Data Structure & Algorithms
Array: 1D Array, 2D Array

Instructor

Dr. Prakash Sharma

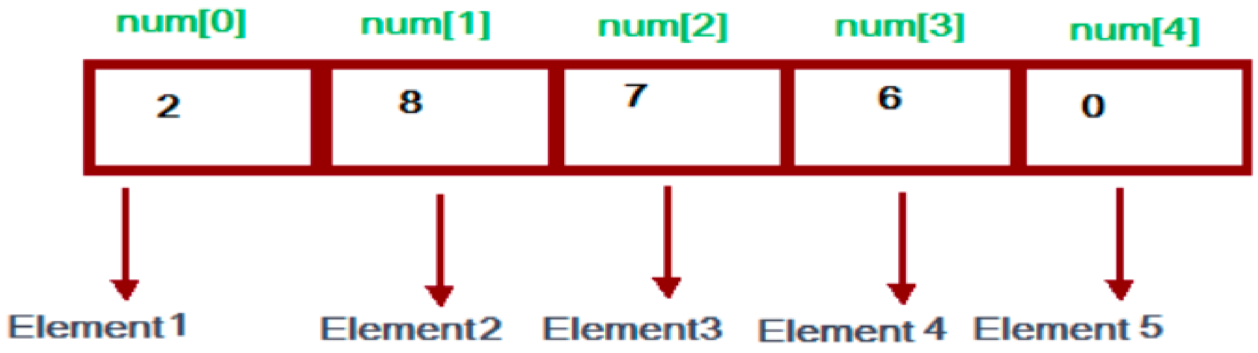
Associate Professor

Department of Information Technology

Manipal University Jaipur

Array

- Arrays are defined as the collection of similar types of data items stored at contiguous memory locations.
- It is one of the simplest data structures where each data element can be randomly accessed by using its index number.

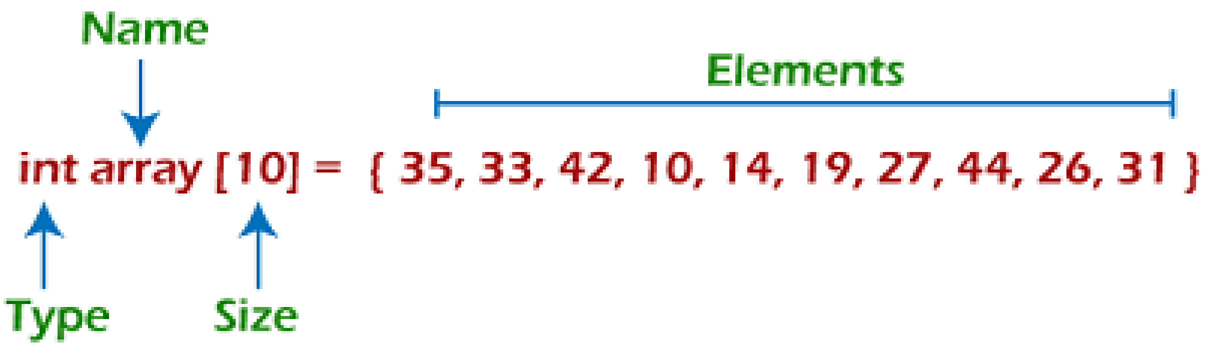


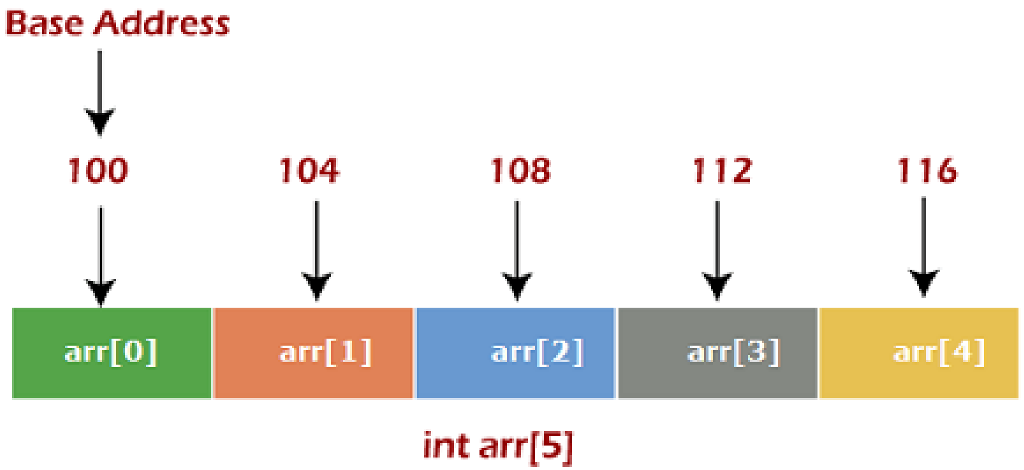
- We can calculate the address of each element of the array with the given base address and the size of the data element.
- The name of the array represents the base address or the address of the first element in the main memory.

