1-D ARRAYS (types, searching, sorting, traversing, insertion, deletion)

2-D ARRAYS

Two dimensional array is an array within an array. It is an array of arrays. In this type of array the position of an data element is referred by two indices instead of one. So it represents a table with rows an dcolumns of data.

Eg1.

T = [[11, 12, 5, 2], [15, 6,10], [10, 8, 12, 5], [12,15,8,6]]

11 12 5 2

15 6 10

10 8 12 5

12 15 8 6

Eg.2

T = [[11, 12, 5, 2], [15, 6,10], [10, 8, 12, 5], [12,15,8,6]]

print(T[0])

print(T[1][2])

**Output**

[11, 12, 5, 2]

10

Inserting Values

We can insert new data elements at specific position by using the insert() method and specifying the index.

T = [[11, 12, 5, 2], [15, 6,10], [10, 8, 12, 5], [12,15,8,6]]

T.insert(2, [0,5,11,13,6])

for r in T:

for c in r:

print(c,end = " ")

print()

**Output**

11 12 5 2

15 6 10

0 5 11 13 6

10 8 12 5

12 15 8 6

## Updating Values

We can update the entire inner array or some specific data elements of the inner array by reassigning the values using the array index.

T = [[11, 12, 5, 2], [15, 6,10], [10, 8, 12, 5], [12,15,8,6]]

T[2] = [11,9]

T[0][3] = 7

for r in T:

for c in r:

print(c,end = " ")

print()

### Output

When the above code is executed, it produces the following result −

11 12 5 7

15 6 10

11 9

12 15 8 6

## Deleting the Values

## We can delete the entire inner array or some specific data elements of the inner array by reassigning the values using the del() method with index. But in case you need to remove specific data elements in one of the inner arrays, then use the update process described above.

T = [[11, 12, 5, 2], [15, 6,10], [10, 8, 12, 5], [12,15,8,6]]

del T[3]

for r in T:

for c in r:

print(c,end = " ")

print()

### Output

When the above code is executed, it produces the following result −

11 12 5 2

15 6 10

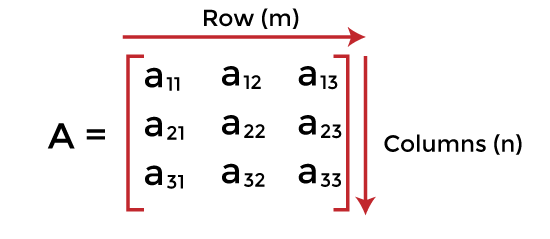
10 8 12 5

# **Sparse Matrix**

### What is a matrix?

A matrix can be defined as a two-dimensional array having 'm' rows and 'n' columns. A matrix with m rows and n columns is called m × n matrix. It is a set of numbers that are arranged in the horizontal or vertical lines of entries.

For example -



### 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 if we can use the simple matrix to store elements?

There are the following benefits of using the sparse matrix -

**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

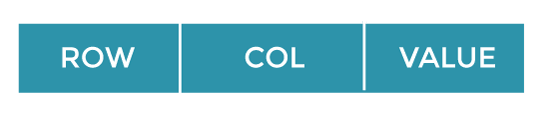
The non-zero elements in the sparse matrix can be stored using triplets that are rows, columns, and values. There are two ways to represent the sparse matrix that are listed as follows -

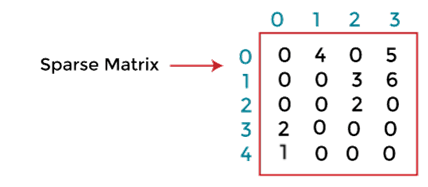
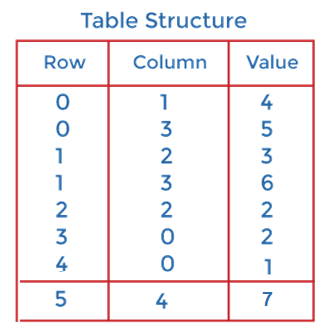
* Array representation
* Linked list representation

**Array representation of the sparse matrix**

Representing a sparse matrix by a 2D array leads to the wastage of lots of memory. This is because zeroes in the matrix are of no use, so storing zeroes with non-zero elements is wastage of memory. To avoid such wastage, we can store only non-zero elements. If we store only non-zero elements, it reduces the traversal time and the storage space.

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).
* 
* In the above figure, we can observe a 5x4 sparse matrix containing 7 non-zero elements and 13 zero elements. The above matrix occupies 5x4 = 20 memory space. Increasing the size of matrix will increase the wastage space.
* AD
* The tabular representation of the above matrix is given below -
* 
* In the above structure, first column represents the rows, the second column represents the columns, and the third column represents the non-zero value. The first row of the table represents the triplets. The first triplet represents that the value 4 is stored at 0th row and 1st column. Similarly, the second triplet represents that the value 5 is stored at the 0th row and 3rd column. In a similar manner, all triplets represent the stored location of the non-zero elements in the matrix.
* The size of the table depends upon the total number of non-zero elements in the given sparse matrix. Above table occupies 8x3 = 24 memory space which is more than the space occupied by the sparse matrix. So, what's the benefit of using the sparse matrix? Consider the case if the matrix is 8\*8 and there are only 8 non-zero elements in the matrix, then the space occupied by the sparse matrix would be 8\*8 = 64, whereas the space occupied by the table represented using triplets would be 8\*3 = 24.

CODE in python

L = [[0, 0, 0, 1, 1],

     [0, 0, 0, 0, 0],

     [1, 0, 0, 0, 0]]

j = 0

non\_zero\_num = 0

zero\_num = 0

L\_1 = []

while j < len(L):

    i=0

    while i < (len(L[j])):

        if L[j][i] == 0:

            zero\_num += 1

        else:

            non\_zero\_num += 1

        i += 1

    j += 1

if non\_zero\_num < zero\_num:

    print("Given Matrix is Sparse")

else:

    print("Given Matrix is Dense")

Output:

Given Matrix is Sparse