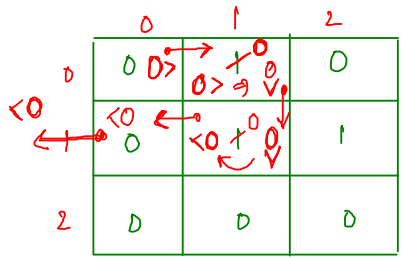
★ Find the way :



op : (1, 0)

last cell before coming out

★ 0 → move forward

1 → turn right
make it 0
move forward

(0, 0) facing = right

(0, 1) facing = right
Now turn right,
facing = down

(1, 1) facing = down
Now turn right,
facing = left

0> (right) = 0

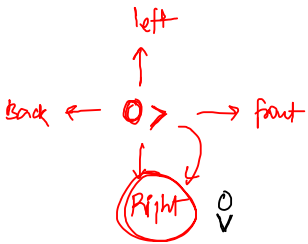
0 (down) = 1

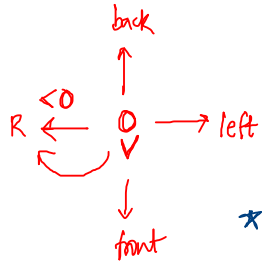
<0 (left) = 2

^0 (top) = 3

We need to know the facing of mouse in order to either move forward or turn right ?

⇒ maintain a variable facing = (0, 1, 2, 3)

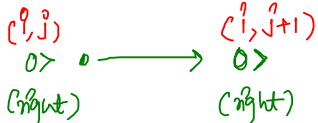




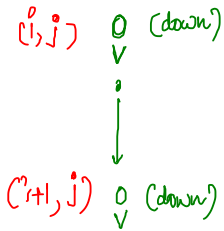
★ TC : O(N * M)

How to move forward using the facing !

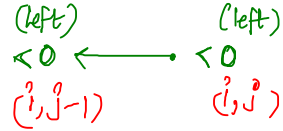
① facing = 0



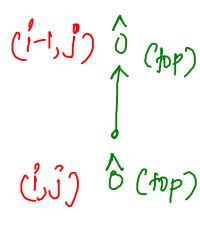
② facing = 1



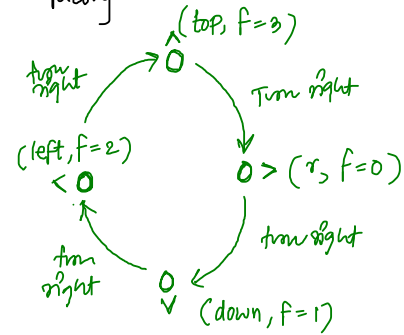
③ facing = 2



④ facing = 3



How to turn based on facing



① facing = 0 ⇒ facing = 1 $(0+1)\%4$

② facing = 1 ⇒ facing = 2 $(1+1)\%4$

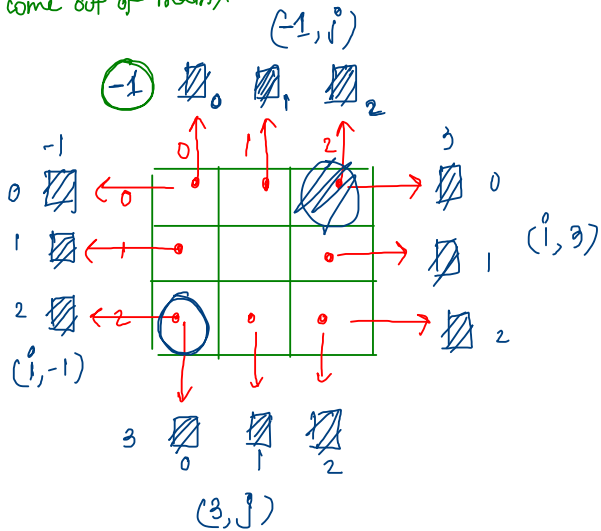
③ facing = 2 ⇒ facing = 3 $(2+1)\%4$

④ facing = 3 ⇒ facing = 0 $(3+1)\%4 \Rightarrow 4\%4$

★ $f = (f+1)\%4$

How to get the answer ?

⇒ you will know it when you come out of matrix



if (i == -1 || j == -1 || i == rows || j == cols)

⇒ exit cell

⇒ ans = just before cell to the exit cell

if (i == -1) (0, j)

if (j == cols) (i, cols - 1)

if (j == -1) (i, 0)

if (i == rows) (rows - 1, j)

★ Maxima Minima :

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

* Ele which is maximum in its col and also minimum in its row.

op: 7

minR0 = 1 (1) ✗
maxC0 = 7

1 is minimum in its row but not a maximum in its col.