Representation & Memory allocation of an array

How to declare an array in C

datatype array Name[array Size];
int data[100];





How to initialize an array

- It is possible to initialize an array during declaration. For example,

  ```
  int mark[5] = {19, 10, 8, 17, 9};
  ```

- You can also initialize an array like this.

  ```
  int mark[] = {19, 10, 8, 17, 9};
  ```

- Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.

- 0 (zero-based indexing): The first element of the array is indexed by subscript of 0 in C/C++

Size of an array

- Number of elements in an 1D array = (Upper bound – Lower Bound) + 1
- Lower bound--index of the first element of the array
- Upper bound--index of the last element of the array
- Size of an array = number of elements * Size of each elements in bytes
- **Address of the element at kth index in one dimensional array:**

a[k] = B + W*(k – Lower bound)

Where

B is the base address of the array

W is the size of each element

K is the index of the element

Lower bound index of the first element of the array

Upper bound index of the last element of the array

Address of element A[k]

- The formula to calculate the address to access an array element –

Address of element A[k] = base address + size * (k - lower index)

A[k] = B + W * (k-LB)

- Example - Suppose an array, A[-10 ..... +2] having Base address (B) = 999 and size of an element (W)= 2 bytes, find the location of A[-1].

L(A[-1]) = B + W * (k - LB)

= 999 + 2 x [(-1) - (-10)]

= 999 + 18

= 1017

Address of element A[k] : Assignment 1

Qu1: Let the base address of the first element of the array is 250 and each element of the array occupies 3 bytes in the memory, then address of the fifth element of a one- dimensional array a[10] ?

Qu2: An array has been declared as follows A: array [-6--------6] of elements where every element takes 4 bytes, if the base address of the array is 3500 find the address of array[0]?

Two-Dimensional Array (2D Array)

- 2D array can be defined as an array of arrays.
- The 2D array is organized as matrices which can be represented as the collection of rows and columns.
- The number of elements that can be present in a 2D array will always be equal to (**no of rows*no of columns**)

a[n][n]

- How to declare 2D Array

```
int arr[max_rows][max_columns];
```

- Initialization of 2D array

```
int arr[2][2] = {0,1,2,3};
```

2D Array: Example

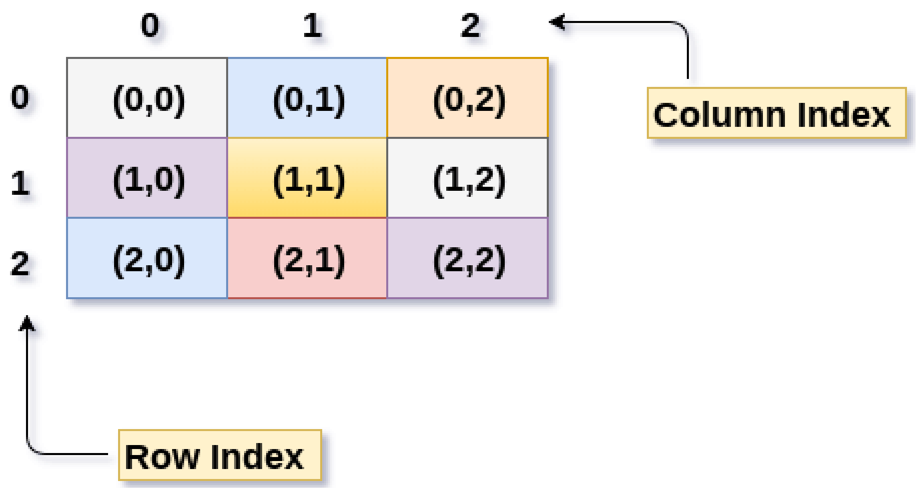
Storing User's data into a 2D array and printing it.

