# Lab 2:
# Sparse Matrix and its Transpose

**➔ Aim:**

Accept the conventional matrix and convert it into Sparse Matrix. Implement Simple and Fast Transpose on the sparse matrix.

**➔ Theory:**

**● What is Matrix?**

Matrix is representation of data using rows and columns. For example, the dimension of a matrix below is 3 x 4, because there are three rows and four columns.

4 5 6 7
0 8 1 6
3 4 1 2

**● What is Sparse Matrix?**

A matrix in which most of the elements are zero is called as sparse matrix.
For example, the below matrix contains only 2 non zero elements and rest of the elements are zero therefore it is a sparse matrix.

0 0 0 0
0 5 0 0
0 0 0 0
0 0 7 0

**● Representation of Sparse Matrix**

The representation of sparse matrix saves a lot of memory space. It is Represented in the triplet form where there are three columns. In the first row, the first column represents the total number of rows, the second column represents the total number of columns and the third column represents the total number of non zero elements. The rest of the rows represents the columns as:
1. Row number 2. Column Number 3. Non zero element
For example: Consider the given sparse matrix.

0 0 0 0
0 5 0 0
0 0 0 0
0 0 7 0

If we count the rows and columns from 1 then the given matrix will be represented as follows:

**Row Col Val**
3 3 2
2 2 5
4 3 7

In the above example, the first row represents the total number of rows in represented matrix, total number of columns will be always 3 as it is represented in triplet form and total number of non zero elements. From second row the representation

is row number column no and the non zero element.

● **Pseudocode to Convert a Matrix into Sparse Matrix**

```
Void sparseMatrix()
{
for(i=0;i<row;i++)
{
for(j=0;j<col;j++)
{
if(matrix[i][j]!=0)
{
nonZero++;
sparse[k][0]=i;
sparse[k][1]=j;
sparse[k][2]=matrix[i][j];
k++;
}
}
}
sparse[0][0]=row;
sparse[0][1]=col;
sparse[0][2]=nonZero;
```

● **Transpose of Sparse Matrix**

The transpose of the sparse matrix is done by switching the rows and columns and arrange them in ascending order.

For example, Consider the following sparse matrix representation.

3 3 2
2 2 5
4 3 7

After transposing the sparse matrix we will get the following matrix:

3 3 2
2 2 5
3 4 5

In the above example, Before switching the rows and columns we first check the order of the column number from the second row. Here two is greater than three therefore the rows and columns will be switched.

To make the transpose of the sparse matrix there are two techniques:

1. Simple Transpose
2. Fast Transpose

The reason for using two techniques is the difference between the complexity simple transpose requires more complexity that of fast transpose, therefore fast transpose came into the picture.

**1. Simple Transpose**

In simple transpose we switch the rows and columns and and while switching they are arranged in ascending order by comparing column numbers.

● **Pseudocode for Simple Transpose**

```
void simpleTranspose()
{
int t=1,simple[10][10];
```

```
for(j=0;j<col;j++)
{
for(i=1;i<=nonZero;i++)
{
if(sparse[i][1]==j)
{
simple[t][1]=sparse[i][0]
simple[t][0]=sparse[i][1];
simple[t][2]=sparse[i][2];
t++;
}
}
}
simple[0][0]=sparse[0][1];
simple[0][1]=sparse[0][0];
simple[0][2]=sparse[0][2];
}
```

**Complexity:** The Simple Transpose requires complexity $O(n2)$

**2. Fast Transpose**

In fast transpose there are two arrays first is frequency array and second Is position or index array. Frequency array stores the values of columns present in the sparse matrix. Position array stores the position of the first occurrence of the values present in the column of sparse matrix. It is then incremented in the program.

