**AIM:**

Accept conventional matrix and convert it into sparse matrix. Implement simple and fast transpose algorithms on sparse matrix.

**THEORY:**

In computer programming, a matrix can be defined with a 2-dimensional array. Any array with 'm' columns and 'n' rows represents a m X n matrix. There may be a situation in which a matrix contains more number of ZERO values than NON-ZERO values. Such matrix is known as sparse matrix.

When a sparse matrix is represented with a 2-dimensional array, we waste a lot of space to represent that matrix. For example, consider a matrix of size 100 X 100 containing only 10 non-zero elements. In this matrix, only 10 spaces are filled with non-zero values and remaining spaces of the matrix are filled with zero. That means, totally we allocate 100 X 100 X 2 = 20000 bytes of space to store this integer matrix. And to access these 10 non-zero elements we have to make scanning for 10000 times. To make it simple we use the following sparse matrix representation.

**TRANSPOSE OF MATRIX:**

Transpose of a [matrix](https://en.wikipedia.org/wiki/Matrix_(mathematics)) is an operator which flips a matrix over its diagonal, that is it switches the row and column indices of the matrix by producing another matrix

**SOURCE CODE:**

/\*

Programmer: ANOM DEVGUN

Implemented finding of sparse mat and transpose

\*/

#include<iostream>

using namespace std;

class mat

{

private :

int a[20][20],b[20][20],rma[20][20],ft[20][20],i,d,j,cma,noterms,nxt,Term,noc,ctr=1,cou=0;

public:

void input(int,int);

void spacal(int,int);

void simtra();

void fatra();

};

void mat::input(int l,int m) //TO INPUT

{

cout<<"Enter Elements Of Matrix\n";

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

{

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

cin>>a[i][j];

}

cout<<"The Entered Matrix is\n";

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

{

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

{

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

}

cout<<"\n";

}

}

void mat::spacal(int l, int m) //FIND SPARSE

{

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

{

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

{

if(a[i][j] != 0)

{

b[ctr][0]=i;

b[ctr][1]=j;

b[ctr][2]=a[i][j];

ctr++;

}

}

}

cma=ctr;

b[0][0]=l;

b[0][1]=m;

b[0][2]=ctr-1;

cout<<"The sparse matrix of the given matrix is : \n";

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

{

cout<<"\n";

for(j=0;j<3;j++)

cout<<b[i][j]<<" ";

}

}

void mat::simtra() //SIMPLE TRANSPOSE

{

rma[0][0]=b[0][1];

rma[0][1]=b[0][0];

rma[0][2]=b[0][2];

noterms=b[0][2];

noc=b[0][1];

if(b[0][2] > 1)

{

nxt=1;

for(d=0;d<noc;d++)

{

for(Term=1;Term<=noterms;Term++)

{

if(b[Term][1]== d)

{

rma[nxt][0]=b[Term][1];

rma[nxt][1]=b[Term][0];

rma[nxt][2]=b[Term][2];

nxt++;

}

}

}

}

for(i=0;i<=rma[0][2];i++)

{

cout<<"\n";

for(j=0;j<3;j++)

cout<<rma[i][j]<<" ";

}

}

void mat :: fatra() //FAST TRANSPOSE

{

int o=0,p=0,cnt[10],pos[10];

for(i=0;i<10;i++)

{

cnt[i]=0;

pos[i]=0;

}

for(i=0;i<20;i++)

{

for(j=0;j<20;j++)

{

ft[i][j]=0;

}

}

for(i=0;i<=b[0][2];i++)

{

o=b[i][1];

cnt[o]++;

}

pos[0]=1;

for(i=1;i<b[0][1];i++)

{

pos[i]=pos[i-1]+cnt[i-1];

}

for(i=1;i<=b[0][2];i++)

{

o=b[i][1];

p=pos[o];

pos[o]++;

ft[p][0]=b[i][1];

ft[p][1]=b[i][0];

ft[p][2]=b[i][2];

}

ft[0][0]=b[0][0];

ft[0][1]=b[0][1];

ft[0][2]=b[0][2];

for(i=0;i<=ft[0][2];i++)

{

cout<<"\n";

for(j=0;j<3;j++)

cout<<ft[i][j]<<" ";

}

}

int main() //MAIN FUNCTION

{

mat obj;

int k,n;

char ch;

cout<<"Enter The Rows and Columns of Matrix \n";

cin>>k>>n;

obj.input(k,n);

obj.spacal(k,n);

while(1){

cout<<"\n1)For Simple transpose\n";

cout<<"2)For Fast Transpose\n";

cout<<"3)To Exit.\n";

cin>>ch;

switch(ch)

{

case '1': obj.simtra();

break;

case '2': obj.fatra();

break;

case '3':cout<<"\nNow Exiting.\n";

exit(0);

break;

default:cout<<"Incorrect choice, no corresponding option\n";

}

}

return 0;

}

**CONCLUSION:**

The complexity of simple transpose algorithm is: O(n^2)

The complexity of fast transpose algorithm is:O(n)