**Koneru Lakshmaiah Education Foundation**

**(Deemed to be University)**

**FRESHMAN ENGINEERING DEPARTMENT**

**A Project Based Lab Report**

**On**

**SPARSE MATRIX**

**SUBMITTED BY:**

I.D NUMBER NAME

180030646 P.SIVA

**UNDER THE GUIDANCE OF**

**DR .P.Siva kumar**

**Associate Professor**



**KL UNIVERSITY**

Green fields, Vaddeswaram – 522 502

Guntur Dt., AP, India.

**DEPARTMENT OF BASIC ENGINEERING SCIENCES-1**



**CERTIFICATE**

This is to certify that the project based laboratory report entitled “SPARSE MATRIX” submitted by **K.MOHAN SAI** bearing Regd. No. 180030360 to the **Department of Basic Engineering Sciences-1, KL University** in partial fulfillment of the requirements for the completion of a project based Laboratory in “TECHNICAL SKILLS-2(CODING)”course in I B Tech 2nd Semester, is a bonafide record of the work carried out by him/her under my supervision during the academic year 2018 – 2 019.

PROJECT SUPERVISOR HEAD OF THE DEPARTMENT

DR. P. SIVA KUMAR T.VAMSIDHAR

**ACKNOWLEDGEMENTS**

It is great pleasure for me to express my gratitude to our honorable President **Sri. KoneruSatyanarayana**, for giving the opportunity and platform with facilities in accomplishing the project based laboratory report.

I express the sincere gratitude to our principal **ProfDr. N.Venkataram** for his administration towards our academic growth.

I express sincere gratitude to HOD-BES-1 **T.VAMSIDHAR**for his leadership and constant motivation provided in successful completion of our academic semester. I record it as my privilege to deeply thank for providing us the efficient faculty and facilities to make our ideas into reality.

I express my sincere thanks to our project supervisor <name> for his/her novel association of ideas, encouragement, appreciation and intellectual zeal which motivated us to venture this project successfully.

Finally, it is pleased to acknowledge the indebtedness to all those who devoted themselves directly or indirectly to make this project report success.

Name Regd.no

P.SIVA 180030646

**ABSTRACT**

**Our project is based on the sparse matrix . Sparse matrix is matrix in which most of the elements are zeroes . We use linked list to represent the sparse matrix . We represent row , column , value of a sparse matrix using linked list.**

**Sparse matrices also have significant advantages in terms of computational efficiency. Unlike operations with full matrices, operations with sparse matrices do not perform unnecessary low-level arithmetic, such as zero-adds (x+0 is always x). The resulting efficiencies can lead to dramatic improvements in execution time for programs working with large amounts of sparse data**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TITLE** | **PAGE NO** |
| 1 | Introduction | 6 |
| 2 | Aim of the Project | 7 |
| 2.1 | Advantages & Disadvantages | 7 |
| 2.2 | Future Implementation | 7 |
| 3 | Software & Hardware Details | 8 |
| 4 | Data Flow Diagram | 9 |
| 5 | Implementation | 11-19 |
| 6 | Algorithm | 10 |
| 7 | Integration and System Testing | 20-21 |
| 8 | Conclusion | 22 |

**1.INTRODUCTION**

A [matrix](https://www.geeksforgeeks.org/data-structures/#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.

* **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.

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

2.AIM

**Representation of sparse matrix using linked list**

**Advantages:-**

**Using sparse matrices to store data that contains a large number of zero-valued elements can both save a significant amount of memory and speed up the processing of that data. sparse is an attribute that you can assign to any two-dimensional MATLAB® matrix that is composed of double or logical elements.**

**The sparse attribute allows MATLAB to:**

**\* Store only the nonzero elements of the matrix, together with their indices.**

**\* Reduce computation time by eliminating operations on zero elements.**

**Disadvantages:-**

**A matrix is an array of numbers in brackets that contain rows (horizontal) and columns (vertical). If there are n rows and m columns, we say the matrix is nxm. Matrices can contain things other than numbers, such as polynomials. Furthermore, those with numbers may contain real or complex numbers**

**Future enhancements:-**

By implementation of spares matrix we can minimize the space of the matrix.

**3. SYSTEM REQUIREMENTS**

* **SOFTWARE REQUIREMENTS:**

The major software requirements of the project are as follows:

Language : Turbo-C

Operating system**:**Windows Xp or later.

