Sparse Matrix and its representations | Set 1 (Using Arrays and Linked Lists)

A matrix is a two-dimensional data object made of m rows and n columns, therefore having total m x n values. If most of the elements of the matrix have **0 value**, then it is called a sparse matrix.
Why to use Sparse Matrix instead of simple matrix ?

- **Storage:** There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.
- **Computing time:** Computing time can be saved by logically designing a data structure traversing only non-zero elements..

Example:

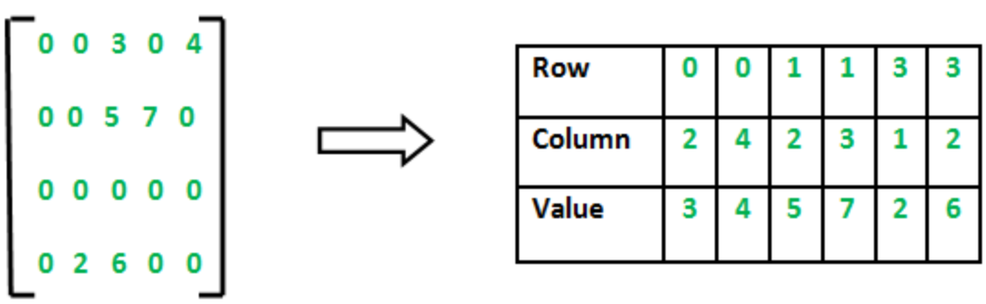
```
0 0 3 0 4

0 0 5 7 0

0 0 0 0 0

0 2 6 0 0
```

Representing a sparse matrix by a 2D array leads to wastage of lots of memory as zeroes in the matrix are of no use in most of the cases. So, instead of storing zeroes with non-zero elements, we only store non-zero elements. This means storing non-zero elements with **triples- (Row, Column, value).**
Sparse Matrix Representations can be done in many ways following are two common representations:

1. Array representation
2. Linked list representation

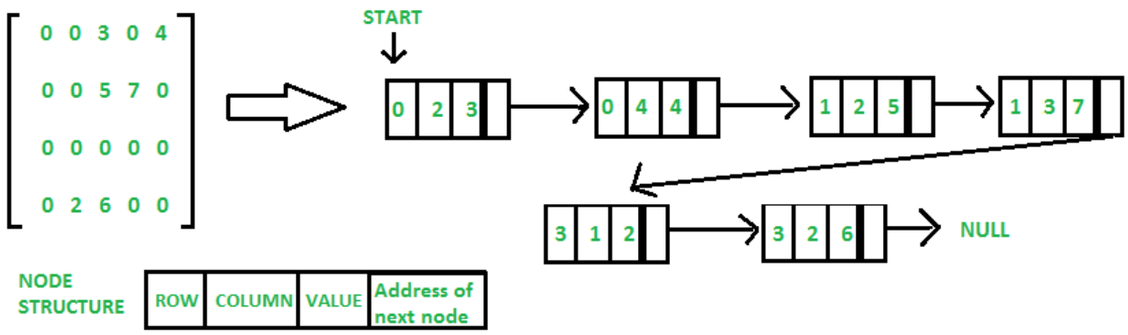
Method 1: Using Arrays
2D array is used to represent a sparse matrix in which there are three rows named as

- **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)



Method 2: Using Linked Lists
In linked list, each node has four fields. These four fields are defined 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)
- **Next node:** Address of the next node



In case of unsorted list, we have to search the entire list every time i.e. we have to keep on searching the list till we find the required element or we reach the end of the list. This is because as elements are not in any order, so any element can be found just anywhere.

Algorithm:

```
linear search(int x[],int n,int key)
{
  int i,flag = 0;
  for(i=0;i < n ; i++)
   {
   if(x[i]==key)
    {
    flag=1;
    break;
    }
   }
   if(flag==0)
   return(-1);
   else
  return(1);
}

BinarySearch(A[0..n-1], value, low, high)

{

   if (high < low)

      return -1       // not found

   mid = low + ((high - low) / 2)

   if (A[mid] > value)

    return BinarySearch(A, value, low, mid-1)

   else if (A[mid] < value)

    return BinarySearch(A, value, mid+1, high)

   else

    return mid                  // found

}
```