minR2 = 7 (7) ✓
maxC0 = 7

minR0 = 1 (2) ✗
maxC1 = 8

```
for(let r=0; r<n; r++) {
  for(let c=0; c<n; c++) {
    const minR = findMinRow(matrix, r)
    const maxC = findMaxCol(matrix, c)
    if(m[r][c] == minR && m[r][c] == maxC)
      return m[r][c];
  }
}
return -1;
```

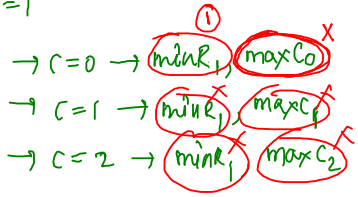
Optimization :

★ PRECOMPUTATION TECHNIQUE

3 × 3 matrix

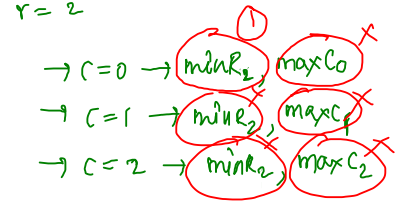
① r = 0

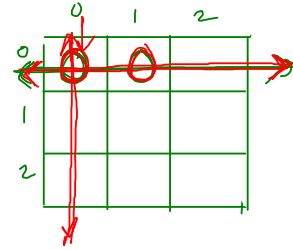
② r = 1



O(N³)

③ r = 2





★ ask once, remember it.

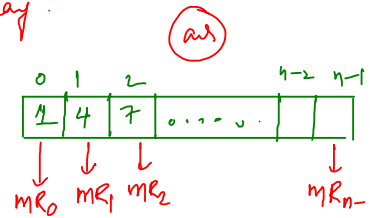
⇒ Find the required computation, store it in a desired datastructure

① Compute min in each row and store in a array.

$minR_0 = 1$

$minR_1 = 4$

$minR_2 = 7$



* arr[i] = minR[i]

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

minRow = ~~findMinRow(mat, r)~~

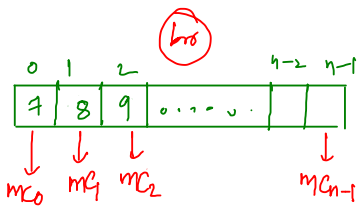
⇒ arr[0], arr[1]

② Compute max in each col and store in a array

$maxC_0 = 7$

$maxC_1 = 8$

$maxC_2 = 9$



maxCol = ~~findMaxCol(mat, c)~~

⇒ brr[0], brr[1]

brr[c]

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

  // 1) precomputation
  const minRow = [];
  for (let r = 0; r < rows; r++) {
    const val = findMinRow(mat, r);
    minRow.push(val);
  }

  const maxCol = [];
  for (let c = 0; c < cols; c++) {
    const val = findMaxCol(mat, c);
    maxCol.push(val);
  }

  // 2) your logic now uses stored computations
  for (let r = 0; r < rows; r++) {
    for (let c = 0; c < cols; c++) {
      const minR = minRow[r];
      const maxC = maxCol[c];
      if (mat[r][c] == minR && mat[r][c] == maxC) {
        return mat[r][c];
      }
    }
  }

  return -1;
}
```

O(findMinRow) → O(rows × cols)

+

O(findMaxCol) → O(cols × rows)

+

O(1) → O(rows × cols)

⇒ O(3 * rows * cols)

⇒ O(N²)

★ Array subtracting :

Eg: [1, 2, 3, 4]

[8, 6]

① diff = arr1[i] - arr2[j]
= 4 - 6 = -2

if (diff < 0) {
diff += 10 ⇒ -2+10 = 8
Carry = -1
}
res.push(diff)
i--; j--;

res = []
= [8]
= [8, 4]
= [8, 4, 1]
= (8, 4, 1, 1)
(1, 1, 4, 8)

② diff = arr1[i] - arr2[j] + Carry
= 3 - 8 + (-1)
= 3 - 8 - 1 = -6

diff < 0 ⇒ -6 + 10 = 4, Carry = -1
res.push(4)
i--; j--;

③ diff = arr1[i] + Carry
= 2 + (-1)
= 1
Carry = 0

④ diff = arr1[i] + Carry
= 1 + 0 = 1

Eg:

[1,2,3,4]
[5,6,7,8] ← Swap

1234 − 5678 = −4444

⇒

```
 5678
-1234
------
 4444
```

① Swap
② diffrence
③ multiply −1

① arr1 = [5,6,7,8]
arr2 = [1,2,3,4]

② apply prev logic ⇒ res = [4,4,4,4]

③ multiply −1 ⇒ res[0] = res[0] * −1
⇒ [−4,4,4,4]

How to decide arr1 < arr2 ?

① len(arr1) > len(arr2)
⇒ arr1 is larger
⇒ return false;

② len(arr1) < len(arr2)
⇒ arr2 is larger
⇒ return true;

③ (9)(1)(2)(4) → arr1
(9)(1)(3)(4) → arr2

find the 1st non equal elem
compare 2 < 3
arr1 < arr2 ⇒ true