* **HARDWARE REQUIREMENTS:**

The hardware requirements that map towards the software are as follows:

RAM : 8GB

PROCESSOR: 32 BIT

**4.DATA FLOW DIAGRAM**

Enter 1st matrix

Enter 2nd matrix

Print menu

Enter your choice

If choice ==1

addition

If choice ==2

substraction

If choice ==3

multiplication

If choice ==4

exit

Enter valid input

**5.ALGORITHM**

**STEP 1: start**

**STEP 2: enter the first matrix**

**STEP 3: enter the second matrix**

**STEP 4: print menu (list of operations to be performed)**

**STEP 5: enter your choice**

**STEP 6: if choice is 1**

**Addition**

**Go to step 5**

**STEP 7: if choice is 2**

**substraction**

**Go to step 5**

**STEP 8: if choice is 3**

**multiplication**

**Go to step 5**

**STEP 9: if choice is 4**

**Exit**

**STEP 10: else print invalid input**

**STEP 11: stop**

**6. IMPLEMENTATION**

#include<stdio.h>

#include<stdlib.h>

void create1(int,int,int);

void create2(int,int,int);

void display1();

void display2();

void createa(int,int,int);

void creates(int,int,int);

void displaya();

void displays();

void createm(int,int,int);

void displaym();

typedef struct node

{

int value;

int row;

int col;

struct node \*nxt;

}m;

m \*head1=NULL,\*head2=NULL,\*tail1=NULL,\*tail2=NULL,\*r1,\*r2,\*r,\*ra,\*temp,\*s,\*heada=NULL,\*taila=NULL,\*tails=NULL,\*heads=NULL,\*rs,\*headm=NULL,\*tailm=NULL,\*rm;

int main()

{

int m1,m2,n1,n2,i,j,ch=0,k,sum=0;

printf("ENTER YOUR FIRST MATRIX SIZE\n");

scanf("%d %d",&m1,&n1);

printf("ENTER YOUR SECOND MATRIX SIZE\n");

scanf("%d %d",&m2,&n2);

int a[m1][n1],b[m2][n2],c[m1][n1],m[m1][n1];

unsigned int s[m1][n1];

printf("ENTER YOUR FIRST MATRIX ELEMENTS\n");

for(i=0;i<m1;i++)

{

for(j=0;j<n1;j++)

{

scanf("%d",&a[i][j]);

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

{

create1(a[i][j],i,j);

}

}

}

printf("ENTER YOUR SECOND MATRIX ELEMENTS\n");

for(i=0;i<m2;i++)

{

for(j=0;j<n2;j++)

{

scanf("%d",&b[i][j]);

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

{

create2(b[i][j],i,j);

}

}

}

printf("YOUR FIRST MATRIX IS\n");

display1();

printf("YOUR SECOND MATRIX IS\n");

display2();

do

{

printf("ENTER YOUR CHOICE OF OPERATION\n");

printf("1.ADDITION\n");

printf("2.SUBTRACTION\n");

printf("3.MULTIPLICATION\n");

printf("4.EXIT\n");

scanf("%d",&ch);

switch(ch)

{

    case 1:

for(i=0;i<m1;i++)

{

    for(j=0;j<n1;j++)

    {

        c[i][j]=a[i][j]+b[i][j];

        if(c[i][j]!=0)

        {

            createa(c[i][j],i,j);

        }

    }

}

displaya();

break;

case 2:

for(i=0;i<m1;i++)

{

    for(j=0;j<n1;j++)

    {

        s[i][j]=a[i][j]-b[i][j];

        if(s[i][j]!=0)

        creates(s[i][j],i,j);

    }

}

displays();

break;

case 3:

if(m1==n2 && m2==n1)

    {

       for (i = 0; i < m1; i++) {

      for (j = 0; j < n2; j++) {

        for (k = 0; k < m2; k++) {

          sum = sum + a[i][k]\*b[k][j];

        }

        m[i][j] = sum;

        sum = 0;

      }

    }

    }

    else

    printf("THE MATRIX MULTIPLICATION IS NOT POSSIBLE\n");

    for(i=0;i<m1;i++)

    {

        for(j=0;j<n1;j++)

