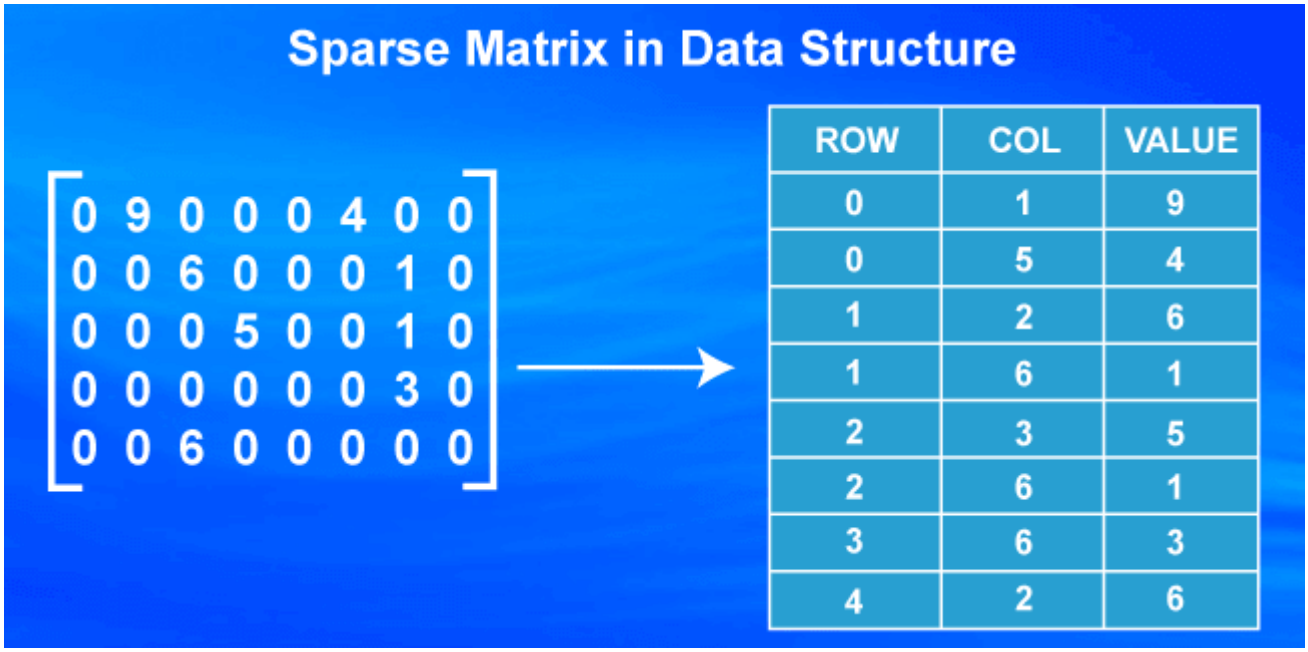
Types of Sparse Matrices?

In this article, we will understand the sparse matrices and their types in detail.

What do you mean by Sparse Matrices?

A wide range of numerical problems in real-life applications such as engineering, scientific, computing, and economic use huge matrices. These matrices often contain many zero elements, and such matrices with high proportions of zero entries are known as sparse matrices.



It is called sparse as it has a relatively low density of non-zero elements. If we store Sparse Matrix as a 2-dimension array, a lot of space is wasted to store all those 0's explicit. Moreover, while performing operations on sparse matrices such as addition, subtraction, and multiplication stored as a 2-dimension array, many operations are performed on elements with zero values which results in the amount of wastage in time. So, a better strategy is to explicitly store the non-zero elements, which greatly reduces the required storage space and computations needed to perform various operations.

Types of Sparse Matrices

There are different variations of sparse matrices, which depend on the nature of the sparsity of the matrices. Based on these properties, sparse matrices can be

- Regular sparse matrices
- Irregular sparse matrices / Non - regular sparse matrices

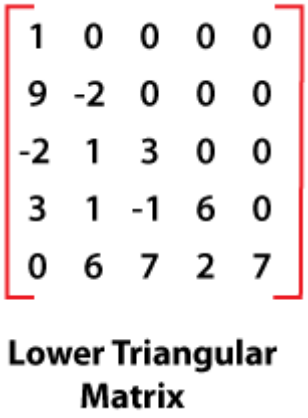
Regular sparse matrices

A regular sparse matri**x** is a square matrix with a well-defined sparsity pattern, i.e., non-zero elements occur in a well-defined pattern. The various types of regular sparse matrices are:

- Lower triangular regular sparse matrices
- Upper triangular regular sparse matrices
- Tri-diagonal regular sparse matrices

Lower triangular regular sparse matrices

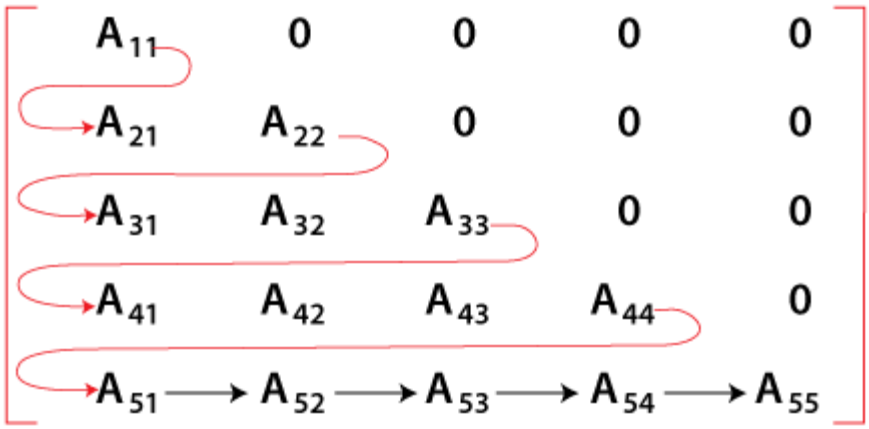
A Lower regular sparse matrix is the one where all elements above the main diagonal are zero value. The following matrix is a lower triangular regular sparse matrix.