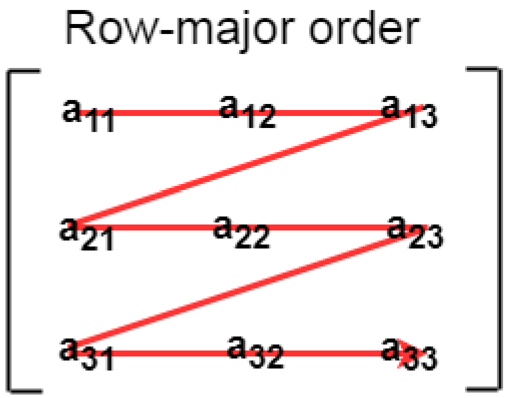
```
#include <stdio.h>
void main ()
{
    int arr[3][3],i,j;
            for (i=0;i<3;i++)
            {
                        for (j=0;j<3;j++)
                        {
                        printf("Enter a[%d][%d]: ",i,j);
                        scanf("%d",&arr[i][j]);
                        }
            }
            printf("\n printing the elements ....\n");
            for(i=0;i<3;i++)
            {
            printf("\n");
                        for (j=0;j<3;j++)
                        {
                        printf("%d\t",arr[i][j]);
                        }
            }
}
```

Representation of 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 row-by-row.
- The 1st row of the array is stored into the memory first, then the 2nd row of the array is stored into the memory completely and so on till the last row.



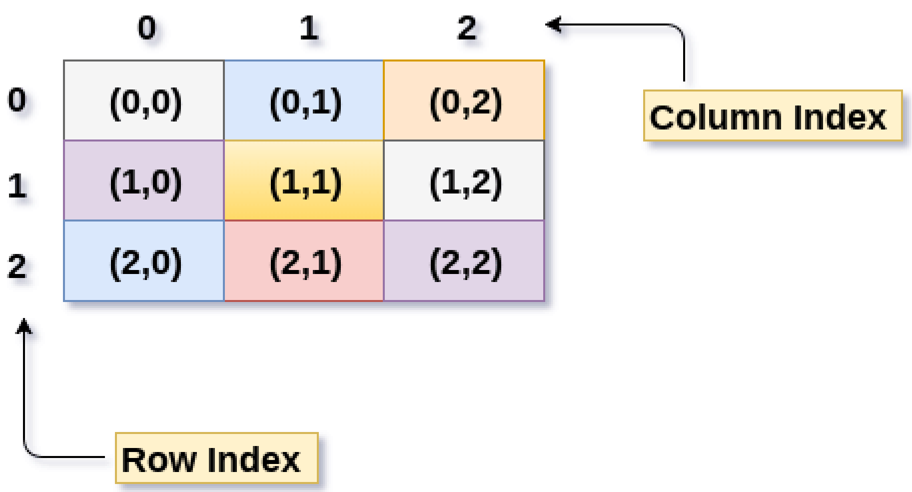


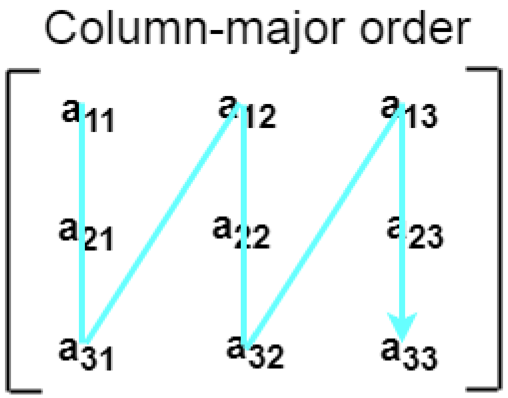
(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)	(2,0)	(2,1)	(2,2)

Representation of 2D array elements into memory

2. Column Major ordering

- According to the column major ordering, all the columns of the 2D array are stored into the memory contiguously.
- The 1st column 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 column of the array.





