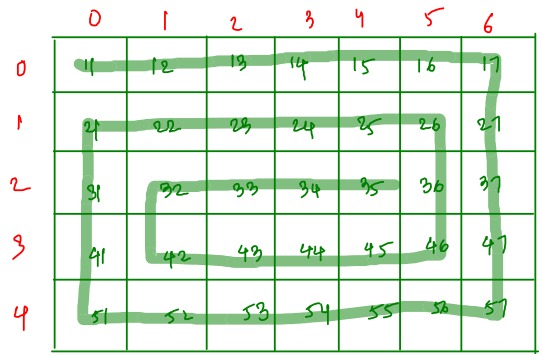
★ Spiral Traversal :



op; 11 12 13 14 15 16 17 27 37 47 57 56
55 54 53 52 51 41 31 21 22 23 24
25 26 36 46 45 44 43 42 32 33
34 35

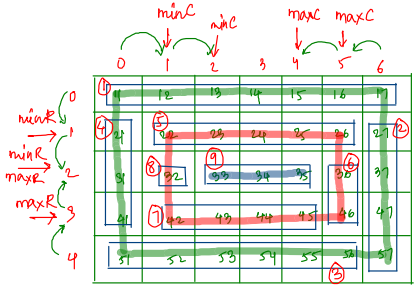
★ TC: How many times each element is visited/touched by your code.

⇒ each ele is visited once.

⇒ There are n*m elements

TC: (n*m*1) ← no. of times each ele visited.

TC: O(rows * cols)



① top wall → minR

② right wall → maxC

③ bottom wall → maxR

④ left wall → minC

① Top wall ((0,0), (0,1), (0,2), (0,3), (0,4), (0,5), (0,6))

[minR, c = minC to maxC]

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

(once the job of wall is completed move to next possible wall)

② Right wall ((1,6), (2,6), (3,6), (4,6))

[r = minR to maxR, maxC]

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

6--, (5)

③ Bottom Wall ((4,5) (4,4) (4,3) (4,2) (4,1) (4,0))

[maxR, c = maxC to minC]

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

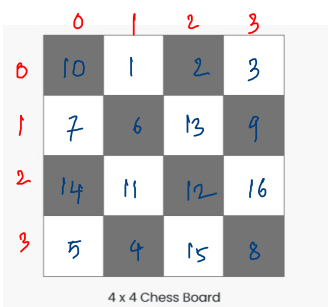
④ Left wall ((3,0) (2,0) (1,0))

[r = maxR to minR, minC]

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

* Repeat the boundary printing process until all the elements in the matrix are covered/touched/printed;

★ Alternate Matrix Sum :



1. Iteration on all elements of matrix (r, c)
2. you need to decide whether the current box (r, c) belongs to white square / black square

bsum = 0
wsum = 0

(0,0) → b ✓ bsum += mat[0][0]
→ w

(0,1) → b → w ✓ wsum += mat[0][1]

(0,2) → b ✓ bsum += mat[0][2]
→ w

(0,3) → b → w ✓ wsum += mat[0][3]

(r, c)
(r+c)
even → black
odd → white

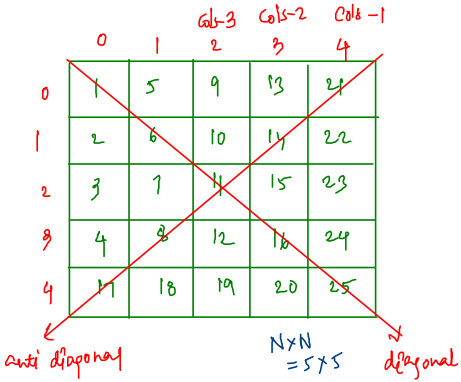
black squares sum
= 10 + 2 + 6 + 9 + 14 + 12 + 4 + 8
= 75

white square sum
= 1 + 3 + 7 + 13 + 11 + 16 + 5 + 15
= 71

} equal or not

black	white
(0,0) = 0	(0,1) = 1
(0,2) = 2	(0,3) = 3
(1,1) = 2	(1,0) = 1
(1,3) = 4	(1,2) = 3
(2,0) = 2	(2,1) = 3
(2,2) = 4	(2,3) = 5
(3,1) = 4	(3,0) = 3
(3,3) = 6	(3,2) = 5

★ Diagonal Difference :



