0$ && $1 == maxC_0$

$minR_0 = 1$

$maxC_1 = 8$

$8 == minR_0$ && $8 == maxC_1$

$minR_2 = 7$

$maxC_0 = 7$

$7 == minR_2$ && $7 == maxC_0$

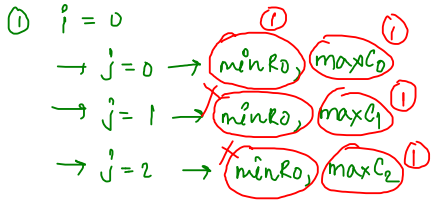
(yes)

★ TC : $O(rows * cols * (rows+cols)) = O(N^3)$, SC : $O(1)$

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

optimise :

3 x 3 matrix,



* we can use the calculated things, need not calculate multiple times.

① Compute min in each Row, and store

$minR_0 = 1$

$minR_1 = 4$

$minR_2 = 7$

If n Rows → n size Arr

② Compute max in each Col, and store

$maxC_0 = 7$

$maxC_1 = 8$

$maxC_2 = 9$

If m cols → m size Arr

③ Run the 1st discussed approach

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

PRE COMPUTATION TECHNIQUE

Calculate once use many times

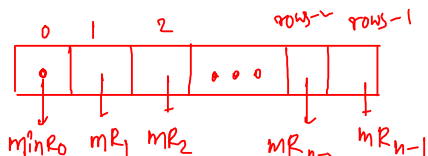
① $i = 0$

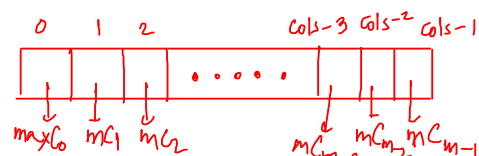
$j = 0 \rightarrow minR_0, maxC_0 \Rightarrow 1, 7 \Rightarrow == 1$

$j = 1 \rightarrow minR_0, maxC_1 \Rightarrow 1, 8 \Rightarrow == 2$

$j = 2 \rightarrow minR_0, maxC_2 \Rightarrow 1, 9 \Rightarrow == 3$

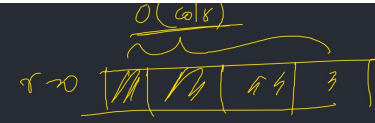
How to build table ?





Time/space:

```
1440  function maximaMinima(mat) {
1441    //Write code here
1442    const rows = mat.length;
1443    const cols = mat[0].length;
1444
1445    const minRow = [];
1446    const maxCol = [];
1447
1448    for (let r = 0; r < rows; r++) {
1449      const minEle = findMinRow(mat, r);
1450      minRow.push(minEle);
1451    }
1452
1453    for (let c = 0; c < cols; c++) {
1454      const maxEle = findMaxCol(mat, c);
1455      maxCol.push(maxEle);
1456    }
1457
1458    for (let r = 0; r < rows; r++) {
1459      for (let c = 0; c < cols; c++) {
1460        const minR = minRow[r];
1461        const maxC = maxCol[c];
1462        if (minR == mat[r][c] && maxC == mat[r][c]) {
1463          return mat[r][c];
1464        }
1465      }
1466    }
1467
1468    return -1;
1469  }
1470
```



2 extra arrays

→ O(rows)

→ O(cols) ⇒ O(r*c)

⇒ O(r+c)

+

+

r

→ O(cols)

→ O(rows)

→ O(rows)

→ O(cols)

⇒ O(r*c)

TC

O(3*r*c) → O(r*c)

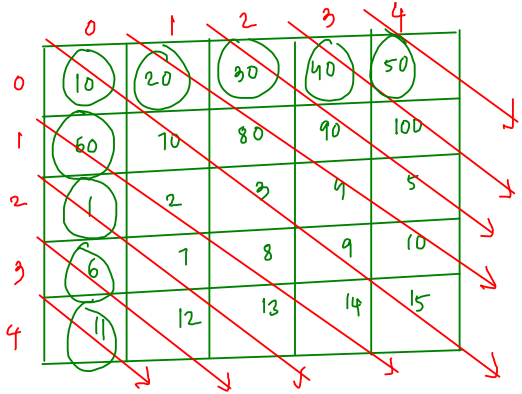
⇒ O(N²)

SC

= O(r+c)

= O(2N) ≈ O(N)

★ Diagonal traversal :



① starting point

② follow the diagonal (+1, +1)

op: 50, 40, 100, 80, 90, 5, 20, 80, 4, 10,
10, 70, 3, 9, 15, 60, 2, 8, 14, 1, 7, 13,
6, 12, 11