**ASSIGNMENT – 13**

**1.PROBLEM STATEMENT**

Write a program in C to implement sparse matrix using one dimensional array.

**2.ALGORITHMS**

Algorithm **Sparse\_Matrix\_to\_1D\_Array**

**Input:**The pointer ‘arr’ to the 2-D array holding the matrix, the pointer ‘list’ to the array in which sparse matrix is to be stored and the dimensions ‘row’ and ‘col’ of the matrix.

**Output:** The elements of the sparse matrix fed into the array ‘list’

**Remarks**:

* If total number of non-zero elements in the input matrix is less than one third of the total elements, then the matrix is taken to be a sparse matrix.
* The list that stores the sparse matrix has three sections within each variable, namely row,col,ele for storing row value,column value and element value respectively.

**Steps:**

1. k=0 // points at beginning of list
2. **For**(i=1 to row) **do** //traversing along rows
3. **For**(j=1 to col) **do** //traversing along columns
4. **If**(arr[i][j]≠0) **then //**if an element is not zero
5. row🡨list[k]🡨i //store i in row section of list
6. col🡨ist[k]🡨j //store j in col section of list
7. ele🡨list[k]🡨arr[i][j]//store value in ele section
8. k=k+1//point k to next variable in list
9. **EndIf**
10. **EndFor**

**11. EndFor**

**12.Stop**

Algorithm **Count\_Nonzero\_Elements**

**Input:**The pointer to the matrix ‘arr’, and the dimensions of the matrix ‘row’ and ‘col’.

**Output:** The count of the total number of non-zero elements.

**Remarks**: It is considered that the matrix is not empty.

**Steps:**

1. cnt=0 //set cnt to zero
2. **For**(i=1 to row) **do** //traversing along rows
3. **For**(j=1 to col) **do** //traversing along columns
4. **If**(arr[i][j]≠0 **then** //if array element is not zero
5. cnt=cnt+1 // increment cnt by 1
6. **EndIf**
7. **EndFor**
8. **EndFor**
9. **Return** cnt // return the value of cnt
10. **Stop**

Algorithm **Is\_Sparse\_Matrix**

**Input:** The pointer to the matrix ‘arr’, and the dimensions of the matrix ‘row’ and ‘col’.

**Output:** If the entered matrix is a sparse matrix, a successful message is shown and the procedure returns **True** otherwise, it returns **False** with an unsuccessful message.

**Remarks:** If total number of non-zero elements in the input matrix is less than one third of the total elements, then the matrix is taken to be a sparse matrix.

**Steps:**

1. count=**Count\_Nonzero\_Elements**(arr,row,col)
2. limit=(row\*col)/3 // maximum number of nonzero elements
3. **If**(count<limit) **then** //if number of nonzero elements is in range
4. **Print** “Input matrix is sparse”
5. **Return**
6. **Else**
7. **Print** “Input matrix is not sparse”
8. **Exit** //terminate the program
9. **Stop**

**3.SOURCE CODE**

#include<stdio.h>

#include<stdlib.h>

//structure to hold information about sparse matrix elements

typedef struct spmat

{

int row;

int col;

int ele;

}spmat;

//function to take input in a matrix

void getmat(int(\*arr)[20],int row,int col)

{

int i,j;

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

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

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

}

//function to display a matrix

void dispmat(int(\*arr)[20],int row,int col)

{

int i,j;

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

{

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

printf("%d\t",arr[i][j]);

printf("\n");

}

}

//function to count the number of nonzero elements in a matrix

int nonzero(int(\*arr)[20],int row,int col)

{

int i,j,cnt=0;

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

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

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

cnt++;

return cnt;

}

//function to tell if a input matrix is sparse or not

void issparse(int(\*arr)[20],int row,int col)

{

int cnt,limit,total=row\*col;

cnt=nonzero(arr,row,col);

limit=total/3;

if(cnt<limit) //if number of nonzero elements is in allowed range

{

printf("Input matrix is a sparse matrix\n");

return;

}

else

{

printf("Input matrix is not a sparse matrix\n");

exit(0);

}

}

//function to store sparse matrix elements in 1-D array

void makesparsearr(int(\*arr)[20],spmat \*list,int row,int col)

{

int i,j,k=0;

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

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

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

{

list[k].row=i;

list[k].col=j;

list[k].ele=arr[i][j];

k++;

}

}

//function to recreate sparse matrix from the auxillary 1-D array

void dispsparse(spmat \*list,int cnt)

{

int i=0;

printf("ROW\tCOLUMN\tELEMENT\n");

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

{

printf("%d\t%d\t%d\n",list[i].row,list[i].col,list[i].ele);

}

}

//function to display an array

void disparr(spmat \*arr,int size)

{

int i;