Lower Triangular Matrix

Storing Lower triangular regular sparse matrices

In a lower triangular regular sparse matrix, the non-zero elements are stored in a 1-dimensional array row by row.



Representation of lower triangular matrix A[5, 5]

For 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}}

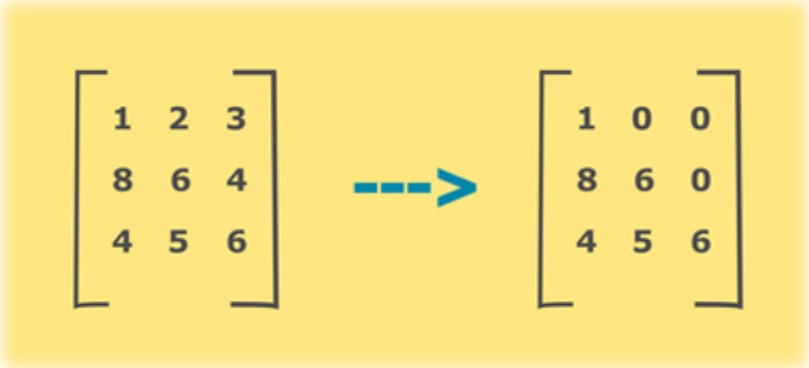
Here B [1] = A_{11}, B [2] = A_{21}, B [3] = A_{22}, …… B [14] = $A_{54,}$ B [15] = A_{55}

To calculate the total number of non-zero elements, we need to know the number of non-zero elements in each row and then add them. Since the number of non-zero elements in an ith row so that the 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

Lower Triangular Matrix

Lower triangular matrix is a square matrix in which all the elements above the principle diagonal will be zero. To find the lower triangular matrix, a matrix needs to be a square matrix that is, number of rows and columns in the matrix needs to be equal. Dimensions of a typical square matrix can be represented by n x n.



Consider the above example, principle diagonal element of given matrix is (1, 6, 6). All the elements above diagonal needs to be made zero. In our example, those elements are at positions (1,2), (1,3) and (2,3). To convert given matrix into the lower triangular matrix, loop through the matrix and set the values of the element to zero where column number is greater than row number.

Algorithm

1. Declare and initialize a two-dimensional array a.
2. Calculate the number of rows and columns present in the array and store it in variables rows and cols respectively.
3. If the number of rows are not equal to the number of columns, then the given matrix is not a square matrix. Hence, given matrix cannot be converted to the lower triangular matrix. Display the error message.
4. If rows = cols, traverse the array a using two loops where outer loop represents the rows, and inner loop represents the columns of the array a. To convert given matrix to lower triangular matrix, set the elements of the array to 0 where (j > i) that is, the column number is greater than row number.
5. Display the resulting matrix.

Lower Solution in Python

```
#Initialize matrix a
a = [
        [1, 2, 3],
        [8, 6, 4],
        [4, 5, 6]
    ];

#Calculates number of rows and columns present in given matrix
rows = len(a);
cols = len(a[0]);

if(rows != cols):
    print("Matrix should be a square matrix");
else:
    #Performs required operation to convert given matrix into lower triangular matrix
    print("Lower triangular matrix: ");
    for i in range(0, rows):
        for j in range(0, cols):
            if(j > i):
                print("0"),
            else:
                print(a[i][j]),

        print(" ");
```

Output:

```
Lower triangular matrix:
1 0 0
8 6 0
4 5 6
```

Lower Solution in C

```
#include <stdio.h>

int main()
{
    int rows, cols;

    //Initialize matrix a
    int a[][3] = {
                    {1, 2, 3},
                    {8, 6, 4},
                    {4, 5, 6}
                };
```

```
13.
14.    //Calculates number of rows and columns present in given matrix
15.    rows = (sizeof(a)/sizeof(a[0]));
16.    cols = (sizeof(a)/sizeof(a[0][0]))/rows;
17.
18.    if(rows != cols){
19.        printf("Matrix should be a square matrix\n");
20.    }
21.    else{
22.        //Performs required operation to convert given matrix into lower triangular matrix

23.        printf("Lower triangular matrix: \n");
24.        for(int i = 0; i < rows; i++){
25.            for(int j = 0; j < cols; j++){
26.              if(j > i)
27.                printf("0 ");
28.              else
29.                printf("%d ", a[i][j]);
30.            }
31.            printf("\n");
32.        }
33.    }
34.    return 0;
35.}
```

Output:

```
Lower triangular matrix:
1 0 0
8 6 0
4 5 6
```

Lower Solution in JAVA

```
1.  public class LowerTriangular
2.  {
3.      public static void main(String[] args) {
4.          int rows, cols;
5.
6.          //Initialize matrix a
7.          int a[][] = {
8.                          {1, 2, 3},
9.                          {8, 6, 4},
10.                         {4, 5, 6}
11.                     };
12.
13.           //Calculates number of rows and columns present in given matrix
14.           rows = a.length;
```

```
15.         cols = a[0].length;
16.
17.         if(rows != cols){
18.             System.out.println("Matrix should be a square matrix");
19.         }
20.         else {
21.             //Performs required operation to convert given matrix into lower triangular m
    atrix
22.             System.out.println("Lower triangular matrix: ");
23.             for(int i = 0; i < rows; i++){
24.                 for(int j = 0; j < cols; j++){
25.                   if(j > i)
26.                     System.out.print("0 ");
27.                   else
28.                     System.out.print(a[i][j] + " ");
29.                 }
30.                 System.out.println();
31.             }
32.         }
33.     }
34. }
```

