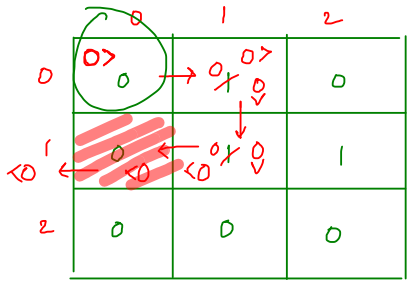
* Find the way :



Let,

0> (right) = 0

0 / v (down) = 1

<0 (left) = 2

^ / 0 (top) = 3

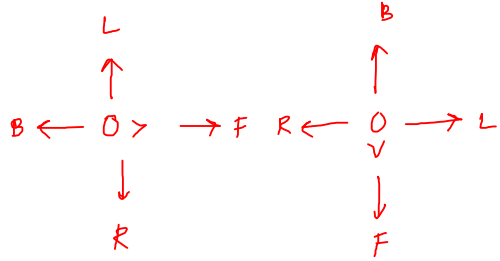
* 0 - forward

1 - make it 0, turn right, move forward

* facing

(i, j) f=0 (i, j+1) 0> ⟶ 0>

(i, j-1) f=2 (i, j) <0 ⟵ <0



(i, j) 0 v ⟶ f=1 ⟶ (i+1, j) 0 v

^ 0 (i-1, j) ⟵ f=3 ⟵ ^ 0 (i, j)

* How move forward looks like in various scenarios ?

```
if (facing == 0) {
    j++;
}
else if (facing == 1) {
    i++;
}
else if (facing == 2) {
    j--;
}
else {
    i--;
}
```

0> = 0 → turn right

0 = 1 ↙

★ How turn right looks like ?

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

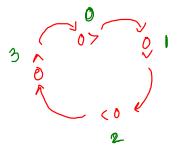
(cyclic pattern) (%)

<0 = 2 0 = 1

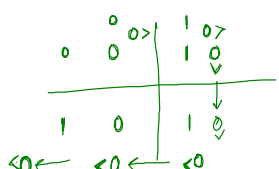
^0 = 3

<0 = 2

^0 = 3 0> = 0



★ facing = (facing + mat[i][j]) % 4



facing = 0

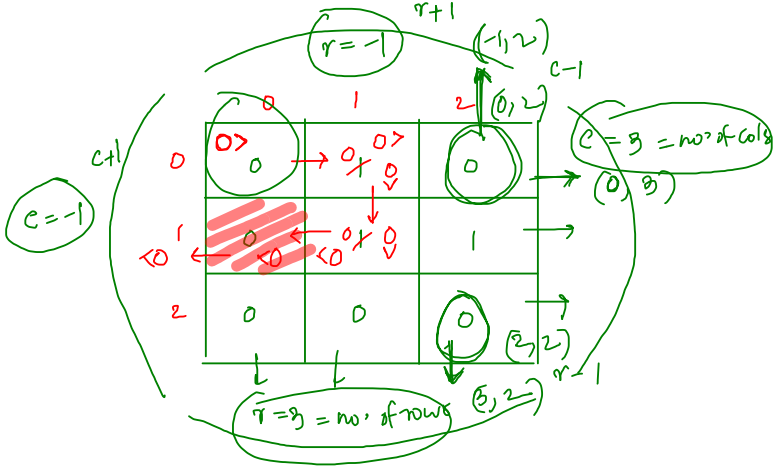
(0+0) % 4 = 0

(0+1) % 4 = 1 % 4 = 1

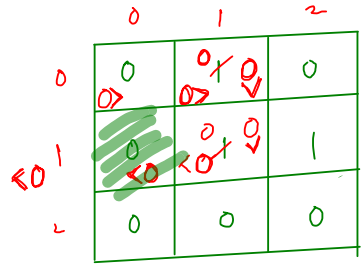
(1+1) % 4 = 2 % 4 = 2

(2+0) % 4 = 2

* how to decide exit ?



```
1234    while (true) {
1235      facing = (facing + matrix[i][j]) % 4;
1236
1237      if (matrix[i][j] == 1) {
1238        matrix[i][j] = 0;
1239      }
1240
1241      if (facing == 0) {
1242        j++;
1243      } else if (facing == 1) {
1244        i++;
1245      } else if (facing == 2) {
1246        j--;
1247      } else {
1248        i--;
1249      }
1250
1251      if (j == cols) {
1252        // exit from right
1253        return [i, j - 1];
1254      } else if (i == rows) {
1255        // exit from bottom
1256        return [i - 1, j];
1257      } else if (j == -1) {
1258        // exit from left
1259        return [i, j + 1];
1260      } else if (i == -1) {
1261        // exit from top
1262        return [i + 1, j];
1263      }
1264    }
1265  }
```



