DATA STRUCTURES

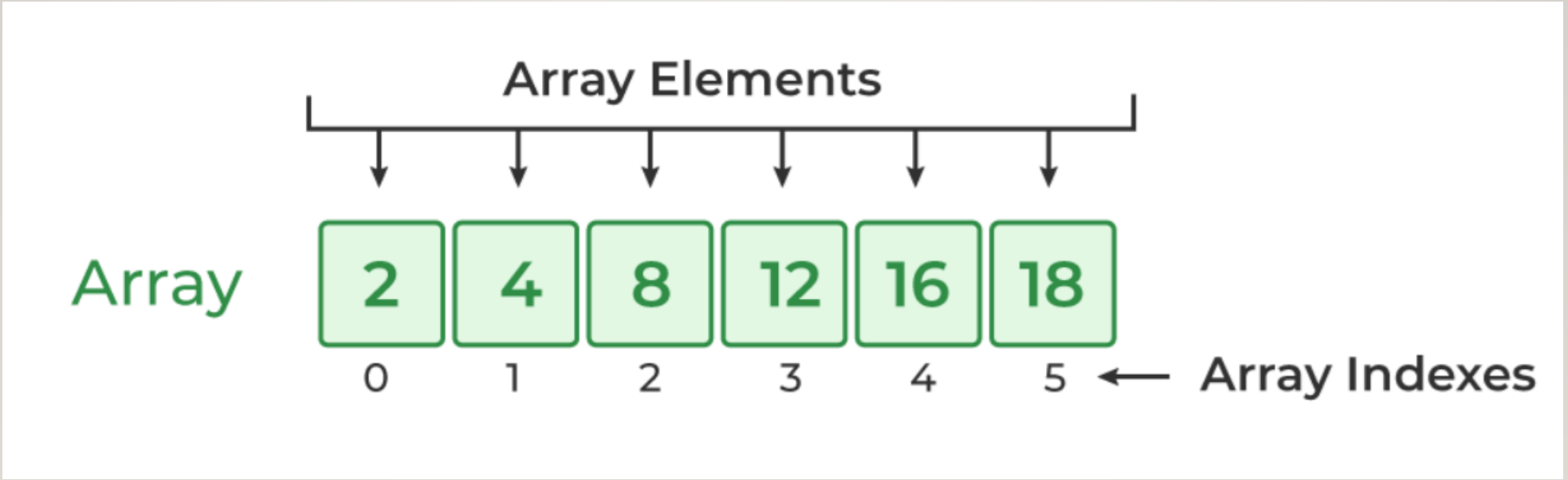
LECTURE-4

ARRAYS

Dr. Sumitra Kisan

One-Dimensional Arrays

- A one-dimensional array can be viewed as a linear sequence of elements.
- Only a single row exists in the one-dimensional array and every element within the array is accessible by the index.
- In C, array indexing starts zero-indexing i.e. the first element is at index 0, the second at index 1, and so on up to n-1 for an array of size n.



Memory Representation of One Dimensional Array

- Memory for an array is allocated in a contiguous manner i.e. all the elements are stored adjacent to its previous and next element in the memory.
- It means that, if the address of the first element and the type of the array is known, the address of any subsequent element can be calculated arithmetically.

To find the address of an element in an array the following formula is used:

$$Address\ of\ A[Index] = B + W * (Index - LB)$$

Where:

- ***Index*** *= The index of the element whose address is to be found (not the value of the element).*
- ***B*** *= Base address of the array.*
- ***W*** *= Storage size of one element in bytes.*
- ***LB*** *= Lower bound of the index (if not specified, assume zero).*

Example: Given the base address of an array **A[1300 …… 1900]** as **1020** and the size of each element is 2 bytes in the memory, find the address of **A[1700].**

Solution:

Given:

Base address (B) = 1020
Lower bound (LB) = 1300
Size of each element (W) = 2 bytes
Index of element (not value) = 1700

Address of A[Index] = B + W * (Index – LB)
Address of A[1700] = 1020 + 2 * (1700 – 1300)
= 1020 + 2 * (400)
= 1020 + 800
Address of A[1700] = 1820

Two-Dimensional Array

We can visualize a two-dimensional array as an array of one-dimensional arrays arranged one over another forming a table with ‘x’ rows and ‘y’ columns where the row number ranges from 0 to (x-1) and the column number ranges from 0 to (y-1).