Output:

```
Lower triangular matrix:
1 0 0
8 6 0
4 5 6
```

Lower Solution in C#

```
1.  using System;
2.
3.  public class LowerTriangular
4.  {
5.      public static void Main()
6.      {
7.          int rows, cols;
8.
9.          //Initialize matrix a
10.         int[,] a = {
11.                         {1, 2, 3},
12.                         {8, 6, 4},
13.                         {4, 5, 6}
14.                     };
15.
16.           //Calculates number of rows and columns present in given matrix
17.           rows = a.GetLength(0);
```

```
18.          cols = a.GetLength(1);
19.
20.        if(rows != cols){
21.            Console.WriteLine("Matrix should be a square matrix");
22.        }
23.        else {
24.            //Performs required operation to convert given matrix into lower triangular ma
    trix
25.            Console.WriteLine("Lower triangular matrix: ");
26.            for(int i = 0; i < rows; i++){
27.                for(int j = 0; j < cols; j++){
28.                  if(j > i)
29.                    Console.Write("0 ");
30.                  else
31.                    Console.Write(a[i,j] + " ");
32.                }
33.                Console.WriteLine();
34.            }
35.        }
36.    }
37.}
```

Output:

```
Lower triangular matrix:
1 0 0
8 6 0
4 5 6
```

Lower Solution in PHP

```
1.  <!DOCTYPE html>
2.  <html>
3.  <body>
4.  <?php
5.  //Initialize matrix a
6.  $a = array(
7.              array(1, 2, 3),
8.              array(8, 6, 4),
9.              array(4, 5, 6)
10.           );
11.
12. //Calculates number of rows and columns present in given matrix
13. $rows = count($a);
14. $cols = count($a[0]);
15.
16. if($rows != $cols){
17.     print("Matrix should be a square matrix <br>");
```

```
}
else {
    //Performs required operation to convert given matrix into lower triangular matrix
    print("Lower triangular matrix: <br>");
    for($i = 0; $i < $rows; $i++){
        for($j = 0; $j < $cols; $j++){
          if($j > $i)
            print("0 ");
          else
            print($a[$i][$j] . " ");
        }
        print("<br>");
    }
}
?>
</body>
</html>
```

Output:

Lower triangular matrix:
1 0 0
8 6 0
4 5 6

Upper triangular regular sparse matrices

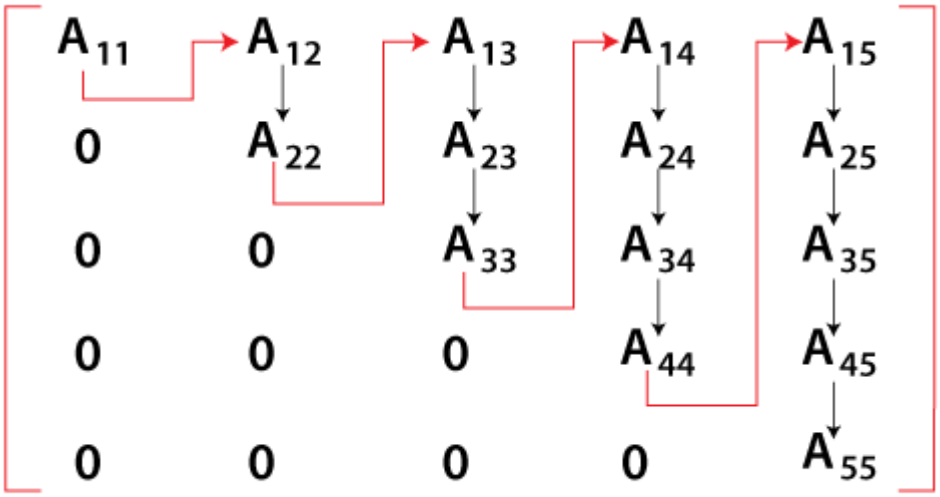
The Upper triangular regular sparse matrix is where all the elements below the main diagonal are zero value. Following matrix is the Upper triangular regular sparse matrix.

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

Upper Triangular Matrix

Storing Upper triangular regular sparse matrices

In an upper triangular regular sparse matrix, the non-zero elements are stored in a 1-dimensional array column by column.



Representation of upper traingular matrix A[5,5]

For 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}}

Here B [1] = A_{11}, B [2] = A_{21}, B [3] = A_{22}, B [4] = A_{31},B [5] = A_{32} ,B [6] = A_{33}, B [7] = A_{41} ,B [8] = A_{42} ,B [9] = A_{43} ,B [10] = A_{44} ,B [11] = A_{51} ,B [12] = A_{52} ,B [13] = A_{53} , B [14] = A_{54}, B [15] = A_{55}

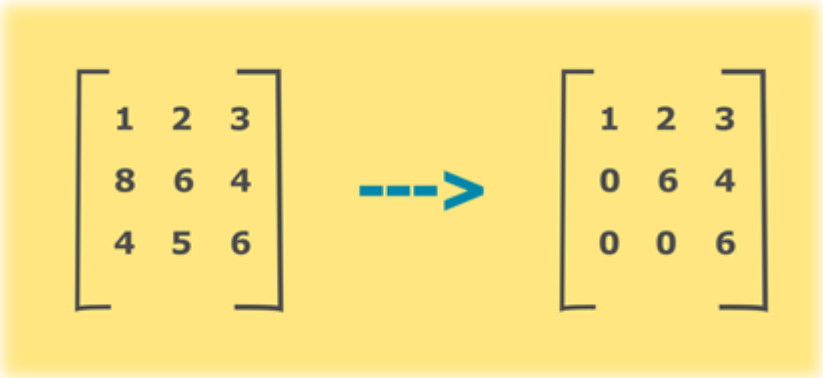
In order to calculate the total number of non-zero elements, we need to know the number of non-zero elements in each row and then add them. Since, the number of non-zero

elements in ith row so that the 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

Upper Triangular Matrix

Upper triangular matrix is a square matrix in which all the elements below the principle diagonal are zero. To find the upper triangular matrix, a matrix needs to be a square matrix that is, the number of rows and columns in the matrix needs to be equal. Dimensions of a typical square matrix can be represented by n x n.