- **Pseudocode for Simple Transpose**

```
void fastTranspose()
{
int i,j,freq[nonZero],pos[nonZero],fast[10][10];
for(i=0;i<=nonZero;i++)
{
freq[i]=0;
}
for(i=1;i<=nonZero;i++)
{
freq[spm[i][1]]++;
}
pos[0]=1;
for(i=1;i<=nonZero;i++)
{
pos[i]=(pos[i-1])+(freq[i-1]);
}
for(i=0;i<=nonZero;i++)
{
j=pos[sparse[i][1]];
fast[j][0]=sparse[i][1];
fast[j][1]=sparse[i][0];
fast[j][2]=sparse[i][2];
```

```
pos[sparse[i][1]]++;
}
fast[0][0]=sparse[0][1];
fast[0][1]=sparse[0][0];
fast[0][2]=sparse[0][2];
}
```

**Complexity:** Fast Transpose requires complexity of $O(n)$

**Program:**

```
#include<iostream.h>

#include<conio.h>

void main()

{

clrscr();

int a[10][10],sp[10][10],m,n,i,j,t=0;

cout<<"Enter no of rows and  columns :";

cin>>m>>n;

cout<<"Enter matrix elements:";

for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

{

cin>>a[i][j];

}

}

for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

{

cout<<a[i][j]<<" ";
```

```
}
cout<<"\n";
}
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
if(a[i][j]!=0)
{
t++;
sp[t][0]=i;
sp[t][1]=j;
sp[t][2]=a[i][j];
}
}
}
sp[0][0]=m;
sp[0][1]=n;
sp[0][2]=t;
cout<<"_____Sparse Matrix_____"<<"\n";
t++;
for(i=0;i<t;i++)
{
for(j=0;j<3;j++)
{
cout<<sp[i][j]<<" ";
}
```

```
cout<<"\n";
}
int n1,currentb,b[10][10];
n1=sp[0][2];
b[0][0]=sp[0][1];
b[0][1]=sp[0][0];
b[0][2]=n1;
if(n1>0)
{
currentb=1;
{
for(i=0;i<sp[0][1];i++)
{
for(j=1;j<=n1;j++)
{
if(sp[j][1]==i)
{
b[currentb][0]=sp[j][1];
b[currentb][1]=sp[j][0];
b[currentb][2]=sp[j][2];
currentb++;
}
}
}
}
}
```

```
cout<<"_____Simple Transpsose_____"<<"\n";
for(i=0;i<t;i++)
{
for(j=0;j<3;j++)
{
cout<<b[i][j]<<" ";
}
cout<<"\n";
}
int freq[10],pos[10],fast[10][10];
for(i=0;i<=t;i++)
{
freq[i]=0;
}
for(i=1;i<t;i++)
{
freq[sp[i][1]]++;
}
pos[0]=1;
for(i=1;i<t;i++)
{
pos[i]=(pos[i-1])+(freq[i-1]);
}
for(i=1;i<=t;i++)
{
j=pos[sp[i][1]];
fast[j][0]=sp[i][1];
```

```
fast[j][1]=sp[i][0];
fast[j][2]=sp[i][2];
pos[sp[i][1]]++;
}
fast[0][0]=sp[0][1];
fast[0][1]=sp[0][0];
fast[0][2]=sp[0][2];
cout<<"__________Fast Transpose___________"<<endl;
for(i=0;i<=t;i++)
{
for(j=0;j<3;j++)
{
cout<<fast[i][j]<<" ";
}
cout<<"\n";
}

getch();

}
```

```
Enter no of rows and  columns :3
3
Enter matrix elements:1 0 1 2 0 2 3 0 3_
```

```
3 3 6
0 0 1
0 2 1
1 0 2
1 2 2
2 0 3
2 2 3
__________Simple Transpsose__________
3 3 6
0 0 1
0 1 2
0 2 3
2 0 1
2 1 2
2 2 3
_________Fast Transpose__________
3 3 6
0 0 1
0 1 2
0 2 3
0 0 1
2 1 2
2 2 3
```

**➔ Conclusion:**

In this way we have successfully studies how to represent the sparse matrix and to implement simple and fast transpose of the sparse matrix. Also we have seen that the complexity required by fast transpose is less than the complexity required by the simple transpose.