	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]

Memory Representation of Two Dimensional Array

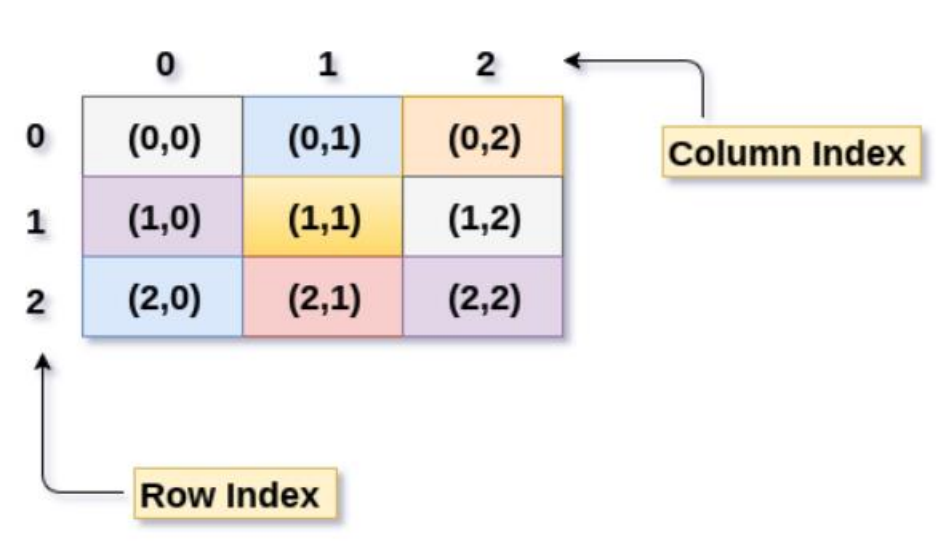
The elements of the 2-D array have to be stored contiguously in memory. As the computers have linear memory addresses, the 2-D arrays must be linearized so as to enable their storage. There are two ways to achieve linearization of array elements:

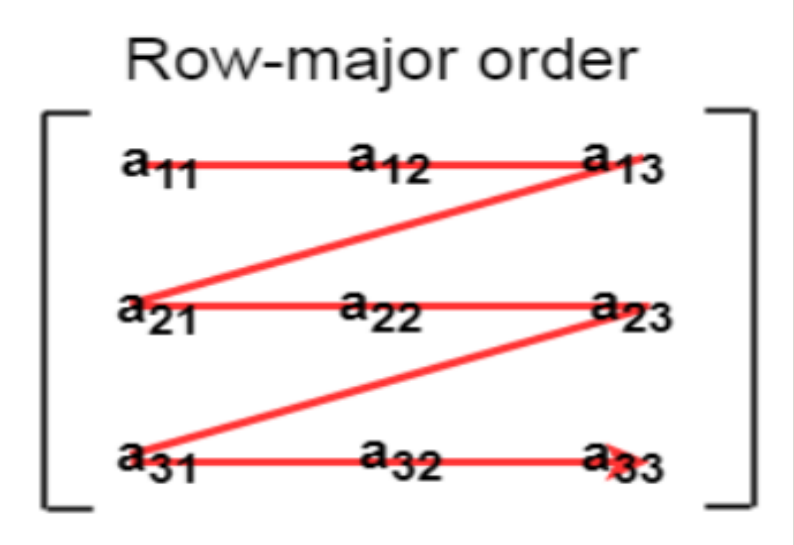
Row-major- This linearization technique stores firstly the first row of the array, then the second row of the array, then the third row, and so on. (i.e. elements are stored row-wise.)

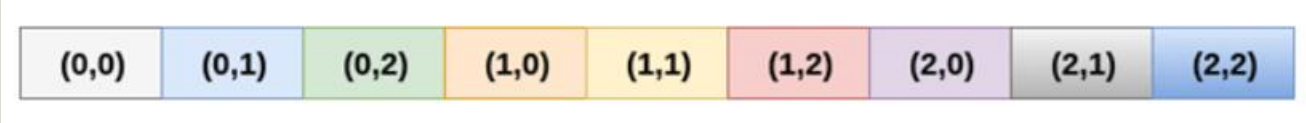
Column-major– This linearization technique stores first the first column, then the second column, then the third column, and so on i.e. (elements are stored column-wise.)

