Devanshu Surana
1032210755
Panel-C Batch-C1
FDS

Lab Assignment - 2

Problem statement

Write a program for sparse matrix realization and operations on-it - simple Transpose, fast transpose

Objective

1. To study the concept of sparse matrix, how it is stored and displayed
2. To understand the implementation of sparse matrix operations - simple and fast transpose

Theory:

Sparse Matrix = A sparse matrix is a matrix in which many or most of the elements have a value of zero. This is in contrast to a dense matrix, where many or most of the elements have a non-zero value

- Need for conversion of sparse matrix to its Compact form.

Representing a sparse matrix by a 2D array leads to wastage of lots of ~~money~~ memory as zeroes in the matrix are of no use in most of the cases.

To avoid such wastage, we can store only non-zero elements. Only storing non-zero elements reduces transversal time and storage space.

- Advantage of ~~fast~~ fast transpose over simple transpose.

→ Time complexity of fast transpose (columns + elements) is less than that of simple transpose (columns X elements)

Implementation

- Platform
- 64-bit open-source Linux or its derivatives
- Open Source C programming tool line gcc/Eclipse editor.

PSEUDO code:

Conversion to compact form

```
void compact (int a[10][10], int c[10][3], int m, int n)
{
    int i,j;
    int k=1;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
        {
            if (a[i][j] != 0)
            {
                c[k][0] = i;
```

```
            c[k][1]=j;
            c[k][2]= a[i][j];
            k++;
            }
        }
    }
    c[0][0] = m;
    c[0][1] = n;
    c[0][2] = k-1;
```

Simple transpose

```
void simpletranspose (int c[10][3],int [10][3] )
{
    int i,j;
    int k = 1;
for (j=1;j<=c[0][1],j++)
{
  for (j=1;j<=c[0][2];j++)
  {
     if (c[j][1]==i)
     {
       t[k][0]=i;
       t[k][1] = c[j][0];
       t[k][2]= c[j][2];
       }
     }
}
 t[0][0] = c[0][1];
```

```
t[0][1] = c[0][0];
t[0][2] = c[0][2]
```

Fast transpose:

```
void fasttranspose (int c[10][3], int ft[10][3] )
{
  int nterm[10], npos[10], i, local;
for (i=0; i<c[0][1]; i++)
 { nterm[i] = 0;
 }
 for (i=1; i<=c[0][2]; i++)
{
   nterm[c[i][1]]++;
}
npos[0] = 1;
 for (i=1; i<c[0][1]; i++)
{
npos[i] = npos[i-1] + nterm[i-1];
}
for (i=1; i<=c[0][2]; i++)
{
  local = npos[c[i][1]]
 ft[local][0] = c[i][1];
 ft[local][1] = c[i][1];
 ft[local][2] = c[i][2];
  npos[c[i][1]]++;
}
```

```
ft[0][0] = c[0][1];
ft[0][1] = c[0][0];
ft[0][2] = c[0][2];
}
```

Time Complexity:
i) For simple transpose = o(n*t)
ii) For fast transpose = o(n+t)

Conclusion:
Implemented Sparse matrix operation assignment. This system is able to perform different operations on sparse matrix such as simple and fast transpose and their time complexity.

FAQ's
Ans - 1) A sparse matrix is a matrix that mostly ~~comprises~~ comprises zeroes.

Applications:
i) Computing/solving partial differential using finite element method.
ii) Optimisation problems
iii) Structural engineering

Ans - 2) It can be presented in 2 ways :-

i) Array Representation

- 2D array is used to represent a sparse matrix in which there are 3 rows ~~and~~ named : Row, Column and value :

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

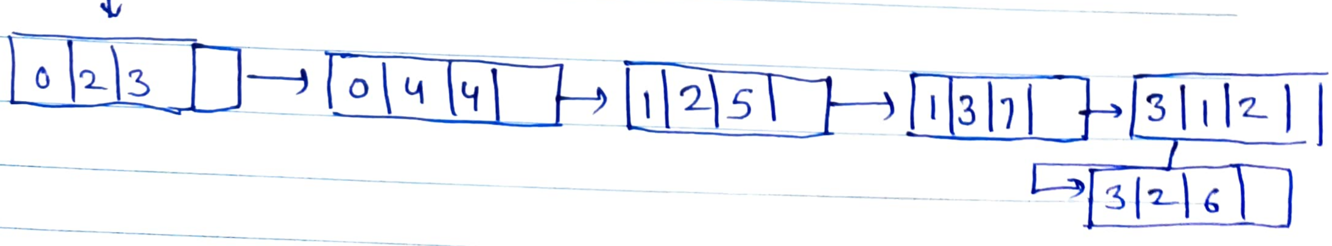
Row .	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

```
Eg: int sp[4][6] = {
        {0,0,3,0 4},
        {0,0,5,7,0},
        {0,0,0,0,4},
        {0,2,6,0,0}
    };
```

ii) Linked list

- In linked list each node has 4 fields - Row, column, value, Next ,Node (address)

