Lesson:

2D Arrays

Pre-Requisites:

- JAVA syntax, for loops
- 1D arrays

List of concepts involved:

- Multidimensional arrays
- 2D arrays addressing
- Rotation of 2D matrix
- Prefix sum concept in 2D matrix.

Multidimensional array:

Array of arrays is known as multidimensional arrays.

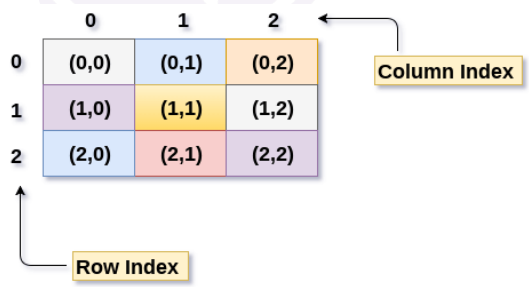
Syntax to declare a N Dimensional array:

data_type[1st dimension][2nd dimension][]..[Nth dimension] array_name = new data_type[size1][size2]....[sizeN];

Syntax to declare a 2Dimensional array of type int:
int[][] arr = new int[rows][column];
where rows imply the number of rows needed for the 2D array and

column implies the number of columns needed.
int[][] arr = new int[4][5];

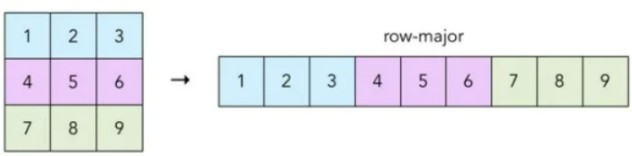
Here, arr is a two-dimensional array. It can hold a maximum of 20 elements of integer type.
We can think of this array as a table with 4 rows and each row has 5 columns as shown below.



Addressing in 2D array:

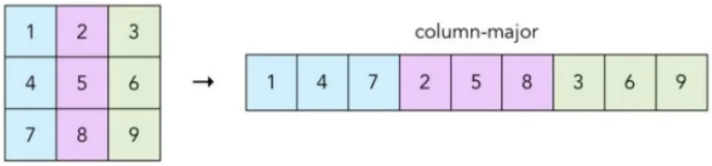
There are two main techniques of storing 2D array elements into memory:

1. Row Major ordering
In row major ordering, all the rows of the 2D array are stored into the memory contiguously. Considering the array shown in the above image, its memory allocation according to row major order is shown as follows.



First, the 1st row of the array is stored into the memory completely, then the 2nd row of the array is stored into the memory completely and so on till the last row.

2. Column Major ordering
According to the column major ordering, all the columns of the 2D array are stored into the memory contiguouslv. The memorv allocation of the arrav which is shown in the above image is given as follows.



There are two different formulas to calculate the address of a random element of

the 2D array. One is by row major ordering and second is by column major ordering.

- **By Row Major Order**

If array is declared by a[m][n] where m is the number of rows while n is the number of columns, then address of an element a[i][j] of the array stored in row major order is calculated as,
Address of A [I][J] = B + W * [N * (I – Lr) + (J – Lc)]
Where,
B = Base address
I = Row subscript of element whose address is to be found
J = Column subscript of element whose address is to be found
W = Storage Size of one element stored in the array (in byte)
Lr = Lower limit of row/start row index of matrix, if not given assume 0 (zero)
Lc = Lower limit of column/start column index of matrix, if not given assume 0 (zero)
M = Number of row of the given matrix
N = Number of column of the given matrix

Q1 : Given a 2D array A[5.....11, 1.....40], whose base address(BA) = 1980, size of an element = 4 bytes . Find the location of cell a[5][12].
Solution: Number or rows say, M = (Ur – Lr) + 1 = [11 – 5] +1 = 7
Number or columns say, N = (Uc – Lc) + 1 = [40 – 1] +1 = 40

Row Major Wise Calculation of above equation
The given values are: B = 1980, W = 4 bytes, I = 5, J = 12, Lr = 5, Lc = 11, N = 40
Address of A [I][J] = B + W * [N * (I – Lr) + (J – Lc)]
= 1980+ 4* [40 * (5 – 5) + (12 – 11)] = 1980 + 4 * [1] = 1984

By Column major order

If array is declared by a[m][n] where m is the number of rows while n is the number of columns, then address of an element a[i][j] of the array stored in row major order is calculated as,
Address of A [I][J] Column Major Wise = B + W * [(I – Lr) + M * (J – Lc)]
Where,
B = Base address
I = Row subscript of element whose address is to be found
J = Column subscript of element whose address is to be found

W = Storage Size of one element stored in the array (in byte)
Lr = Lower limit of row/start row index of matrix, if not given assume 0 (zero)
Lc = Lower limit of column/start column index of matrix, if not given assume 0 (zero)
M = Number of row of the given matrix
N = Number of column of the given matrix
Important : Usually number of rows and columns of a matrix are given (like A[20][30] or A[40][60]) but if it is given as A[Lr- – – – – Ur, Lc- – – – – Uc]. In this case number of rows and columns are calculated using the following methods:
Number of rows (M) will be calculated as = (Ur – Lr) + 1
Number of columns (N) will be calculated as = (Uc – Lc) + 1

Q2 :Given A[10][20], requires one byte of storage. If the beginning location is 1500 determine the location of A[15][20].
The given values are: B = 1500, W = 1 byte, I = 15, J = 20, Lr = -15, Lc = 15, M = 26
Address of A [I][J] = B + W * [(I – Lr) + M * (J – Lc)]
= 1500 + 1 * [(15 – (-15)) + 26 * (20 – 15)] = 1500 + 1 * [30 + 26 * 5] = 1500 + 1 * [160] = 1660.