$diag = 21 + 14 + 11 + 8 + 17 = X$

$adiag = 1 + 6 + 11 + 16 + 25 = Y$

$ans = abs(X - Y)$

diag

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

adiag

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

★ given R,

col in diag, $c = R$

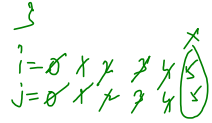
col in adiag, $c = Cols - R - 1$

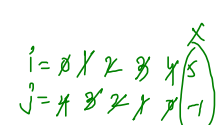
$5 - 0 - 1 = 4$

$5 - 1 - 1 = 3$

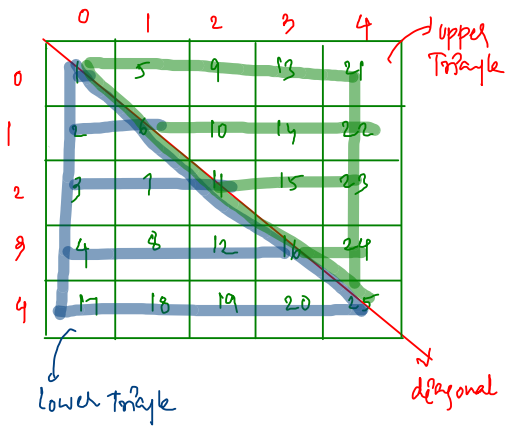
$5 - 2 - 1 = 2$

```
for (let r = 0; r < rows; r++) {
    diag += mat[r][r];
    adiag += mat[r][Cols - r - 1];
}
```





★ Sum of upper and lower triangle :



★ (r, c) → upperTriangle
→ lowerTriangle

How to decide ?

lsum = 0
usum = 0

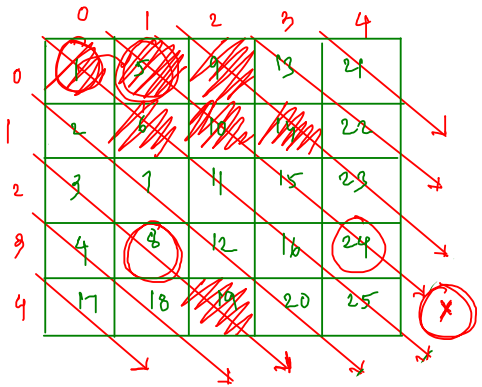
lowerTriangle (r >= c)

(0,0) (0>=0)
(1, 0) (1>=0)
(1, 1) (1>=1)
(2, 0) (2>=0)
(2, 1) (2>=1)
(2, 2) (2>=2)
(3, 0) (3>=0)
(3, 1) (3>=1)
(3, 2)
(3, 3)

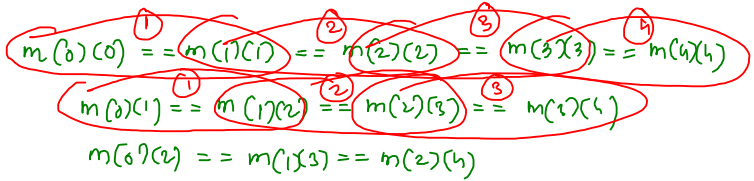
upperTriangle (r <= c)

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

* Toeplitz Matrix :



All elements in a same diagonal must be equal



r+1 < rows
c+1 < cols

(3, 4) 5<5X 4<5✓

(4, 5) out of Box

r+1, c+1 is valid

i = 0 1 2 3 4
j = 0 1 2 3 4

i = 0 1 2 3
j = 1 2 3 4

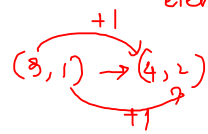
i = 0 1 2
j = 2 3 4

i = 1 2 3 4
j = 0 1 2 3

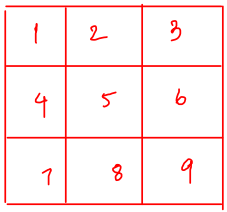
1. keep Iterating on all elements of matrix
2. At (r, c) check whether the next diagonal element is equal or not

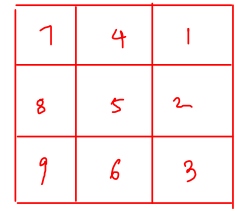
(0,0) → (1,1)
(0,1) → (1,2)
(0,2) → (1,3)

(r, c) ⇓ (r+1, c+1)

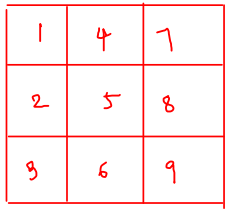


* Rotate Matrix by 90° :

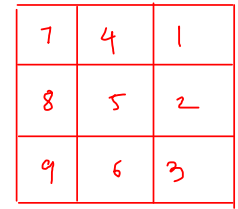




cols ↔ Rows | transpose ↓



reverse every row →



↑ 90°