Advanced Data Structure

# 16. Matrix

Handwritten by pankaj kumar

# Matrix

Date
Page No. 230.

# Matrix

Date
Page No. 231.

## 1. Introduction :-

A matrix represents a collection of numbers arranged in order of rows and columns. it is necessary to enclose the elements of a matrix in parantheses or brackets.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

ex:- A matrix with 9 elements. have 3 rows & 3 columns, every element can be represented by its row & column no. Ex:- $a_{23} = 6$

order of matrix = no. of rows x no. of column
= 3x3 (in this example)

## (*) Transpose of a matrix :-

if $A = [a_{ij}]_{m \times n}$ :

then $A^T = [b_{ij}]_{n \times m}$, where $b_{ij} = a_{ji}$

ex:-
$$m = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad \text{then,} \quad m^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

Properties :-

- $(A^T)^T = A$
- $(A+B)^T = A^T + B^T$
- $(AB)^T = B^T A^T$

## (*) Properties of matrix addition and multiplication :-

1. $A+B = B+A$ (Commutative)
2. $(A+B)+C = A+(B+C)$ (Associative)
3. $AB \neq BA$ (Not Commutative)
4. $(AB)C = A(BC)$ (Associative)
5. $A(B+C) = AB+AC$ (Distributive)

## (*) Terminologies :-

- Square matrix :- no. of rows = no. of columns.
- Symmetric matrix :- $(A^T) = A$.

- Skew-Symmetric :- $A^T = -A$.
- Diagonal Matrix :- entries outside the main diagonal are all zeros.
- Identity Matrix :- all the elements of principal diagonal are ones and all other elements are zeros. denoted as $I$.
- Orthogonal matrix :- $AA^T = A^TA = I$
- Idempotent Matrix :- $A^2 = A$.
- Involutary Matrix :- $A^2 = I$.
- Singular matrix :- A square matrix A is said to be singular matrix if its determinant is zero i.e $|A| = 0$.
- Non Singular matrix :- A square matrix is said to be non-singular matrix if its determinant is non zero i.e $|A| \neq 0$.

Note :- Every square matrix can uniquely be expressed as the sum of a symmetric matrix and skew symmetric matrix.

$$A = \frac{1}{2}(A^T + A) + \frac{1}{2}(A - A^T)$$

Trace of a matrix :- trace of a matrix is denoted as $tr(A)$ which is used only for square matrix and equals the sum of the diagonal elements of the matrix. For example.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad tr(A) = 1+5+9 = 15$$

## (2) Matrix Implementation :-

- matrix in programming languages can be implemented using 2-D arrays / Two-dimensional arrays / Arrays of arrays.
- example of 2-D array with column 3 and row 3.

| | column 0 | column 1 | column 2 |
|---|---|---|---|
| row 0 | x[0][0] | x[0][1] | x[0][2] |
| row 1 | x[1][0] | x[1][1] | x[1][2] |
| row 2 | x[2][0] | x[2][1] | x[2][2] |

⇒

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

## ⊛ Declaration

```
data_type   array_name[size1][size2]
```

- data_type → type of data stored in array
- array_name → name of the array
- size1 → No. of rows
- size2 → No. of columns

size = size1 × size2

ex:-
```
int arr[2][5];
```

## ⊛ Accessing elements

arr[row_index][column_index]

ex:- arr[0][0]=1;

Note:- index will be from 0 to N-1 for size N of a array.

element present in first row & first column.

## ⊛ Printing all element

```
for (int i=0; i<N; i++){
    for (int j=0; j<M; j++){
        print arr[i][j];
    }
}
```

## ⊛ Searching an element

```
for (int i=0; i<N; i++){
    for (int j=0; j<m; j++){
        if (arr[i][j] == key){
            print (i, j);
            break;
        }
    }
}
```

## (3) Addition, Subtraction, multiplication of matrix :-

### ⊛ Matrices Addition:-

- The addition of two matrices Amxn and Bmxn gives Cmxn.

```
for i in 1 to m
    for j in 1 to n
        Cij = aij + bij
```

Note:-
- A+B = B+A
- A+(B+C) = (A+B)+C

TC: O(mxn)

## Matrices subtraction :-

$A_{m*n} - B_{m*n} = C_{m*n}$

```
for i in 1 to m
    for j in 1 to n
        cij = aij - bij
```

TC: O(m*n)

**Note :-**
- A - B ≠ B - A
- A - (B - C) ≠ (A - B) - C

## Matrices multiplication :-

$A_{m*n} \times B_{n*p} = C_{m*p}$

ex:-

$$\begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 5\times1+6\times4 & 5\times2+6\times5 \\ 8\times1+9\times4 & 8\times2+9\times5 \end{bmatrix}$$

$$= \begin{bmatrix} 29 & 40 \\ 44 & 61 \end{bmatrix}$$

Algorithm :-

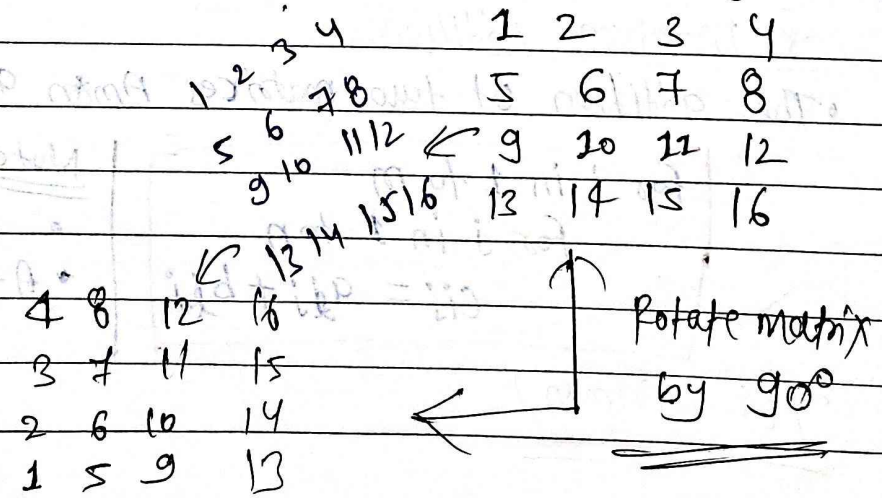
```
for i in 1 to m
    for j in 1 to p
        cij = 0
        for k in 1 to n
            cij += aik * bkj
```