Row Major ordering

In row major ordering, all the rows of the 2D array are stored into the memory contiguously.







To find the address of the element using row-major order uses the following formula:

$$Address\ of\ A[I][J] = B + W * ((I - LR) * N + (J - LC))$$

I = Row Subset of an element whose address to be found,
J = Column Subset of an element whose address to be found,
B = Base address,
W = Storage size of one element store in an array(in byte),
LR = Lower Limit of row/start row index of the matrix(If not given assume it as zero),
LC = Lower Limit of column/start column index of the matrix(If not given assume it as zero),
N = Number of column given in the matrix.

Example: Given an array, **arr[1……10][1……15]** with base value **100** and the size of each element is **1 Byte** in memory. Find the address of **arr[8][6]** with the help of row-major order.

Given:

Base address B = 100
Storage size of one element store in any array W = 1 Bytes
Row Subset of an element whose address to be found I = 8
Column Subset of an element whose address to be found J = 6
Lower Limit of row/start row index of matrix LR = 1
Lower Limit of column/start column index of matrix = 1
Number of column given in the matrix N = Upper Bound – Lower Bound + 1
= 15 – 1 + 1
= 15

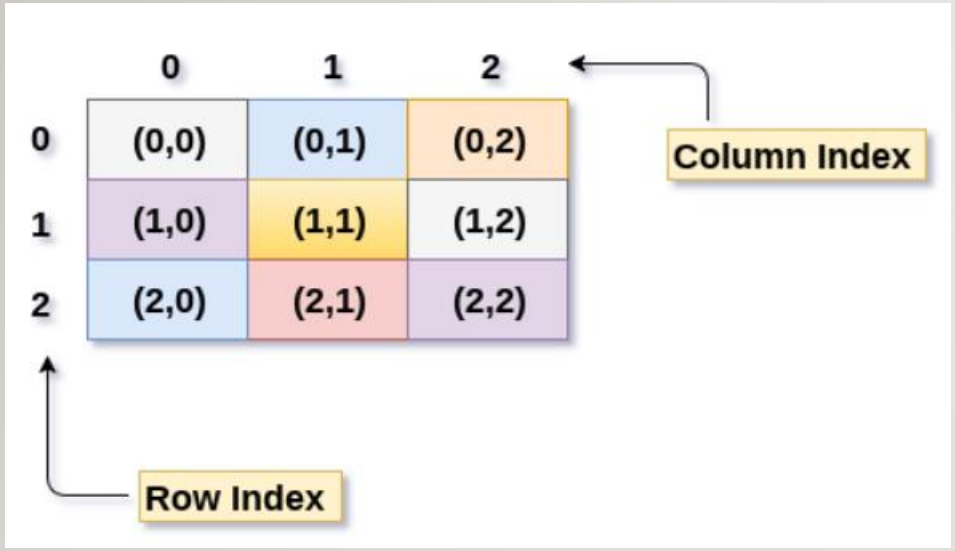
Address of A[I][J] = B + W * ((I – LR) * N + (J – LC))

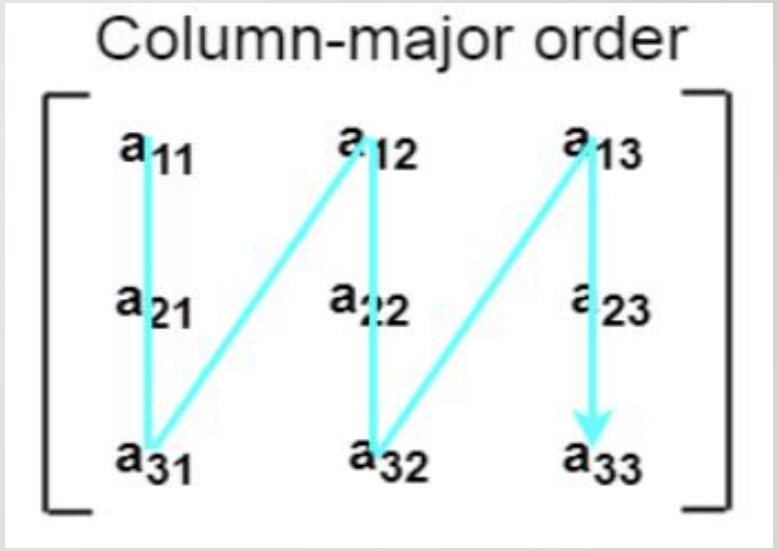
Solution:

*Address of A[8][6] = 100 + 1 * ((8 – 1) * 15 + (6 – 1))*
*= 100 + 1 * ((7) * 15 + (5))*
*= 100 + 1 * (110)*
Address of A[I][J] = 210

Column Major ordering

All the columns of the 2D array are stored into the memory contiguously.