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

{

printf("%d %d %d ",arr[i].row,arr[i].col,arr[i].ele);

}

}

int main(void)

{

int arr[20][20],row,col,cnt;

spmat \*list; //pointer to a array of structures

printf("Enter the number of rows: ");

scanf("%d",&row);

printf("Enter the number of columns: ");

scanf("%d",&col);

printf("Enter %d elements of the matrix: \n",row\*col);

getmat(arr,row,col);

printf("Entered matrix: \n");

dispmat(arr,row,col);

issparse(arr,row,col);

cnt=nonzero(arr,row,col);

list=(spmat\*)malloc(cnt\*sizeof(spmat));

makesparsearr(arr,list,row,col);

printf("\nElements of the 1-D array: ");

disparr(list,cnt);

printf("\nSparse matrix constructed from 1-D array: \n");

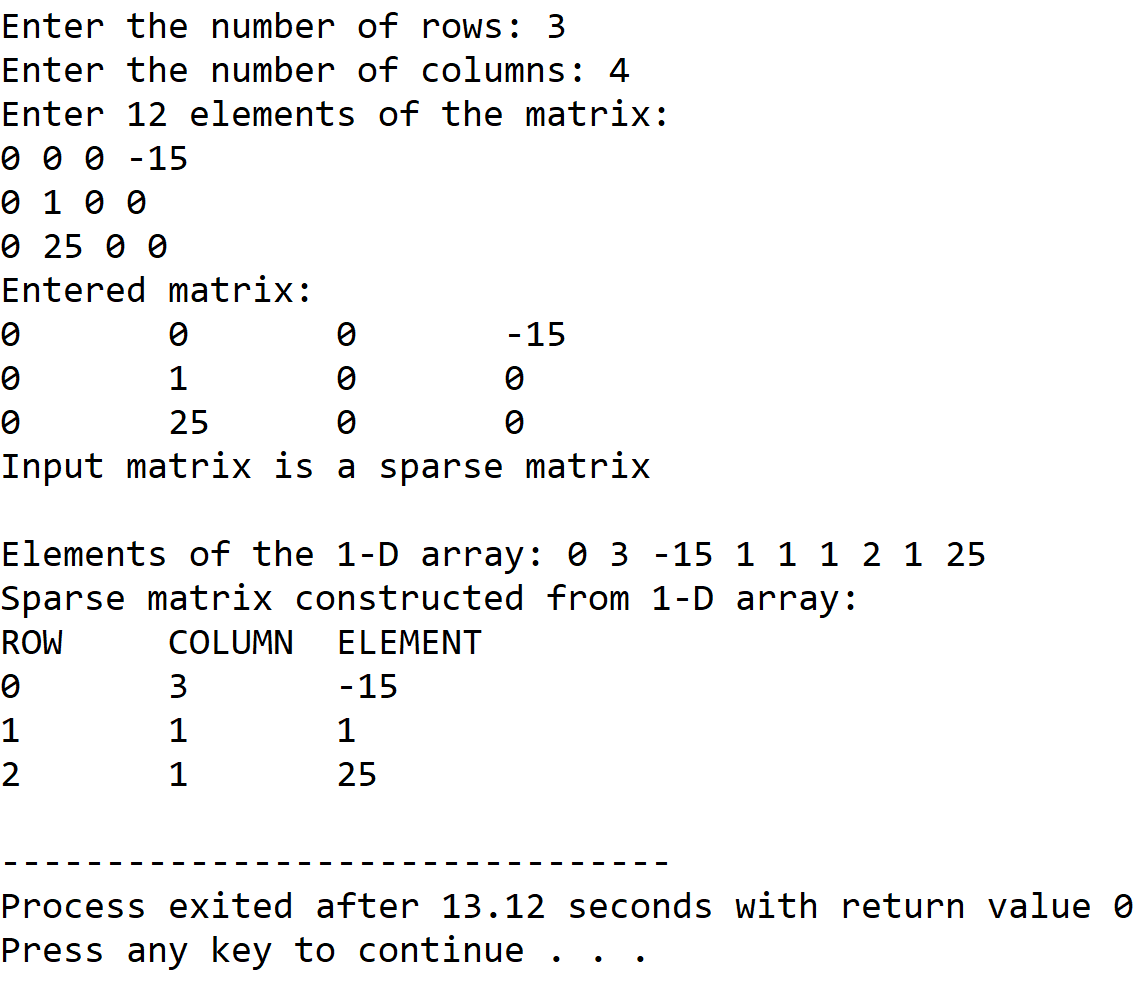
dispsparse(list,cnt);

return 0;