i = 0 1
j = 0 1 0 -1
facing = 0 1 2

9:20pm BREAK
≃ 9:35pm

① facing = (0+0)%4
= 0
j++ ⇒ j = 0 1

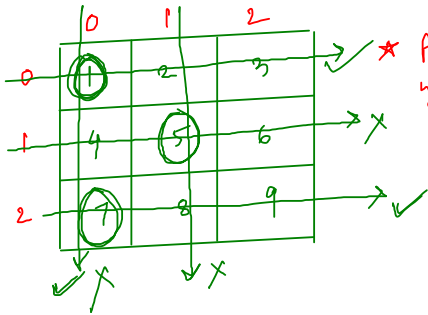
② facing = (0+1)%4
= 1
i++ ⇒ i = 0 1

③ facing = (1+1)%4
= 2
j-- ⇒ j = 1 0

④ facing = (2+0)%4
= 2
j-- ⇒ j = 0 -1

[i, j+1] ⇒ [1, -1+1] ⇒ [1, 0]

★ Maxima Minima :



★ find an element which is minimum in its Row and maximum in its Col.

$min\ R_0 = 1$ $min\ R_1 = 4$

$max\ C_0 = 7$ $max\ C_1 = 8$

$min\ R_2 = 7$

$max\ C_0 = 7$

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

★ TC : (rows * cols * (cols + rows))

: O(N * N * (N + N))

: O(2N³) : O(N³)

SC : O(1)

≠ optimize / Improve :

3 rows → find min in each row
3 cols → find max in each col
} store it

i = 0

→ j = 0 → minR0 (1), maxC0 (1)

→ j = 1 → minR0 (X), maxC1 (1)

→ j = 2 → minR0 (X), maxC2 (1)

i = 1

→ j = 0 → minR1 (1), maxC0 (X)

→ j = 1 → minR1 (X), maxC1 (X)

→ j = 2 → minR1 (X), maxC2 (X)

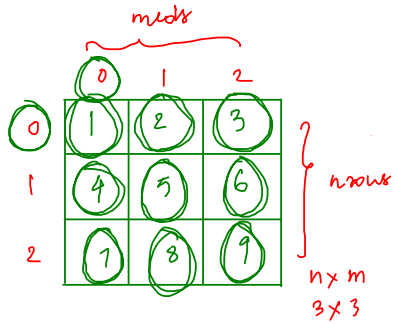
i = 2

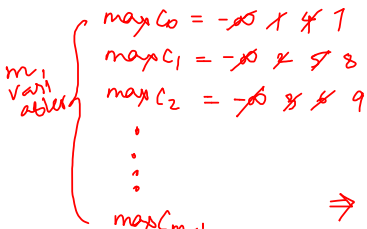
→ j = 0 → minR2 (1), maxC0 (X)

→ j = 1 → minR2 (X), maxC1 (X)

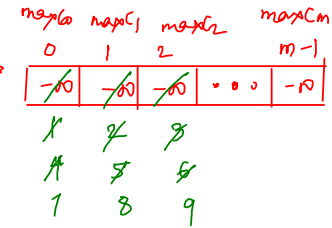
→ j = 2 → minR2 (X), maxC2 (X)

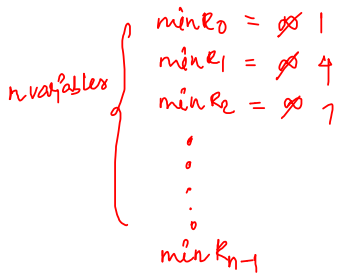
★ calculate once and store it somewhere so that you reuse it instead of calculating again & again.

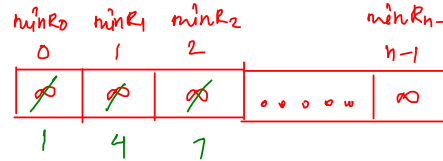




* precomputing
(calculating once, using it again & again)







* TC : precomputation + logic

: O(rows + cols + (rows * cols)) + O(rows * cols)

: O(rows * cols) : $O(N^2)$ ↓

* SC : (rows + cols) ↑

: O(N)

Imp methods / techniques / pattern ?

flag variable → store in flag → break → out of loop → yes / NO

digit iteration → %10 (last digit), /10 (remove last digit)

two pointer → reverse an array

rotation (cyclic) → use %

generating pair, triplets, subarray → template (code structure)

running stream → max/min/sum/prod at every point/idx of array

precomputation → store in your mind use later

}

1. what did you like the most from lectures ?
2. what did you hate the most overall ?
3. one / two words about the module ?