Consider the above example, principle diagonal element of given matrix is (1, 6, 6). All the elements below diagonal needs to be zero to convert it into an upper triangular matrix, in our example, those elements are at positions (2,1), (3,1) and (3,2). To convert given matrix into the upper triangular matrix, loop through the matrix and set the values of the element to zero where row number is greater than column number.

Algorithm

1. Declare and initialize a two-dimensional array a.
2. Calculate the number of rows and columns present in the array and store it in variables rows and cols respectively.
3. If the number of rows is not equal to the number of columns, it implies that the given matrix is not a square matrix. Hence, given matrix cannot be converted to the upper triangular matrix. Display the error message.
4. If rows = cols, traverse the array a using two loops where outer loop represents the rows, and inner loop represents the columns of the array a. To convert given matrix to upper triangular matrix set the elements of the array to 0 where (i > j) that is, the row number is greater than column number.
5. Display the resulting matrix.

Upper Matrix Solution in Python

```
#Initialize matrix a
a = [
      [1, 2, 3],
```

```
        [8, 6, 4],
        [4, 5, 6]
    ];

#Calculates number of rows and columns present in given matrix
rows = len(a);
cols = len(a[0]);

if(rows != cols):
    print("Matrix should be a square matrix");
else:
    #Performs required operation to convert given matrix into upper triangular matrix
    print("Upper triangular matrix: ");
    for i in range(0, rows):
        for j in range(0, cols):
            if(i > j):
                print("0"),
            else:
                print(a[i][j]),

        print(" ");
```

Output:

```
Upper triangular matrix:
1 2 3
0 6 4
0 0 6
```

Upper Matrix Solution in C

```
#include <stdio.h>

int main()
{
    int rows, cols;

    //Initialize matrix a
    int a[][3] = {
                    {1, 2, 3},
                  {8, 6, 4},
                    {4, 5, 6}
                };

    //Calculates number of rows and columns present in given matrix
    rows = (sizeof(a)/sizeof(a[0]));
    cols = (sizeof(a)/sizeof(a[0][0]))/rows;

    if(rows != cols){
        printf("Matrix should be a square matrix\n");
```

```
    }
    else{
        //Performs required operation to convert given matrix into upper triangular matrix
        printf("Upper triangular matrix: \n");
        for(int i = 0; i < rows; i++){
            for(int j = 0; j < cols; j++){
                if(i > j)
                    printf("0 ");
                else
                    printf("%d ", a[i][j]);
            }
            printf("\n");
        }
    }

    return 0;
}
```

Output:

```
Upper triangular matrix:
1 2 3
0 6 4
0 0 6
```

Upper Matrix Solution in JAVA

```
public class UpperTriangular
{
    public static void main(String[] args) {
        int rows, cols;

        //Initialize matrix a
        int a[][] = {
                        {1, 2, 3},
                        {8, 6, 4},
                        {4, 5, 6}
                    };

        //Calculates number of rows and columns present in given matrix
        rows = a.length;
        cols = a[0].length;

        if(rows != cols){
            System.out.println("Matrix should be a square matrix");
        }
        else {
            //Performs required operation to convert given matrix into upper triangular ma
trix
            System.out.println("Upper triangular matrix: ");
```

```
            for(int i = 0; i < rows; i++){
              for(int j = 0; j < cols; j++){
                 if(i > j)
                  System.out.print("0 ");
                 else
                  System.out.print(a[i][j] + " ");
               }
              System.out.println();
            }
        }
    }
}
```

Output:

```
Upper triangular matrix:
1 2 3
0 6 4
0 0 6
```

Upper Matrix Solution in C#

```
using System;

public class UpperTriangular
{
    public static void Main()
    {
        int rows, cols;

        //Initialize matrix a
        int[,] a = {
                        {1, 2, 3},
                      {8, 6, 4},
                        {4, 5, 6}
                  };

         //Calculates number of rows and columns present in given matrix
          rows = a.GetLength(0);
        cols = a.GetLength(1);

        if(rows != cols){
            Console.WriteLine("Matrix should be a square matrix");
        }
        else {
           //Performs required operation to convert given matrix into upper triangular matrix
            Console.WriteLine("Upper triangular matrix: ");
           for(int i = 0; i < rows; i++){
               for(int j = 0; j < cols; j++){
                if(i > j)
```

```
                Console.Write("0 ");
            else
                Console.Write(a[i,j] + " ");
        }
        Console.WriteLine();
    }
  }
}
```

Output:

```
Upper triangular matrix:
1 2 3
0 6 4
0 0 6
```

Upper Matrix Solution in PHP

```
<!DOCTYPE html>
<html>
<body>
<?php
//Initialize matrix a
$a = array(
        array(1, 2, 3),
        array(8, 6, 4),
        array(4, 5, 6)
    );

//Calculates number of rows and columns present in given matrix
$rows = count($a);
$cols = count($a[0]);

if($rows != $cols){
   print("Matrix should be a square matrix<br>");
}
else {
   //Performs required operation to convert given matrix into upper triangular matrix
   print("Upper triangular matrix: <br>");
   for($i = 0; $i < $rows; $i++){
      for($j = 0; $j < $cols; $j++){
        if($i > $j)
          print("0 ");
        else
          print($a[$i][$j] . " ");
      }
      print("<br>");
   }
}
```

```
?>
</body>
</html>
```

Output:

Upper triangular matrix:
1 2 3
0 6 4
0 0 6

Upper triangular regular sparse matrices

The Upper triangular regular sparse matrix is where all the elements below the main diagonal are zero value. The following matrix is an Upper triangular regular sparse matrix.