        {

            if(m[i][j]!=0)

            createm(m[i][j],i,j);

        }

    }

    displaym();

    break;

    case 4:exit(0);

    break;

default : printf("WRONG OPTION\n");

}

}while(ch>=1 && ch<=4);

}

void create1(int x,int mr,int mc)

{

r1=(m\*) malloc(sizeof(m));

r1->value=x;

r1->row=mr;

r1->col=mc;

r1->nxt=NULL;

if(head1==NULL)

{

head1=tail1=r1;

}

else

{

tail1->nxt=r1;

tail1=r1;

}

}

void create2(int x,int mr,int mc)

{

r2=(m\*) malloc(sizeof(m));

r2->value=x;

r2->row=mr;

r2->col=mc;

r2->nxt=NULL;

if(head2==NULL)

{

head2=tail2=r2;

}

else

{

tail2->nxt=r2;

tail2=r2;

}

}

void display1()

{

temp=r=s=head1;

printf("row    :");

while(temp!=NULL)

{

printf("%d ",temp->row);

temp=temp->nxt;

}

printf("\n");

printf("column :");

while(r!=NULL)

{

printf("%d ",r->col);

r=r->nxt;

}

printf("\n");

printf("value  :");

while(s!=NULL)

{

printf("%d ",s->value);

s=s->nxt;

}

printf("\n");

}

void display2()

{

temp=r=s=head2;

printf("row    :");

while(temp!=NULL)

{

printf("%d ",temp->row);

temp=temp->nxt;

}

printf("\n");

printf("column :");

while(r!=NULL)

{

printf("%d ",r->col);

r=r->nxt;

}

printf("\n");

printf("value  :");

while(s!=NULL)

{

printf("%d ",s->value);

s=s->nxt;

}

printf("\n");

}

void createa(int x,int mr,int mc)

{

    ra=(m\*) malloc(sizeof(m));

    ra->value=x;

    ra->row=mr;

    ra->col=mc;

    ra->nxt=NULL;

    if(heada==NULL)

    {

        heada=taila=ra;

    }

    else

    {

        taila->nxt=ra;

        taila=ra;

    }

}

void displaya()

{

    temp=r=s=heada;

    printf("YOUR NEW MATRIX AFTER ADDITON OPERATION IS\n");

    printf("row    :");

    while(temp!=NULL)

    {

        printf("%d ",temp->row);

        temp=temp->nxt;

    }

    printf("\n");

    printf("column :");

    while(r!=NULL)

    {

        printf("%d ",r->col);

        r=r->nxt;

    }

    printf("\n");

    printf("value  :");

    while(s!=NULL)

    {

        printf("%d ",s->value);

        s=s->nxt;

    }

    printf("\n");

}

void creates(int x,int mr,int mc)

{

    rs=(m\*)malloc(sizeof(m));

    rs->value=x;

    rs->row=mr;

    rs->col=mc;

    rs->nxt=NULL;

    if(heads==NULL)

    {

        heads=tails=rs;

    }

    else

    {

        tails->nxt=rs;

        tails=rs;

    }

}

void displays()

{

    temp=r=s=heads;

    printf("YOUR NEW MATRIX AFTER SUBTRACTION\n");

    printf("row    :");

    while(temp!=NULL)

    {

        printf("%d ",temp->row);

        temp=temp->nxt;

    }

    printf("\n");

    printf("column :");

    while(r!=NULL)

    {

        printf("%d ",r->col);

        r=r->nxt;

    }

    printf("\n");

    printf("value  :");

    while(s!=NULL)

    {

        printf("%d ",s->value);

        s=s->nxt;

    }

    printf("\n");

}

void createm(int x,int mr,int mc)

{

    rm=(m\*)malloc(sizeof(m));

    rm->value=x;

    rm->row=mr;

    rm->col=mc;

    rm->nxt=NULL;

    if(headm==NULL)

    headm=tailm=rm;

    else

    {

        tailm->nxt=rm;

        tailm=rm;

    }

}

void displaym()

{

    temp=r=s=headm;

    printf("YOUR NEW MATRIX AFTER MULTIPLICATION OPERATION\n");

    printf("row    :");

    while(temp!=NULL)

    {

        printf("%d ",temp->row);

        temp=temp->nxt;