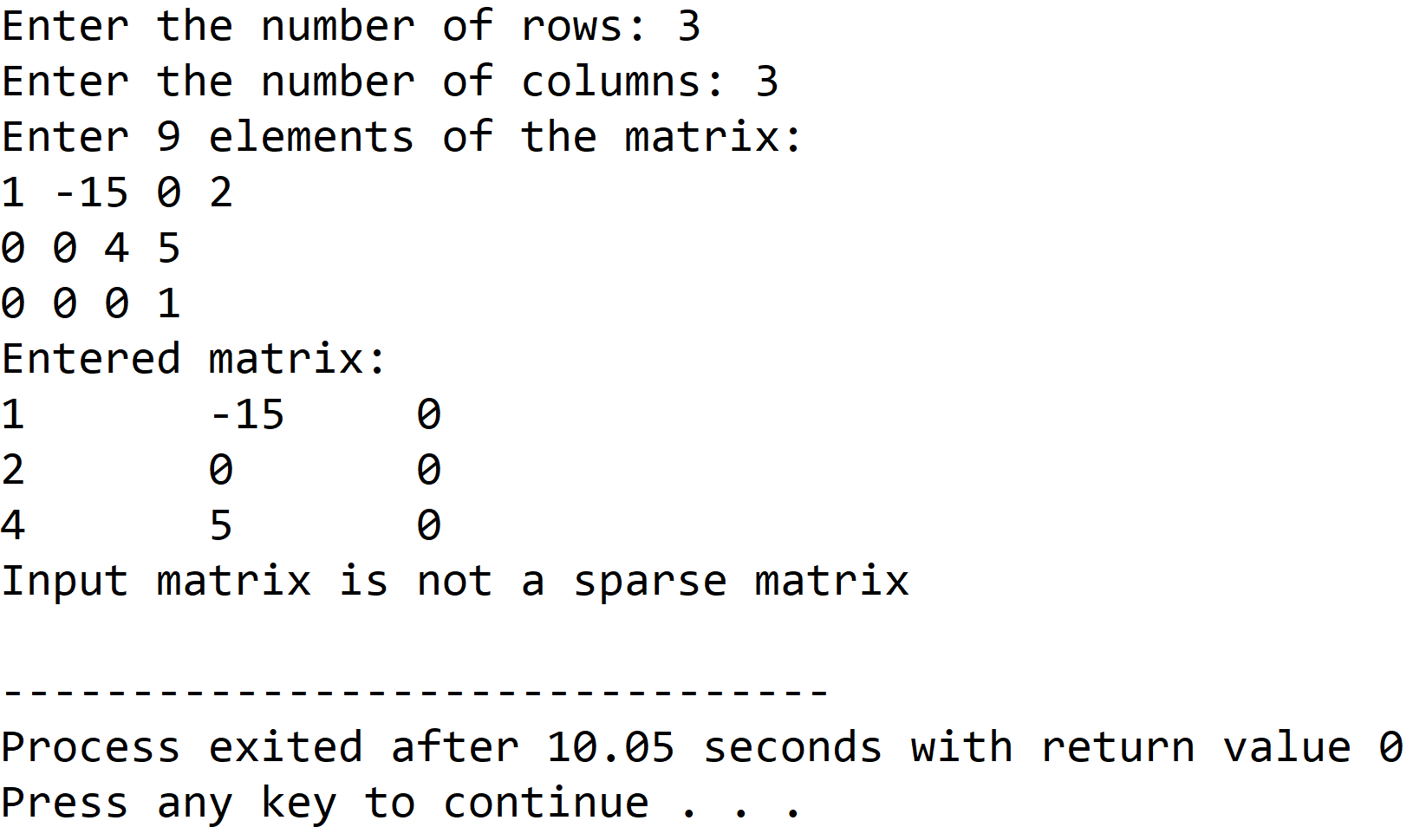
}

**4.OUTPUT**

**SET 1:**Input matrix is a sparse matrix.



**SET 1:**Input matrix is not a sparse matrix.



**5.DISCUSSIONS**

**Variable Description:**

* **arr:** Pointer to the 2-D arraty holding the matrix.
* **\*list:**Pointer to the array of structures to hold the sparse matrix information.
* **row,col:** Dimensions of the 2-D array.
* **cnt:** Count of the number of non-zero elements in matrix.
* **limit:**Maximum non-zero element count for a sparse matrix.
* **total:** Total number of elements in a matrix.
* **I,j,k:** loop counters.

**Limitations:**

* The two dimensional array used in the program is statically allocated, which means that there could be wastage of memory or lack of required memory.

**Uses:**

* The program shows how a particular data structure like a sparse matrix can be stored in the computer’s memory in a much more optimized manner if we store only that information which is unique and that cannot be reconstructed.

**Future Scope:**

* The two dimensional array used in the program can be dynamically allocated, leading to more effiecient use of memory.

**Teacher’s Signature**