Storing Upper triangular regular sparse matrices

In an upper triangular regular sparse matrix, the non-zero elements are stored in a 1-dimensional array column by column.

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

C = { A_{11}, A_{12}, A_{22}, A_{13}, A_{23}, A_{33}, A_{14}, A_{24}, A_{34}, A_{44}, A_{15}, A_{25}, A_{35}, A_{45}, A_{55}}

Here C [1] = A_{11}, C [2] = A_{12}, C [3] = A_{22}, C [4] = A_{13},C [5] = A_{23} ,C [6] = A_{33}, C [7] = A_{14} ,C [8] = A_{24} ,C [9] = A_{34}, C [10] = A_{44} ,C [11] = A_{15} ,C [12] = A_{25} ,C [13] = A_{35} , C [14] = A_{45}, C [15] = A_{55}

In order to calculate the total number of non-zero elements, we need to know the number of non-zero elements in each column and then add them. Since, the number of non-zero elements in ith column is **i** so the total number of non-zero elements in upper triangular regular sparse matrix of **n** columns is:

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

Tri-diagonal regular sparse matrices

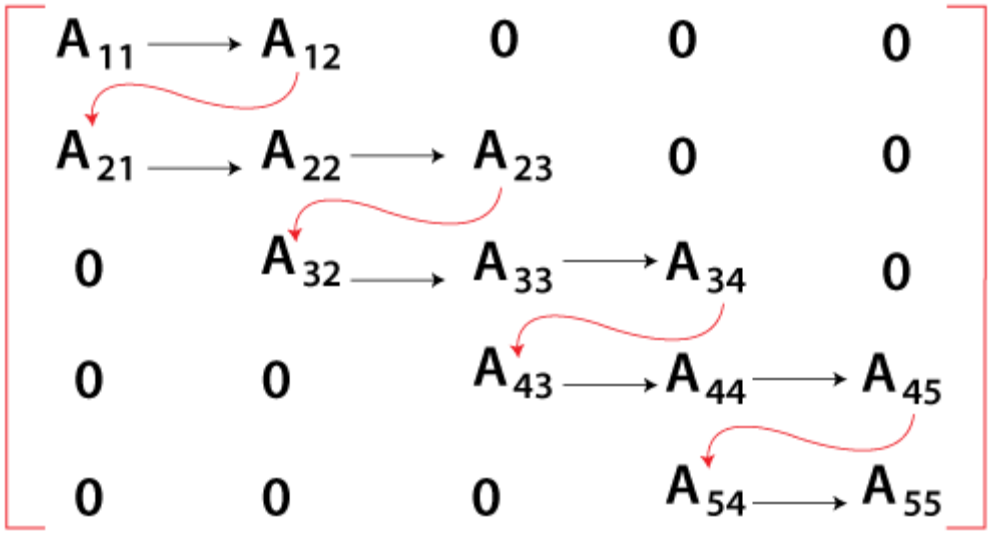
The tridiagonal regular sparse matrix where all non-zero elements lie on one of the three diagonals, the main diagonal above and below.

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

Tridiagonal Matrix

Storing Tri-diagonal regular sparse matrices

In a tri-diagonal regular sparse matrix, all the non-zero elements are stored in a 1-dimensional array row by row.



Representation of Tridiagonal matrix A[5, 5]

For 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}}

Here D [1] = A_{11}, D [2] = A_{12}, D [3] = A_{21}, D [4] = A_{22}, D [5] = A_{23}, D [6] = A_{32} ,D [7] = A_{33}, D [8] = A_{34} ,D [9] = A_{43} ,D [10] = A_{44} ,D [11] = A_{45} ,D [12] = A_{54}, D [13] = A_{55}

In order to calculate the total number of non-zero elements, we need to know the number of non-zero elements along the diagonal, above the diagonal and below the diagonal. The number of elements in a square matrix along the diagonal is **'n'** and above the diagonal is **(n-1)**. So the total number of non-zero elements in n-square matrix is:

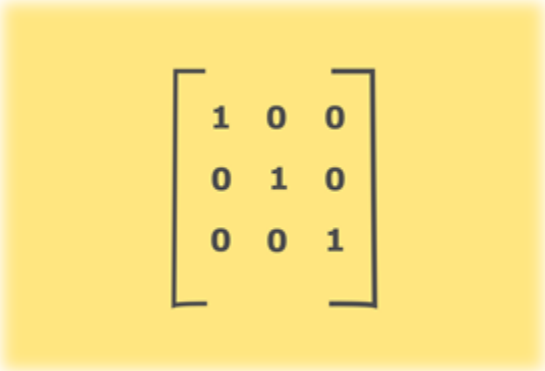
1. n + (n-1) + (n-1)
2. the diagonal Above the diagonal Below the diagonal

Therefore, the total number of elements for n-squre matrix = 3n-2

In our case, n = 5 so the total number of elements = 3 * 5 - 2 = 13

Program to determine whether a given matrix is an identity matrix

Identity Matrix



A matrix is said to be the identity matrix if it is the square matrix in which elements of principle diagonal are ones, and the rest of the elements are zeroes.