Looping through 2D arrays:
Suppose you want to store 10 at every index of a 2D array of dimensions 4 X 5, you can do so using the following code:

```
for (int i = 0; i < 4; i++) {
   for (int j = 0; j < 5; j++) {
            arr[i][j] = 10;
        }
}
```

Taking 2D array as input from the user:

Following code will show how we can take a 2D array as input from the user.

LP_CODE1.Java

```
Enter the no of rows :
3
Enter the number of columns :
4
Please enter 12 elements nows.
1 2 3 4
4 5 6 7
5 6 7 8
The Input array is :
1       2       3       4
4       5       6       7
5       6       7       8
```

Rotation of the matrix :

Q3. Write a program to rotate a given matrix by 90 degrees in clockwise direction.

Input :
1 2 3
4 5 6
7 8 9

Output :

7 4 1
8 5 2
9 6 3

Solution:

Code : LP_Code1.java

Approach :

- The rotation of a matrix involves two steps:
- First, find the transpose of the given matrix.
- Swap the elements of the first column with the last column (if the matrix is of 3*3). The second column remains the same.

- Note: Matrix must have the same number of rows and columns.

- Let's understand through an example. Suppose, the matrix is:

$$\begin{bmatrix} 5 & 9 & 10 \\ 12 & 23 & 34 \\ 45 & 8 & 19 \end{bmatrix}$$

Let's find the transpose of the matrix.

$$\begin{bmatrix} 5 & 12 & 45 \\ 9 & 23 & 8 \\ 10 & 34 & 19 \end{bmatrix}$$

- To get the rotated matrix, swap the first column with the last column.

$$\begin{bmatrix} 45 & 12 & 5 \\ 8 & 23 & 9 \\ 19 & 34 & 10 \end{bmatrix}$$

The above matrix is rotated by 90 degrees.

- If the given matrix is 4*4 matrix, swap the first column with the last column and the second column with the third column. For example, consider the following figure.

This is an in place rotation and we have not used any extra space.
Therefore ,

Time complexity : O(n*n) where n = number of rows in the matrix.
Space complexity: O(1) or constant space.

Prefix sum Concept :

Q4 : Given a matrix and a couple of coordinate pairs (x1 , y1) and (x2 , y2) respectively. Return the sum of the rectangle formed using these coordinates as opposite corners.

Input :
Arr[][] = [
1 2 3 4
5 6 7 8
3 7 6 4
0 8 9 1
]
x1 = 0 , y1 = 1 , x2 = 3 , y2 = 2

Output : 48

Explanation:
1 2 3 4
5 6 7 8
3 7 6 4
0 8 9 1
The formed rectangle is shown with red color.

Solution :

LP_Code2.java

Output :

```
enter the number of rows : 4
enter the number of column : 4
enter the  matrix element :
1 2 3 4
5 6 7 8
3 7 6 4
0 8 9 1
enter the value of l1 coordinate :0
enter the value of r1 coordinate :1
enter the value of l2 coordinate :3
enter the value of r2 coordinate :2
The sum of the elements of given rectangle is : 48
```

Approach :
- We have simply traversed the array from (x1 , y1) to (x2 , y2) coordinate and added the sum.

Time complexity : O(n*m) where n = number of rows in the matrix
And m = number of columns in the matrix

Space complexity : O(1) since we have not used any extra space.

But if multiple queries with different sets of coordinates are given then this approach is not efficient.
We cannot calculate the sum for every query. Rather we have to do things somewhat in a smart manner.

There comes the concept of prefix sum.

For this, we first need to calculate the prefix sum array for the matrix.

Something like this :

Prefix sum is the cumulative sum of the matrix.
Prefix_sum[i][j] = arr[0][0] + arr[0][1] + arr[0][2] + . . . + arr[i][j]

Prefix Sum of matrix with each cell=1

1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16

Now, let's say we want to find the sum of following region:

1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16

Region(Answer)
So we first need the sum from each of the following regions:

Region(A)

1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16

Region(B)

1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16

Region(C)

1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16

Region(D)

1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16

Then we calculate the following for required answer:

Region(Answer) = Region(A) - Region(B) - Region(C) + Region(D)

LP_Code3.java

Output :

```
enter the number of rows : 4
enter the number of column : 4
enter the  matrix element :
1 2 3 4
5 6 7 8
3 7 6 4
0 8 9 1
enter the value of l1 coordinate :0
enter the value of r1 coordinate :1
enter the value of l2 coordinate :3
enter the value of r2 coordinate :2
The sum of the elements of given rectangle is : 48
```

Time complexity : currently this code is also consuming O(n*m) time. But when multiple queries are there this will take O(1) operations to return the sum once the prefix sum matrix is calculated.

Space complexity : O(n*m) because we have constructed a new matrix of n*m dimensions.

Note: This problem can be solved in place only i.e. without constructing a new array. We can just take the input and create the prefix sum there only. and then use the updated array to find the region sum. First we can take the vertical sum of complete array, then we can take the horizontal sum of complete array.