Matrix 8

Thursday, 3 August 2023 8:42 AM

① Multidimensional Array in Python

(a) A 2D array is simply a list of lists in Python

→ An alternate way of traversal

```
arr = [[1,2,3] ,[4,5,6,7,8]]
for i in range(len(arr)):
    for j in range(len(arr[i])):
        print(arr[i][j], end=" ")
    print()
```

output :
1 2 3
4 5 6 7 8

→ User specified Dimensions

```
rows = 3
cols = 4
arr = [[0]*cols]*rows
arr[0][0] = 1
for i in arr:
    print(i)
```

output
[1,0,0,0]
[1,0,0,0]
[1,0,0,0]

(not a recommended way to create 2D array in Python)

```
print()
arr = [[0 for i in range(cols)] for j in range(rows)]
arr[0][0] = 1
for i in arr:
    print(i)
```

op:
[1, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]

② Matrix in Snake Pattern

up : 1 2 3 4 / 5 6 7 8 / 9 10 11 12

op : 1 2 3 4 8 7 6 5 9 10 11 12

```
def printSnake(mat):
    M = len(mat)
    N = len(mat[0])
    for i in range(M):
        if i%2==0:
            for j in range(N):
                print(mat[i][j], end=" ")
        else:
            for j in range(N-1,-1,-1):
                print(mat[i][j], end=" ")
```

θ(m×n)

③ Matrix Boundary Traversal

1 2 3 4 / 5 6 7 8 / 9 10 11 12 ⇒ 1 2 3 4 8 12 11 10 9 5

```
def BTraversal(mat):
    R = len(mat)
    C = len(mat[0])
    if R==1:
        for i in range(C):
            print(mat[0][i], end=" ")
    elif C==1:
        for i in range(R):
            print(mat[i][0], end=" ")
    else:
        for i in range(C):
            print(mat[0][i], end=" ")
        for i in range(R):
            print(mat[i][C-1], end=" ")
        for i in range(C-2,-1,-1):
            print(mat[R-1][i], end=" ")
        for i in range(R-2,0,-1):
            print(mat[i][0], end=" ")
```

θ(R+C)

④ Transpose of a Matrix

1 2 3 4 / 5 6 7 8 / 9 10 11 12 / 13 14 15 16 ⇒ 1 5 9 13 / 2 6 10 14 / 3 7 11 15 / 4 8 12 16

```
def transpose(mat):
    N = len(mat)
    temp = [[0]*N for i in range(N)]
    for i in range(N):
        for j in range(N):
            temp[i][j] = mat[j][i]
    for i in range(N):
        for j in range(N):
            mat[i][j] = temp[i][j]
```

Aux θ(n²) / θ(n²)

```
def transpose(mat):
    N = len(mat)
    for i in range(N):
        for j in range(i+1,N):
            mat[i][j], mat[j][i] = mat[j][i], mat[i][j]
```

→ In-place
→ One traversal

⑤ Rotate matrix anticlockwise by 90 degree

1 2 3 / 4 5 6 / 7 8 9 ⇒ 3 6 9 / 2 5 8 / 1 4 7

→ Last column becomes first row
→ Second last column becomes second row
and so on
→ First column becomes last row

```
def rotate(mat):
    N = len(mat)
    temp = [[0]*N for i in range(N)]
    for i in range(N):
        for j in range(N):
            temp[N-j-1][i] = mat[i][j]
    for i in range(N):
        for j in range(N):
            mat[i][j] = temp[i][j]
```

Aux θ(n²) / θ(n²)

θ(n²) time and θ(1) Aux space

1. Find transpose of matrix
2. Reverse individual columns

```
def rotate(mat):
    N = len(mat)
    for i in range(N):                          # Transpose
        for j in range(i+1,N):
            mat[i][j], mat[j][i] = mat[j][i], mat[i][j]
    for i in range(N):                          # Reverse columns
        low = 0
        high = N-1
        while low < high:
            mat[low][i], mat[high][i] = mat[high][i], mat[low][i]
            low += 1
            high -= 1
```

⑥ Spiral Traversal of matrix

1 2 3 4 / 5 6 7 8 / 9 10 11 12 / 13 14 15 16

1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

top → 1 2 3 4 5 / 6 7 8 9 10 / 11 12 13 14 15 / 16 17 18 19 20 ← bottom

right ↓ ; left ↑

top row
right column
bottom row (reverse)
left column (reverse)

```
def spiral mat(mat):
    R = len(mat)
    C = len(mat[0])
    top = 0; left = 0
    bottom = R-1; right = C-1
    while (top<=bottom and left<=right):
        for i in range(left,right+1):
            print(mat[top][i], end=" ")          # top row
        top += 1
        for i in range(top,bottom+1):
            print(mat[i][right], end=" ")        # right column
        right -= 1
        if (top<=bottom):
            for i in range(right,left-1,-1):
                print(mat[bottom][i], end=" ")   # bottom row in reverse order
            bottom -= 1
        if (left<=right):
            for i in range(bottom,top-1,-1):
                print(mat[i][left], end=" ")     # left column in reverse order
            left += 1
```

⑦ Search in a row and column wise sorted Matrix

Naive Approach
→ Traverse matelements one by one and print index if found else "Not found"

Efficient
→ Begin from the top right corner
→ If x is same, print position and return
→ If x is smaller, move left
→ If x is greater, move down

```
def search(mat,x):
    R = len(mat)
    C = len(mat[0])
    i = 0
    j = C-1
    while i<R and j>=0:
        if mat[i][j]==x:
            print("found at",i,j)
            return
        elif mat[i][j]>x:
            j = j-1
        else:
            i = i+1
    print("Not found")
```

θ(R+C)

⑧ Median of a row wise sorted matrix

Assumptions: (a) odd sized matrix (b) all distinct elements (to understand the logic)

Naive
1. Put all elements in an array θ(R×C)
2. Sort the array O(RC log RC)
3. Return the middle element of sorted array O(1)

Efficient — O(R log(max−min) × log C) (max element, min element)
→ Find the min element, mn
→ Find the max element, mx
→ Find the target position, tpos = (R×C+1)/2
→ Do binary search starting from (mn+mx)/2

```
def getMed(mat):
    R, C = len(mat), len(mat[0])
    mn, mx = mat[0][0], mat[0][C-1]
    for i in range(1,R):
        mn = min(mn, mat[i][0])
        mx = max(mx, mat[i][C-1])
    tpos = (R*C+1)//2
    while mn < mx:
        mid = (mn+mx)//2
        midpos = 0
        for i in range(R):
            midpos += bisect(mat[i],mid)
        if midpos < tpos:
            mn = mid+1
        else:
            mx = mid
    return mn
```