To find the address of the element using column-major order use the following formula:

Address of A[I][J] = B + W * ((J – LC) * M + (I – LR))

I = Row Subset of an element whose address to be found,
J = Column Subset of an element whose address to be found,
B = Base address,
W = Storage size of one element store in any array(in byte),
LR = Lower Limit of row/start row index of matrix(If not given assume it as zero),
LC = Lower Limit of column/start column index of matrix(If not given assume it as zero),
M = Number of rows given in the matrix.

Example: Given an array **arr[1……10][1……15]** with a base value of **100** and the size of each element is **1 Byte** in memory find the address of arr[8][6] with the help of column-major order.

G***iven:***

Base address B = 100

Storage size of one element store in any array W = 1 Bytes

Row Subset of an element whose address to be found I = 8

Column Subset of an element whose address to be found J = 6

Lower Limit of row/start row index of matrix LR = 1

Lower Limit of column/start column index of matrix = 1

Number of Rows given in the matrix M = Upper Bound – Lower Bound + 1

= 10 – 1 + 1

= 10

Formula: used

*Address of A[I][J] = B + W * ((J – LC) * M + (I – LR))*

*Address of A[8][6] = 100 + 1 * ((6 – 1) * 10 + (8 – 1))*

*= 100 + 1 * ((5) * 10 + (7))*

*= 100 + 1 * (57)*

Address of A[I][J] = 157

Sparse Matrix

Sparse matrices are those matrices that have the majority of their elements equal to zero. In other words, the sparse matrix can be defined as the matrix that has a greater number of zero elements than the non-zero elements.

Benefits of using the sparse matrix :

- **Storage -** Sparse matrix contains lesser non-zero elements than zero, so less memory can be used to store elements. It evaluates only the non-zero elements.

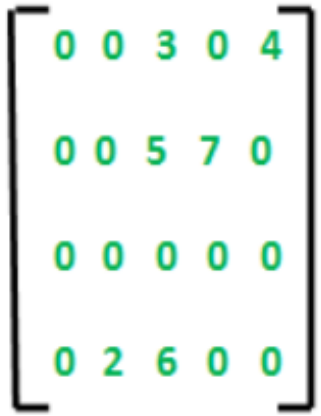
- **Computing time:** In the case of searching in sparse matrix, we need to traverse only the non-zero elements rather than traversing all the sparse matrix elements. It saves computing time by logically designing a data structure traversing non-zero elements.

```
0 0 3 0 4
0 0 5 7 0
0 0 0 0 0
0 2 6 0 0
```

Sparse Matrix Representation

2D array is used to represent a sparse matrix in which there are three rows named as

- **Row:** Index of row, where non-zero element is located
- **Column:** Index of column, where non-zero element is located
- **Value:** Value of the non zero element located at index – (row, column)



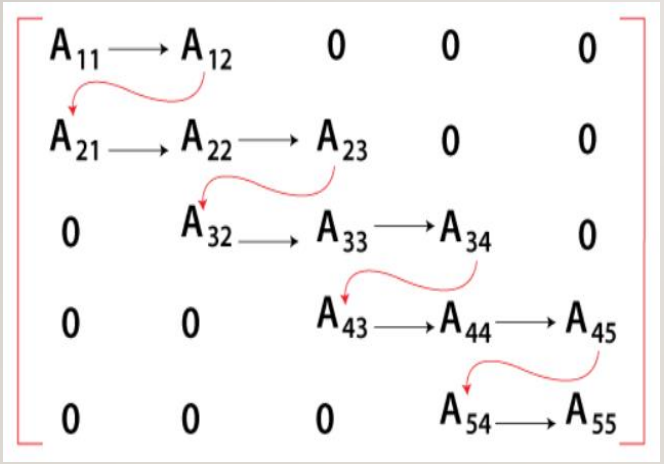
Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

Types Sparse Matrix

There are different variations of sparse matrices, which depend on the nature of the sparsity of the matrices.