(0,0)	(1,0)	(2,0)	(0,1)	(1,1)	(2,1)	(0,2)	(1,2)	(2,2)

Row Major implementation of 2D array : Address of a[i][j]

Address of a[i][j] = B + W*[(Uc –Lc + 1) (i-Lr) + (j-Lc)]

Or

Address of a[i][j] = B + W*[n (i - Lr) + (j - Lc)]

B = Base address

W = Size of each element

Lr = Lower bound of rows

Ur = Upper bound of rows

Lc = Lower bound of columns

Uc = Upper bound of columns

(Uc – Lc + 1) = numbers of columns = n

(i - Lr) = number of rows before us

(j - Lc) = number of elements before us in current row

Calculating the Address of the random element of a 2D array

Example:

a[10...30, 55...75],

base address of the array (B) = 0, size of an element = 4 bytes .

Find the location of a[15][68].

Address(a[i][j]) = $\mathbf{B + W(n*(i - L_r) + (j - L_c))}$

Address(a[15][68]) = 0 + 4 ((75 - 55 + 1) x (15 - 10) + (68 - 55))

= 4 (21 x 5 + 13)

= 118 x 4

= 472 answer

Coloumn Major implementation of 2D array: Address of a[i][j]

Address of a[i][j] = B + W*[(Ur – Lr + 1) (j - Lc) + (I - Lr)]

Or

Address of a[i][j] = B + W*[m (j - Lc) + (i - Lr)]

B = Base address

W = Size of each element

Lr = Lower bound of rows

Ur = Upper bound of rows

Lc = Lower bound of columns

Uc = Upper bound of columns

(Ur – Lr + 1) = numbers of rows = m

(j - Lc) = number of coloumns before us

(i - Lr) = number of elements before us in current coloumn

Calculating the Address of the random element of a 2D array

- Example

A[5 ... +20][20 ... 70], BA = 1020, Size of element = 8 bytes.

Find the location of a[0][30].

Address of (a[i][j]) = B + W(m*(j - L_c) + (i - L_r))

m=20-5+1=16 = no of rows

Address[A[0][30]) = 1020 + 8*((0-5) x 16 + (30-20))

= 1020 + 8*(-70)=1020-560

= 460bytes

Assignments – 1 (Qu 1 to 7)

Qu 1: A 2D array defined as a[4,…7, -1,…3] requires 2-bytes of storage space for each element. If the array is stored in row-major form, then calculate the address of element at location a[6,2]. Given that the base address is 100.

Address of (a[i][j]) = B + W(n*(i - L_r) + (j - L_c))

Qu 2: Each element of an array a[-20,…20, 10…35] requires one byte of storage. If the array is coloumn major implemented and the beginning address of the array is 600. Determine the address of element a[0,30]

Qu 3: Calculate the address of X[4,3] in a 2D array X[1,…5, 1…4] stored in row major order in the main memory. Assume the base address to be 1000 and each element requires 4 words of storage.

Address of (a[i][j]) = B + W(n*(i - L_r) + (j - L_c))

Qu 4: Each element of an array data[20][50] requires 4 bytes of storage. Base address of data is 2000. Determine the location of data [10][10] when the array is stored as

(a) Row Major

(b) Coloumn Major

Qu 5: Given an array X[6][6] whose base address is 100. Calculate the location X[2][5] if each element occupies 4 bytes and array is stored row wise.

Address of (a[i][j]) = B + W(n*(i - L_r) + (j - L_c))

Qu 6: A 2D array X[5][4] is stored row wise in the memory. The first element of the array is stored at location 80. Find the memory location of X[3][2]; if the each elements of array requires 4 bytes memory address.

Address of (a[i][j]) = B + W(n*(i - L_r) + (j - L_c))

Qu 7: An array VAL[1…15][1…10] is stored in the memory with each element requiring 4 bytes of storage. If the base address of the array VAL is 1500, determine the location of VAL[12][9] when the array VAL is stored

(i) Row wise

(ii) Column wise

Advantages of Array

- Array provides the single name for the group of variables of the same type. Therefore, it is easy to remember the name of all the elements of an array.

- Traversing an array is a very simple process; we just need to increment the base address of the array in order to visit each element one by one.

- Any element in the array can be directly accessed by using the index.

Disadvantages of Array

- Array is homogenous. It means that the elements with similar data type can only be stored in it.

- In array, there is static memory allocation that is size of an array cannot be altered.

- There will be wastage of memory if we store less number of elements than the declared size.

What is a 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.
- Why is a sparse matrix required
- **Storage -** We know that a 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.