Eg: Start



- Simple transpose

First the sparse matrix is converted into its compact form for simple transpose, column numbers of the non-zero elements in the sparse matrix are considered. we check if 1st entry in column 1 of compact form is zero for all elements, and write down those which are in format (column, row and value) If there are no zero in column 1 we check for 1, and continue this process till the column number is equal to no. of columns in sparse matrix.

Ex. $$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 8 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix} \rightarrow \begin{bmatrix} 5 & 4 & 5 \\ 0 & 1 & 1 \\ 1 & 1 & 5 \\ 2 & 0 & 8 \\ 3 & 2 & 7 \\ 4 & 3 & 10 \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 5 & 5 \\ 0 & 2 & 8 \\ 1 & 0 & 1 \\ 1 & 1 & 5 \\ 2 & 3 & 7 \\ 3 & 4 & 10 \end{bmatrix}$$

Sparse matrix — Compact matrix — Simple transpose

Fast transpose

1st we calculate how many elements are there in all columns. Based on that we determine the starting position from where the elements having a certain column number in the sparse matrix will have. Then all the elements are entered starting from 0 in 'column' in (column, row and value) format.

Ex. $\begin{bmatrix} 1 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 \\ 0 & 8 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 5 & 5 & 7 \\ 0 & 0 & 1 \\ 0 & 4 & 3 \\ 1 & 3 & 5 \\ 2 & 1 & 6 \\ 2 & 2 & 7 \\ 3 & 2 & 9 \\ 4 & 1 & 8 \end{bmatrix}$

column : 0 1 2 3 4
n-terms : 1 2 2 1 1
→ Starting pos : 1 2 4 6 7

Fast transpose

$\begin{bmatrix} 5 & 5 & 7 \\ 0 & 0 & 1 \\ 1 & 2 & 6 \\ 1 & 4 & 8 \\ 2 & 2 & 7 \\ 2 & 3 & 9 \\ 3 & 1 & 5 \\ 4 & 0 & 3 \end{bmatrix}$

Ans 3) $M_1 = \begin{bmatrix} 4 & 5 & 6 \\ 0 & 3 & 5 \\ 1 & 3 & 8 \\ 1 & 4 & 45 \\ 2 & 3 & 4 \\ 3 & 2 & 45 \\ 4 & 1 & 2 \end{bmatrix}$ $M_2 = \begin{bmatrix} 4 & 5 & 6 \\ 0 & 3 & 7 \\ 0 & 4 & 6 \\ 1 & 4 & 4 \\ 2 & 1 & 8 \\ 3 & 2 & 45 \\ 4 & 4 & 21 \end{bmatrix}$

(Result) $M_1 + M_2 = \begin{bmatrix} 4 & 5 & 10 \\ 0 & 3 & 12 \\ 0 & 4 & 6 \\ 1 & 3 & 8 \\ 1 & 4 & 4 \\ 1 & 4 & 45 \\ 2 & 1 & 8 \\ 2 & 3 & 4 \\ 3 & 2 & 45 \\ 4 & 1 & 2 \\ 4 & 4 & 21 \end{bmatrix}$

$$\begin{bmatrix} 5 & 4 & 6 \\ 1 & 4 & 2 \\ 2 & 3 & 45 \\ 3 & 0 & 5 \\ 3 & 1 & 8 \\ 3 & 2 & 4 \\ 4 & 1 & 45 \end{bmatrix}$$

M_1 Simple Transpose
M_1 Fast Transpose

$$\begin{bmatrix} 5 & 4 & 6 \\ 1 & 2 & 0 \\ 2 & 3 & 45 \\ 3 & 0 & 7 \\ 4 & 0 & 6 \\ 4 & 1 & 4 \\ 4 & 4 & 21 \end{bmatrix}$$

M_2 Simple Transpose
M_2 Fast Transpose

num terms(m_1):	0	1	2	3	4
Starting position (M_1)	0	1	1	3	1
	0	1	2	3	6
num terms (M_2):	0	1	2	3	4
Starting position (M_2):	0	1	1	1	3
	0	1	2	3	4

2/11/22