- **Diagonal matrix:** only diagonal elements have non zero values.

- **Tri-diagonal sparse matrices:** Elements at diagonal as well as elements immediately below and above main diagonal have of non zero elements.



Example, The 5 by 5 tri-diagonal regular sparse matrix, as shown in the above figure, is stored in one-dimensional array D is:

D = { A_{11}, A_{12}, A_{21}, A_{22}, A_{23}, A_{32}, A_{33}, A_{34}, A_{43}, A_{44}, A_{45}, A_{54}, A_{55}}

Total number of elements for n-square matrix = 3n-2

➢ Lower triangular sparse matrix:

A Lower regular sparse matrix is the one where all elements above the main diagonal are zero value.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 9 & -2 & 0 & 0 & 0 \\ -2 & 1 & 3 & 0 & 0 \\ 3 & 1 & -1 & 6 & 0 \\ 0 & 6 & 7 & 2 & 7 \end{bmatrix}$$

Representation of Lower Lower regular sparse

Example: The 5 by 5 lower triangular regular sparse matrix as shown in the above figure is stored in one-dimensional array B is:

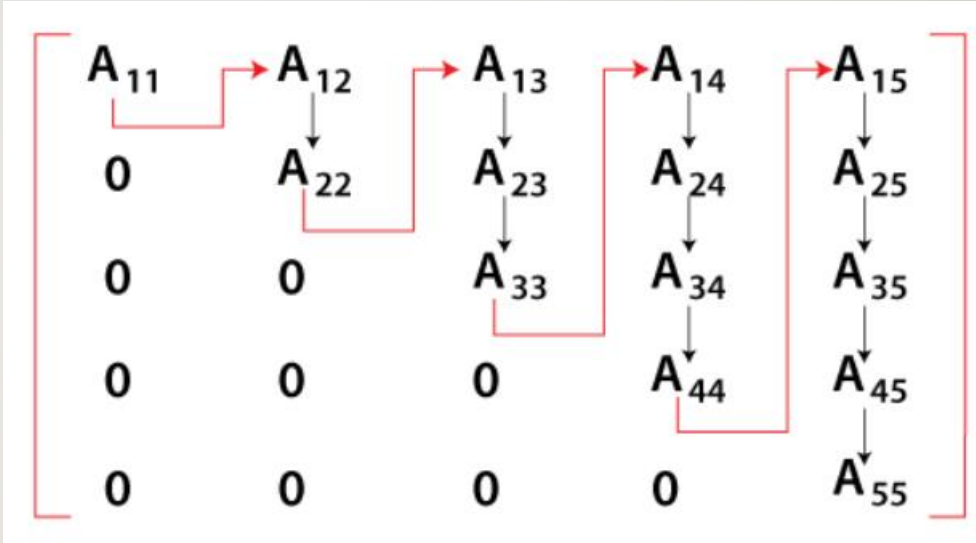
$$B = \{ A_{11}, A_{21}, A_{22}, A_{31}, A_{32}, A_{33}, A_{41}, A_{42}, A_{43}, A_{44}, A_{51}, A_{52}, A_{53}, A_{54}, A_{55}\}$$

Total number of non-zero elements in the lower triangular regular sparse matrix of n rows is:

1 + 2 + …… + i + …….. + (n-1) + n = n (n+1)/2

- **Upper triangular sparse matrix:**
  All the elements below the main diagonal are zero value.

$$\begin{bmatrix} 2 & -1 & -5 & 9 & 7 \\ 0 & 4 & 2 & 4 & 6 \\ 0 & 0 & 3 & 1 & 5 \\ 0 & 0 & 0 & 6 & 4 \\ 0 & 0 & 0 & 0 & -2 \end{bmatrix}$$



Example, The 5 by 5 lower triangular regular sparse matrix, as shown in the above figure, is stored in one-dimensional array B is:

$B = \{ A_{11}, A_{21}, A_{22}, A_{31}, A_{32}, A_{33}, A_{41}, A_{42}, A_{43}, A_{44}, A_{51}, A_{52}, A_{53}, A_{54}, A_{55}\}$

Total number of non-zero elements in lower triangular regular sparse matrix of **n** rows is

1 + 2 + …… + i + …….. + (n-1) + n = n (n+1)/2