**Note :-**
- A * B ≠ B * A
- A * (B * C) = (A * B) * C

TC : O(m*n*p)

## (4) Matrix Rotation :-

Given a square matrix of dimension N*N. the task is to rotate the matrix in anti-clockwise direction by 90 degrees.



## (*) Observation

- First row of destination (4, 8, 12, 16) = last column of source
- second row of destination (3, 7, 11, 15) = last second column of src.
- :
- last row of destination (1, 5, 9, 13) = First column of source

∴ Rotating a matrix in anti-clockwise direction by 90° is equivalent to replacing row from top of the matrix by column from the end.

## (*) Implementation using extra space :-

original matrix = mat [N] [N]
temporary matrix = temp [N] [N]

```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        temp[i][j] = mat[j][N-i-1];
```

## (*) Without using extra space: (In-place Rotation):-

- An NxN matrix will have Floor (N/2) square cycles.
- For example, a 4x4 matrix will have 2 cycles. the first cycle is formed by its 1st row, last column, last row and First column. The second cycle is formed by 2nd row, second-last column, second-last row and 2nd column.
- The idea is for each square cycle, we swap the elements involved with the corresponding cell in the matrix in anti-clockwise direction. i.e From top to left, left to bottom, bottom to right, and from right to top one at a time.

Steps :-

First cycle with red elements.

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

- moving First group of four elements (1st elements of 1st row, last row, 1st column & last column) of First cycle in anti-clockwise.

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

- moving next group of four elements of First cycle in anti-clockwise.

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

- moving final group of four elements of first cycles in anti-clockwise.

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

Second cycle:-

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

fixing → Second cycle.

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

Code:-

```
for(int i=0; i<N/2; i++){  //consider all square one by one.
    for(int j=i; j<N-i-1; j++){  //consider elements in group of
                                 //4 in current squares.
        int temp = mat[i][j];                          // Store current cell in temp variable
        mat[i][j] = mat[j][N-1-i];                     // move value from right to top
        mat[j][N-1-i] = mat[N-1-i][N-1-j];             // move value from bottom to right
        mat[N-1-i][N-1-j] = mat[N-1-j][i];             // move value from left to bottom
        mat[N-1-j][i] = temp;  // assign temp to left
    }
}
```

## (5) Transpose of a Matrix:-

1 2 3
4 5 6
7 8 9

→ Transpose

1 4 7
2 5 8
3 6 9

Pseudo code :-

```
void transpose (A[n][n]){
    B[n][n] //Transpose matrix
    for (i=0 to n-1)
    {
        for (j=0 to n-1)
            B[i][j] = A[j][i]
    }
}
// TC: O(n*n)
// SC: O(n*n)
```

(6) Search Element in Row-wise and column-wise sorted matrix.

Soln :- idea is to solve problem with row and column elimination, reducing search space.

There are three possible cases :-

(1) all no. in a row will be smaller than the key if the right most of current row element is smaller than key.

(2) The no. we are searching for is smaller than the current no. This will ensure, that all the elements in the current column get eliminated and we continue our search on the previous column.

(3) The no. we are searching is equal to current no. this will end our search.

Pseudo code :-

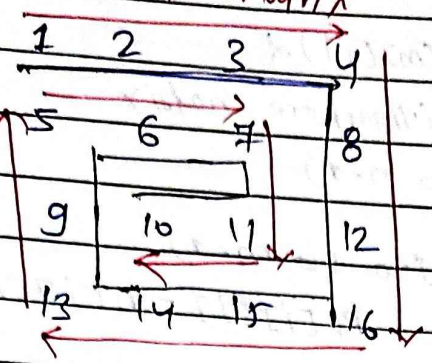
```
void search (mat[][], int x){
    i=0, j=n-1
    while (i<n && j>=0){
        if (mat[i][j]==x){
            print (i,j)
            break
        }
        else if (mat[i][j]>x){
            j--;
        }
        else
            i++;
    }
}
```

// TC: O(n)
// SC: O(1)

Date

Page No. 238.

## (7) Spiral Traversal of matrix




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

**Solution:-** we will solve it with the help of 5 variable

- k → starting row index
- m – ending row index
- l – starting column index
- n – ending column index
- i – iterator

**Pseudo code:-**

```
void print_spiral (A[][], m, n) {
    k=0, l=0
    while (k<m && l<n) {
        for (i=l to n-1)
            print (A[k][i])          (Print the first row from the remaining rows)
        k++;
        for (i=k to i=m-1)
            print (A[i][n-1])        (Print the last column from the remaining column)
        n--;
        if (k<m) {
            for (i=n-1; i>=l; --i)
                print (A[m-1][i])    (Print the last row from the remaining rows)
            m--
        }
        if (l<n) {
            for (i=m-1; i>=k; --i)
                print (A[i][l])      (Print the first column from the remaining column)
            l++
        }
    }
}
```