Representation of sparse matrix

There are two ways to represent the sparse matrix that are listed as follows -

1. Array Representation of Sparse Matrix: In 2D array representation of sparse matrix, there are three fields used that are named as -

- **Row -** It is the index of a row where a non-zero element is located in the matrix.
- **Column -** It is the index of the column where a non-zero element is located in the matrix.
- **Value -** It is the value of the non-zero element that is located at the index (row, column).

Array Representation of Sparse Matrix

- **Example -**

Sparse Matrix

	0	1	2	3
0	0	4	0	5
1	0	0	3	6
2	0	0	2	0
3	2	0	0	0
4	1	0	0	0

ROW	COL	VALUE

Table Structure

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

Linked List representation of the sparse matrix

- The advantage of using a linked list to represent the sparse matrix is that the complexity of inserting or deleting a node in a linked list is lesser than the array.
- A node in the linked list representation consists of four fields. The four fields of the linked list are given as follows -
- **Row -** It represents the index of the row where the non-zero element is located.
- **Column -** It represents the index of the column where the non-zero element is located.
- **Value -** It is the value of the non-zero element that is located at the index (row, column).
- **Next node -** It stores the address of the next node.

Linked List representation of the sparse matrix

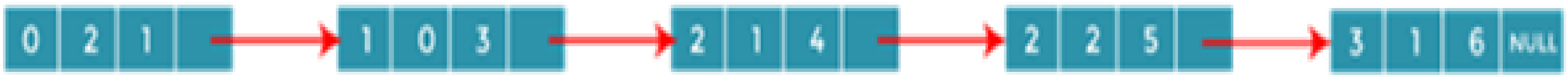
- **Example –**

Node Structure

Row	Column	Value	Pointer to Next Node

Sparse Matrix →

$$
\begin{array}{c|cccc|}
 & 0 & 1 & 2 & 3 \\
\hline
0 & 0 & 0 & 1 & 0 \\
1 & 3 & 0 & 0 & 0 \\
2 & 0 & 4 & 5 & 0 \\
3 & 0 & 6 & 0 & 0 \\
\hline
\end{array}
$$



Diagonal Matrix

- Only the elements in the diagonal are non-zero and other than diagonal, all elements must be ‘0’. Then only we say it is a diagonal matrix.

$$M_{5 \times 5} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 7 \end{bmatrix}$$

If we have non-zero elements other than diagonal, then that will not be a diagonal matrix. Below is not a diagonal matrix.

If row number and column number are the same, then the value will be non-zero and if row number and column number are different then the value will be ‘**0**’ in the diagonal matrix.

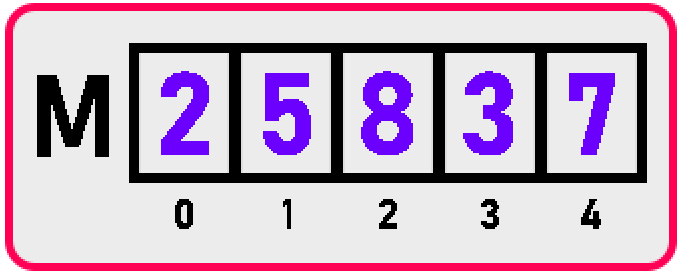
$$M_{5 \times 5} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 8 & 0 & 0 & 0 & 7 \end{bmatrix}$$

$$M[i, j] = 0, \text{ if } i \neq j$$

Diagonal Matrix

- For storing non-zero elements we can take just a single dimension array and store these elements.

$$M = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 7 \end{bmatrix}_{5 \times 5}$$



Now let us see how we can access these elements from a single dimension array if we want to access them

1. If we want to access **M [0, 0]**, this element is present on the **0th** index in the array.
2. If we want to access **M [1, 1]**, this element is present on the **1** index in the array.

TriDiagonal Matrix:

- If we look at the elements, non-zero elements are present in the main diagonal, lower diagonal, and upper diagonal and the rest of the elements are all zeros.

$$
M = \begin{array}{c c} & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{bmatrix} \end{array}
$$

If we observe the indices of elements of:

1. **Main Diagonal:** row number is equal to column number **(i = j).**
2. **Lower Diagonal:** row number – column number = 1 **(i – j = 1).**
3. **Upper Diagonal:** row number – column number = -1 **(i – j = -1).**

So, the conditions are:

Main Diagonal : $i = j$.
Lower Diagonal: $i - j = 1$.
Upper Diagonal: $i - j = -1$.