    }

    printf("\n");

    printf("column :");

    while(r!=NULL)

    {

        printf("%d ",r->col);

        r=r->nxt;

    }

    printf("\n");

    printf("value  :");

    while(s!=NULL)

    {

        printf("%d ",s->value);

        s=s->nxt;

    }

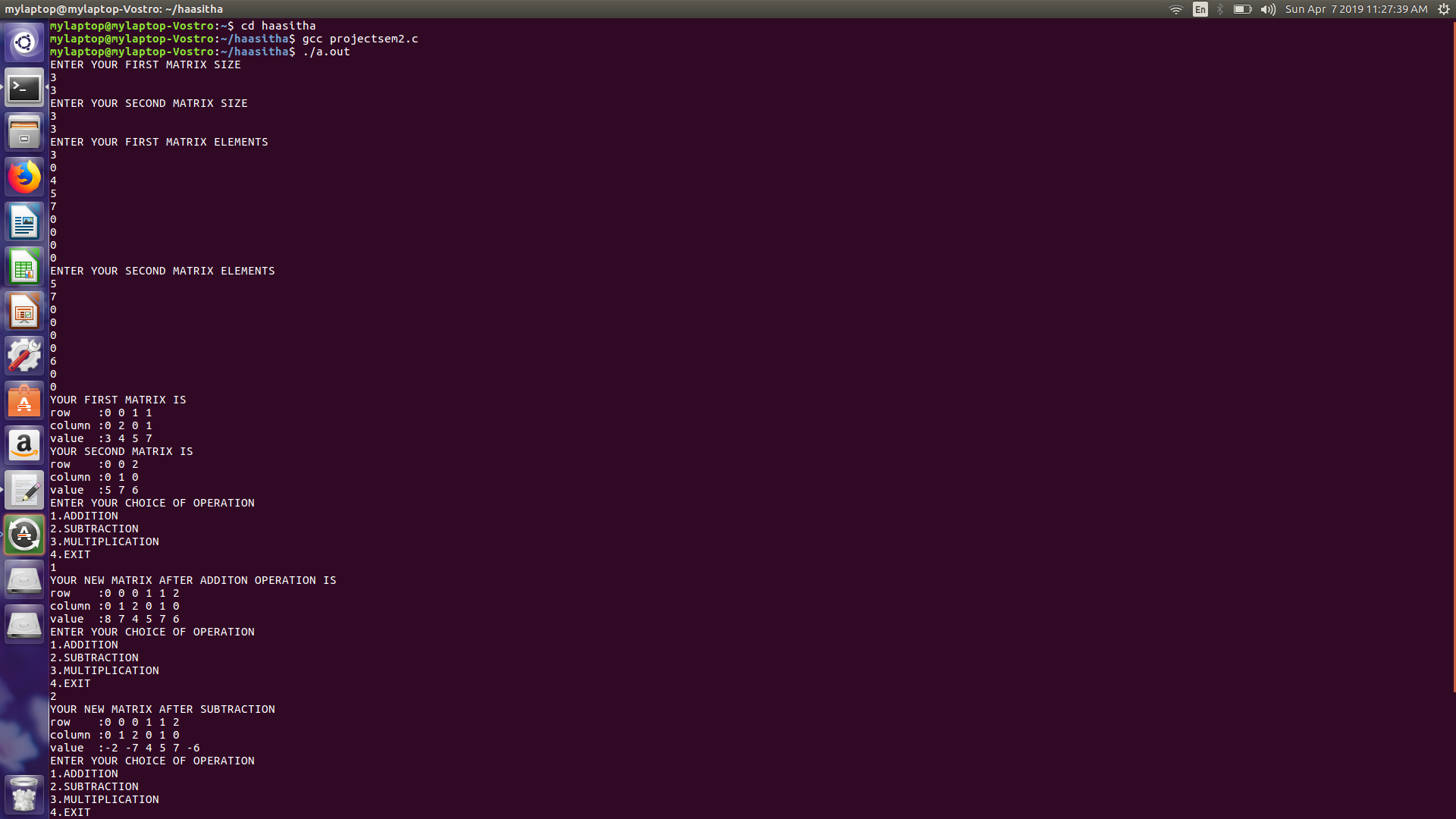
    printf("\n");