Algorithm

1. Declare and initialize a two-dimensional array a.
2. Calculate the number of rows and columns present in the array and store it in variables rows and columns respectively.
3. Initialize variable flag to true.
4. Check if given matrix has the same number of rows and columns(square matrix).
5. If not, print the error message "Matrix should be a square matrix."
6. If given matrix is a square matrix then, loop through the array and check if all the elements of main diagonal are 1 and the rest of the elements are 0.
7. If any of the condition is not satisfied, set the flag to false and break the loop.
8. If the flag is equal to true which implies given matrix is an identity matrix.
9. Else, given matrix is not an identity matrix.

---

identity matrix Solution in Python

```
#Initialize matrix a
a = [
        [1, 0, 0],
       [0, 1, 0],
        [0, 0, 1]
    ];

flag = True;

#Calculates number of rows and columns present in given matrix
rows = len(a);
cols = len(a[0]);

```

```
#Checks whether given matrix is a square matrix or not
if(rows != cols):
    print("Matrix should be a square matrix");

else:
    #Checks if diagonal elements are equal to 1 and rest of elements are 0
    for i in range(0, rows):
        for j in range(0, cols):
            if(i == j and a[i][j] != 1):
                flag = False;
                break;

            if(i != j and a[i][j] != 0):
                flag = False;
                break;


    if(flag):
        print("Given matrix is an identity matrix");
    else:
        print("Given matrix is not an identity matrix");
```

Output:

```
Given matrix is an identity matrix
```

identity matrix Solution in C

```
#include <stdio.h>
#include <stdbool.h>

int main()
{
    int rows, cols;
    bool flag = true;

    //Initialize matrix a
    int a[][3] = {
                    {1, 0, 0},
                   {0, 1, 0},
                    {0, 0, 1}
                };

    //Calculates number of rows and columns present in given matrix
    rows = (sizeof(a)/sizeof(a[0]));
    cols = (sizeof(a)/sizeof(a[0][0]))/rows;

    //Checks whether given matrix is a square matrix or not
    if(rows != cols){
        printf("Matrix should be a square matrix");
```

```
    }
    else{
        //Checks if diagonal elements are equal to 1 and rest of elements are 0
        for(int i = 0; i < rows; i++){
            for(int j = 0; j < cols; j++){
             if(i == j && a[i][j] != 1){
                  flag = false;
                break;
             }
             if(i != j && a[i][j] != 0){
                  flag = false;
                break;
             }
           }
        }

        if(flag)
           printf("Given matrix is an identity matrix");
        else
           printf("Given matrix is not an identity matrix");
    }
    return 0;
}
```

Output:

Given matrix is an identity matrix

identity matrix Solution in JAVA

```
public class IdentityMatrix
{
    public static void main(String[] args) {
        int rows, cols;
         boolean flag = true;

         //Initialize matrix a
         int a[][] = {
                        {1, 0, 0},
                       {0, 1, 0},
                        {0, 0, 1}
                    };

        //Calculates the number of rows and columns present in the given matrix

          rows = a.length;
         cols = a[0].length;

         //Checks whether given matrix is a square matrix or not
         if(rows != cols){
```

```
            System.out.println("Matrix should be a square matrix");
        }
        else {
            //Checks if diagonal elements are equal to 1 and rest of elements are 0
            for(int i = 0; i < rows; i++){
                for(int j = 0; j < cols; j++){
                    if(i == j && a[i][j] != 1){
                        flag = false;
                        break;
                    }
                    if(i != j && a[i][j] != 0){
                        flag = false;
                        break;
                    }
                    }
            }

            if(flag)
                System.out.println("Given matrix is an identity matrix");
            else
                System.out.println("Given matrix is not an identity matrix");
        }
    }
}
```

Output:

Given matrix is an identity matrix

identity matrix Solution in C#

```
using System;

public class IdentityMatrix
{
    public static void Main()
    {
        int rows, cols;
        Boolean flag = true;

        //Initialize matrix a
        int[,] a = {
                        {1, 0, 0},
                         {0, 1, 0},
                        {0, 0, 1}
                    };

        //Calculates the number of rows and columns present in the given matrix

          rows = a.GetLength(0);
```

```
20.         cols = a.GetLength(1);
21.
22.         //Checks whether given matrix is a square matrix or not
23.          if(rows != cols){
24.             Console.WriteLine("Matrix should be a square matrix");
25.         }
26.         else {
27.              //Checks if diagonal elements are equal to 1 and rest of elements are 0
28.             for(int i = 0; i < rows; i++){
29.                  for(int j = 0; j < cols; j++){
30.                   if(i == j && a[i,j] != 1){
31.                        flag = false;
32.                     break;
33.                   }
34.                   if(i != j && a[i,j] != 0){
35.                        flag = false;
36.                     break;
37.                   }
38.                }
39.             }
40.
41.             if(flag)
42.                Console.WriteLine("Given matrix is an identity matrix");
43.             else
44.                Console.WriteLine("Given matrix is not an identity matrix");
45.         }
46.     }
47. }
```

Output:

> *Given matrix is an identity matrix*

identity matrix Solution in PHP

```
1.  <!DOCTYPE html>
2.  <html>
3.  <body>
4.  <?php
5.  //Initialize matrix a
6.  $a = array(
7.               array(1, 0, 0),
8.              array(0, 1, 0),
9.               array(0, 0, 1)
10.           );
11.
12. $flag = true;
13.
14. //Calculates number of rows and columns present in given matrix
15. $rows = count($a);
```