$M[i][j] \neq 0$ if $|i - j| <= 1$
$M[i][j] = 0$ if $|i - j| > 1$

TriDiagonal Matrix:

- **How many non-zero elements are there?**

= n + n-1 + n-1.

= 3n – 2 (non-zero elements)

= 3 * 5 – 2

= 13

TriDiagonal Matrix:

Now how to map these?

case 1: if $|i - j| = 1$

index: $i - 1$

B	a_{21}	a_{32}	a_{43}	a_{54}									
	0	1	2	3	4	5	6	7	8	9	10	11	12
	Lower Diagonal												

case 2: if $i - j = 0$

index: $(n-1) + (i-1)$

B	a_{21}	a_{32}	a_{43}	a_{54}	a_{11}	a_{22}	a_{33}	a_{44}	a_{55}				
	0	1	2	3	4	5	6	7	8	9	10	11	12
	Lower Diagonal				Main Diagonal								

case 3: if $i - j = -1$

index: $(2n-1) + (i-1)$

B	a_{21}	a_{32}	a_{43}	a_{54}	a_{11}	a_{22}	a_{33}	a_{44}	a_{55}	a_{12}	a_{23}	a_{34}	a_{45}
	0	1	2	3	4	5	6	7	8	9	10	11	12
	Lower Diagonal				Main Diagonal					Upper Diagonal			

Lower Triangular Matrix

- A lower triangular matrix is a square matrix in which the lower triangular part of a matrix is non-zero elements and the upper triangular part is all zeros and none of them is non-zero. The set of non-zero elements are forming a triangle.

$$
M = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & a_{11} & 0 & 0 & 0 & 0 \\ 2 & a_{21} & a_{22} & 0 & 0 & 0 \\ 3 & a_{31} & a_{32} & a_{33} & 0 & 0 \\ 4 & a_{41} & a_{42} & a_{43} & a_{44} & 0 \\ 5 & a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{array}
$$

$j \rightarrow$, $i \downarrow$

$$M[i, j] = 0, \text{ if } i < j$$
$$M[i, j] \neq 0, \text{ if } i >= j$$

For non-zero elements ‘i’ value is always greater and for zero elements, the ‘i’ value is smaller. It means if the row number is less than the column number (i < j) then the element must be zero.

Lower Triangular Matrix

- 1^{st} row is having 1 non-zero element.
- 2^{nd} row is having 2 non-zero elements.
- …
- 5^{th} row is having 5 non-zero elements.
- For any n x n matrix,

  number of non-zero elements = 1 + 2 + … + n = n (n + 1) / 2.
- In a matrix of n x n, total n^2 elements will be there and n (n + 1) / 2 are non-zero elements. So,

$$\text{Number of zero elements} = n^2 - n(n+1)/2$$
$$= n(n-1)/2$$

Upper Triangular Matrix

- An upper triangular matrix is a square matrix in which the upper triangular part of a matrix is non-zero elements and the lower triangular part is all zeros and none of them is non-zero. The set of non-zero elements are forming a triangle.

$$
M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\
0 & a_{22} & a_{23} & a_{24} & a_{25} \\
0 & 0 & a_{33} & a_{34} & a_{35} \\
0 & 0 & 0 & a_{44} & a_{45} \\
0 & 0 & 0 & 0 & a_{55}
\end{bmatrix}
\end{array}
$$

condition for finding non-zero and zero elements

$$
\boxed{\begin{array}{l} M[i, j] = 0, \text{ if } i > j \\ M[i, j] \neq 0, \text{ if } i <= j \end{array}}
$$

Upper Triangular Matrix

- 1^{st} row is having 5 non-zero element.
- 2^{nd} row is having 4 non-zero elements.
- …
- 5^{th} row is having 1 non-zero elements.
- For any n x n matrix,

  number of non-zero elements = $n + (n-1) + \dots 2 + 1 = n (n + 1) / 2$.
- In a matrix of n x n, total n^2 elements will be there and $n (n + 1) / 2$ are non-zero elements. So,

$$\text{Number of zero elements} = n^2 - n (n + 1) / 2$$
$$= n (n-1) / 2$$