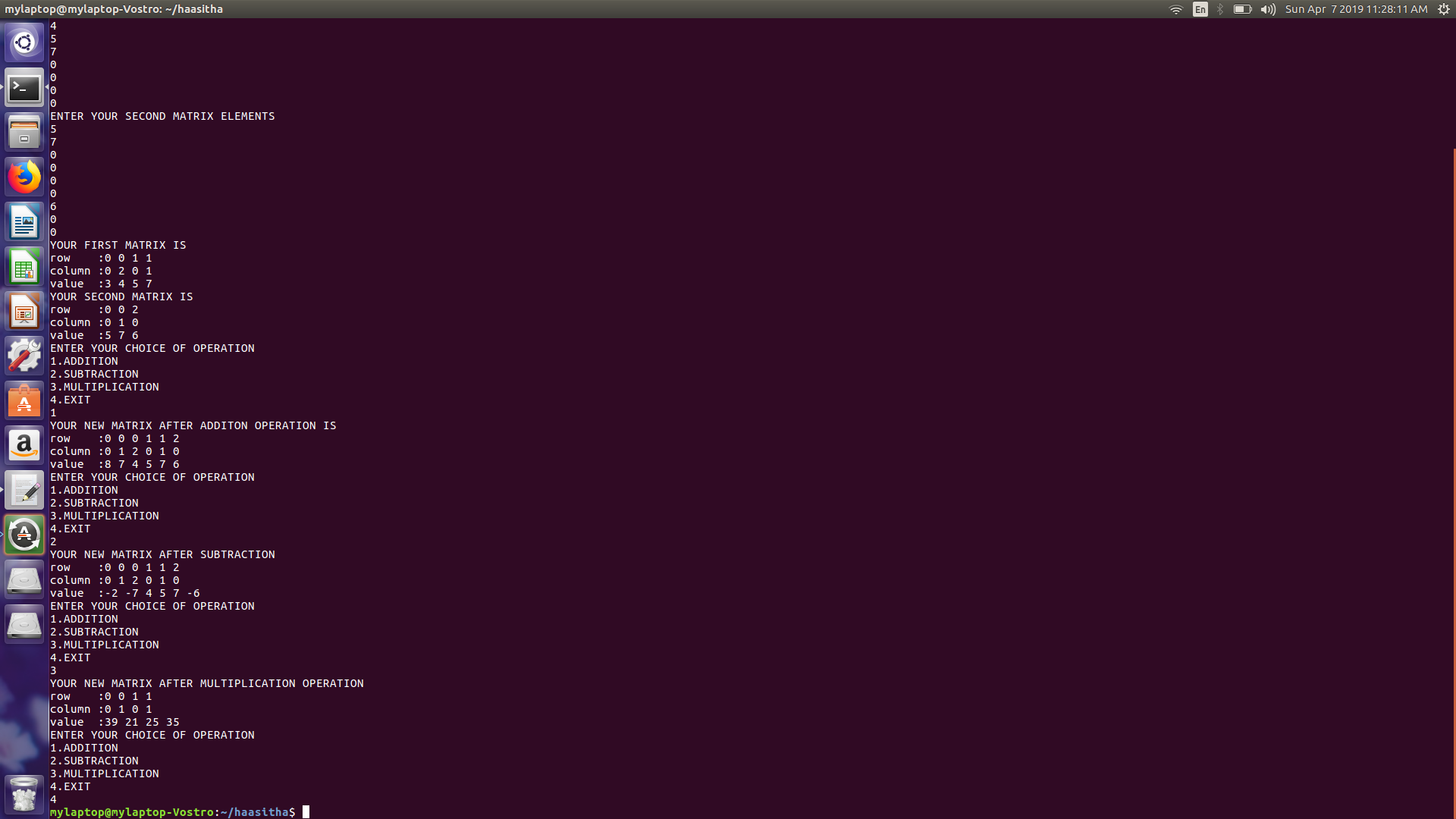
}

7.**INTEGRATION AND SYSTEM TESTING**

OUTPUTS

Screen Shots:



****

**8.CONCLUSION**

The only advantage of using a sparse matrix is that, if your matrix is mainly composed by zero elements, you could save space memorising just the non-zero elements. This lead to an implementation that is essentially a list of lists and will let you lose the O(1) time complexity of access of each elements.