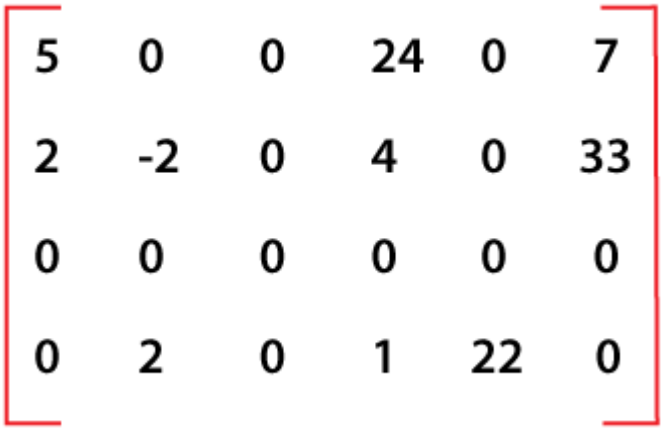
```
$cols = count($a[0]);

//Checks whether given matrix is a square matrix or not
if($rows != $cols){
    print("Matrix should be a square matrix <br>");
}
else {
    //Checks if diagonal elements are equal to 1 and rest of elements are 0
    for($i = 0; $i < $rows; $i++){
        for($j = 0; $j < $cols; $j++){
         if($i == $j && $a[$i][$j] != 1){
             $flag = false;
            break;
          }
         if($i != $j && $a[$i][$j] != 0){
             $flag = false;
            break;
          }
        }
    }

    if($flag)
        print("Given matrix is an identity matrix <br>");
    else
        print("Given matrix is not an identity matrix <br>");
}
?>
</body>
</html>
```

Output:

Given matrix is an identity matrix

Irregular sparse matrices

The irregular sparse matrices are the ones that behave an irregular or unstructured pattern of occurrences of non-zero elements. The following matrix is an irregular sparse matrix.



In the above diagram, there are several common storage schemes to store sparse matrices. Most of these schemes store all the non-zero elements of the matrix into a 1-dimensional array.

Storage schemes of Irregular sparse matrices:

In this scheme, we store all the non-zero elements of the matrix into a 1-dimensional array and use several auxiliary arrays to specify the location of non-zero elements in the sparse matrix. Among the several available storage schemes for storing irregular sparse matrices, the most common storage schemes are:

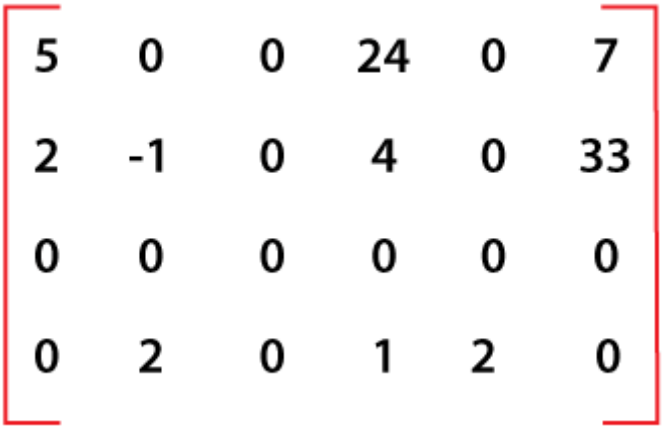
- Storage by Index Method
- Storage by Compressed Row Format

Storing irregular sparse matrix using storage by Index Method

Using storage by the index method, we can store irregular sparse matrices A by constructing three 1-Dimensional arrays AN, AI, and AJ, each having elements equal to a total number of non-zero elements.

- The array AN contains the non-zero elements stored row contiguously by row.
- AI contains the corresponding row number of non-zero elements A[I, J] in matrix A.
- AJ contains the corresponding column number of non-zero elements A[I, J] in matrix A.

Let us consider a 4 by 6 sparse matrix A as shown below:



5	0	0	24	0	7
2	-1	0	4	0	33
0	0	0	0	0	0
0	2	0	1	2	0

Out of 24 elements of this Sparse Matrix, only 10 elements are non-zero. These non-zero elements are:

1. A [1,1] = 5
2. A [1,4] = 24
3. A [1,6] = 7
4. A [2,1] = 2
5. A [2,2] = -1
6. A [2,4] = 4
7. A [2,6] = 33
8. A [4,2] = 2
9. A [4,4] = 1
10. A [4,5] = 22

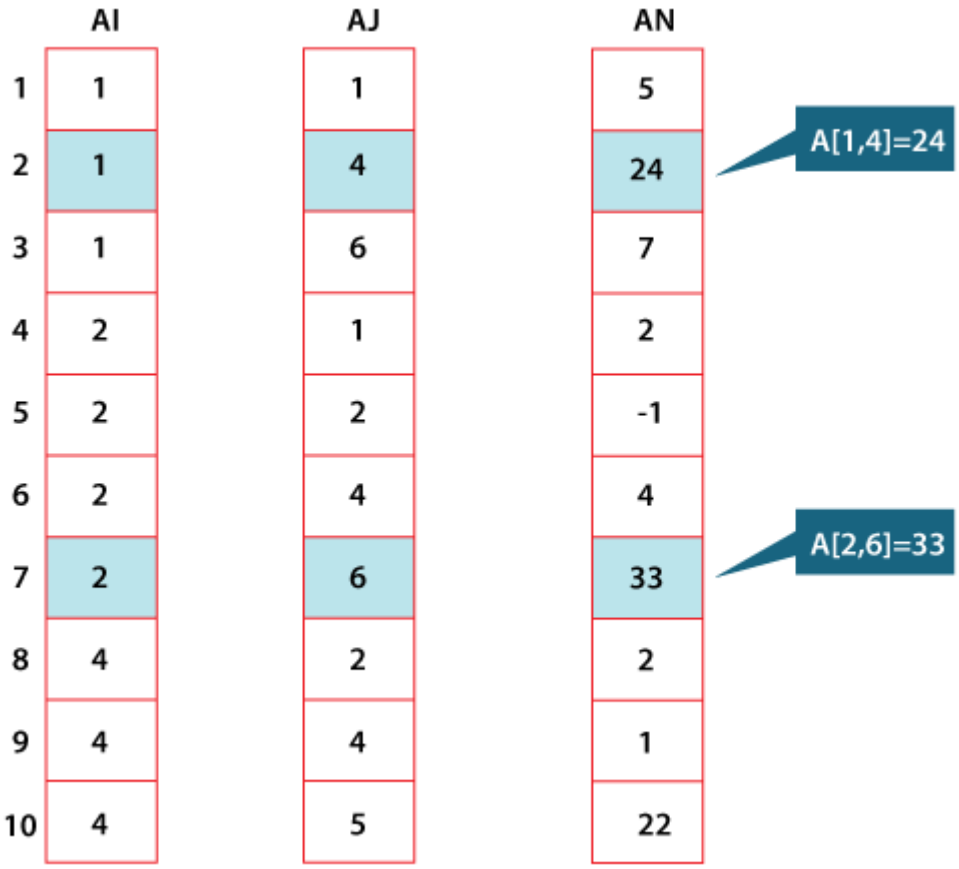


Figure: Representation of three 1 - D arrays AI, AJ, AN

To store it using the storage by index method. We construct three-dimensional arrays AN, SI, AJ, each having 10 elements (i.e., the total number of non-zero elements) as shown in the above diagram.

The array AN contains the 10 non-zero elements stored contiguously. The corresponding row number and column number of each non-zero element are stored in arrays AI and AJ, respectively.

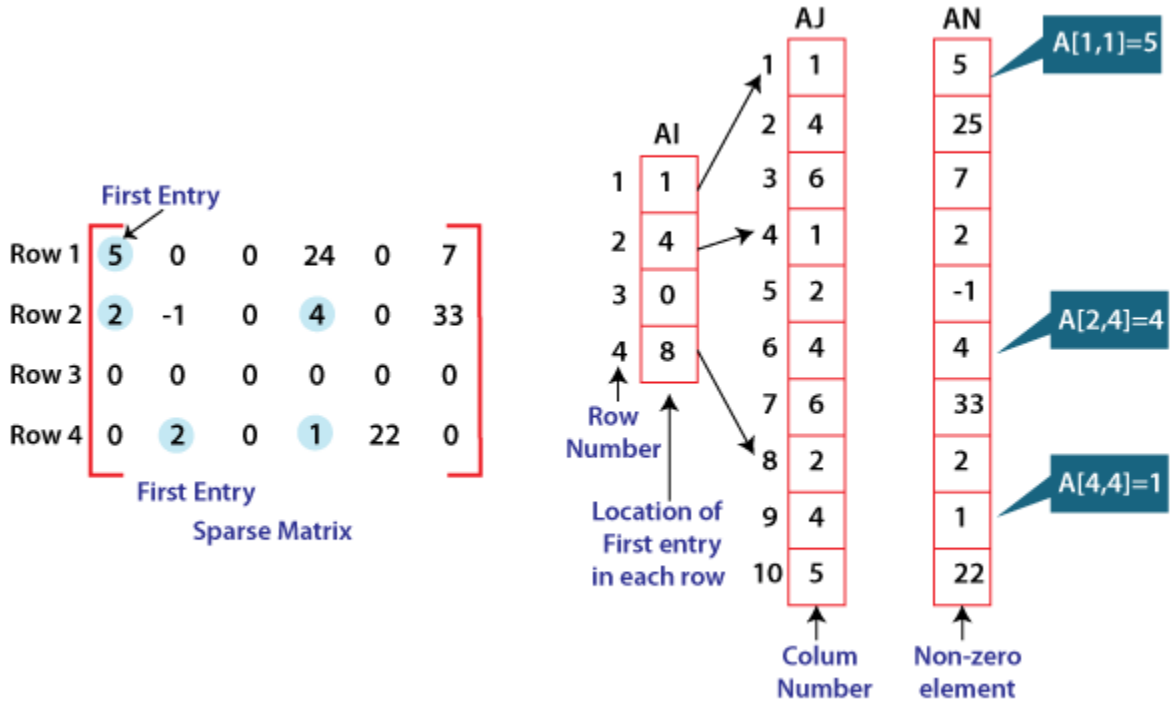
Storing irregular sparse matrix using the compressed row storage format

Another space-efficient and most widely used scheme for representing irregular sparse matrices is compressing row storage. In this scheme, we take three 1-Dimensional arrays AN, AI, and AJ. One-dimensional array AN contains all the values of non-zero elements taken in a row-wise fashion from the matrix. The next 1-dimensional array AJ of length equal to AN's length is constructed that contains the original column positions of the corresponding elements in AN. A 1-dimensional array AI with a length equal to several rows is constructed that stores the location of the first entry of non-zero elements in AN array.

For Example:

AI [2] = 4 indicates that the first non-zero elements in row 2 are stored in AN[4], which is 2.

To store it by compressing row storage format, we construct three 1-dimensional arrays AN, AI, and AJ, as shown in the following figure:



Compressed row storage format

The array AN contains 10 non-zero elements stored contiguously in a row-wise fashion. The corresponding column numbers are stored in 1-dimensional array AJ. The 1-dimensional array AI contains elements which is pointer to the first non-zero element of each row in array AJ and AN.