

**EXPERIMENT REPORT OF DATA STRUCTURE**

Experiment 5

NAME : ABID ALI

STUDENT ID : 2019380141

DATE : 5/24/2021

E-Mail : abiduu354@gmail.com

**Problem Description:**

In this assignment, we are going to write a C program files(dse\_assign05.cpp”and text file with code also included), which includes taking two sparse matrices and performing basic operations such as add, multiply and transpose of the matrices. These program needs to use software tool for implementation.We used CodeBlocks (IDE). Using this program, result should consist of three sparse matrices, one will be obtained by adding the two input matrices, another will be multiplying the two matrices and furthermore obtained by transposing the first matrix. The sparse matrix will be stored in the sequential form.

**Goal:**

The program is going to addition ,multiplication and transpose operation on sparse matrix.

**Theory:**

**Sparse matrices** are those **matrices** that have the majority of their elements equal to zero. In other words, the **sparse matrix** can be defined as the **matrix** that has a greater number of zero elements than the non-zero elements.

In [numerical analysis](https://en.wikipedia.org/wiki/Numerical_analysis) and [scientific computing](https://en.wikipedia.org/wiki/Scientific_computing), a sparse matrix or sparse array is a [matrix](https://en.wikipedia.org/wiki/Matrix_(mathematics)) in which most of the elements are zero. There is no strict definition how many elements need to be zero for a matrix to be considered sparse but a common criterion is that the number of non-zero elements is roughly the number of rows or columns. By contrast, if most of the elements are nonzero, then the matrix is considered dense. The number of zero-valued elements divided by the total number of elements (e.g., m × n for an m × n matrix) is sometimes referred to as the sparsity of the matrix.

A sparse matrix can be represented by using TWO representations, those are as follows...

1. Triplet Representation (Array Representation)
2. Linked Representation

We are going to learn this following topics after learning this two Sparse matrix:

* We are going to understand Sparse matrix addition,multiplication and transpose operation .
* We will learn to store sequential form of storage.
* We have created classes and headed file and implementation files.
* We will understand the use of data-dependency of large files through this program which we will face in realworld.

**Implementation:**

We are going to define a program that allows to perform the addition, multiplication and transpose operations on sparse matrices. These functions which we are going to use are mainly given below:

**1. void AddTSMatrix(TSMatrix M, TSMatrix N, TSMatrix \*T)**

We are going to store the addition value of the sparse matrices M and N into T. To add the matrices, we are simply going to traverse through both matrices element by element and insert the smaller element (one which has smaller row and column value) into our desired resultant matrix. If we find an element with the same row and column value, we are going to add their values and insert the added data (non-zero number) into the resultant matrix.Then,eventually we will get our desired resultant matrix.

void AddTSMatrix(sparse\_matrix b)

{

// If the matrices doesn't have same dimensions

if (row != b.row || column != b.column)

{

cout << "Matrix addition cannot be done.";

}

else

{

int mat1\_position = 0, mat2\_position = 0;

sparse\_matrix result(row, column);

while (mat1\_position < l && mat2\_position < b.l)

{

// If 2nd matrix row and column is smaller

if(Elements[mat1\_position][0] > b.Elements[mat2\_position][0] || (Elements[mat1\_position][0] == b.Elements[mat2\_position][0] &&

Elements[mat1\_position][1] > b.Elements[mat2\_position][1]))

{

// Insert smaller value into result

result.insert(b.Elements[mat2\_position][0],

b.Elements[mat2\_position][1],

b.Elements[mat2\_position][2]);

mat2\_position++;

}

else if (Elements[mat1\_position][0] < b.Elements[mat2\_position][0] || // if a's row and column is smaller

(Elements[mat1\_position][0] == b.Elements[mat2\_position][0] &&

Elements[mat1\_position][1] < b.Elements[mat2\_position][1]))

{

// Insert smaller values into result

result.insert(Elements[mat1\_position][0],

Elements[mat1\_position][1],

Elements[mat1\_position][2]);

mat1\_position++;

}

else

{

// AddSTMatrix the values as row and column is same

int AddTSMatrixedvalue = Elements[mat1\_position][2] +

b.Elements[mat2\_position][2];

if (AddTSMatrixedvalue != 0)

result.insert(Elements[mat1\_position][0],

Elements[mat1\_position][1],

AddTSMatrixedvalue);

// then insert

mat1\_position++;

mat2\_position++;

}

}

**2. void MultiplyTSMatrix(TSMatrix M, TSMatrix N, TSMatrix \*Q)**

Then,we have to store the multiplication of the sparse matrices M and N into Q. This is how we will Multiply the matrices, the process is that one row of M one at a time. Any row that has value equal to x in the first matrix and column value equal to y in the second matrix will go towards result[x][y]. This will help us to obtain by multiplying all such elements having corresponding column value in M and row value in N and adding only those with the row as x in first matrix and column as y in the second matrix to get the result[x][y]. This process is performed by maintaining an array rpos[] whose i-th value indicates the number of elements in the matrix less then row i, and a temporary array ctemp[] to store the adding results for the entire row of Q.

void MultiplySMatrix(sparse\_matrix b)

{

if (column != b.row)

{

// Invalid multiplication

cout << "Error...Invalid dimensions";

return;

}

// Transpose SMatrix b to compare row and column value and to AddSMatrix them at the end

b = b.TransposeTSMatrix();

int matrix1\_position, matrix2\_position;

// result matrix of dimension row X b.column

// However b has been TransposeSMatrix,

// Hence row X b.row

sparse\_matrix result(row, b.row);

// iterate over all Elements of A

for (matrix1\_position = 0; matrix1\_position < l;)

{

// Current row of result matrix

int r = Elements[matrix1\_position][0];

// Iterate over all Elements of B

for (matrix2\_position = 0; matrix2\_position < b.l;)

{

// current column of result matrix

// Elements[,0] used as b is Transpose SMatrix

int c = b.Elements[matrix2\_position][0];

// temporary pointers created to AddSMatrix all

// Multiplied value to obtain current value

// Elements of result matrix

int tempa = matrix1\_position;

int tempb = matrix2\_position;

int sum = 0;

// Iterating over all Elements with

// Same row and column value then we are going calculate result[r]

while (tempa < l && Elements[tempa][0] == r &&

tempb < b.l && b.Elements[tempb][0] == c)

{

if (Elements[tempa][1] < b.Elements[tempb][1])

// Skipped a

tempa++;

else if (Elements[tempa][1] > b.Elements[tempb][1])

// Skipped b

tempb++;

else

// Same column, therfore Multiply TSMatrix and increment

sum += Elements[tempa++][2] \*

b.Elements[tempb++][2];

}

if (sum != 0)

result.insert(r, c, sum);

while (matrix2\_position < b.l &&

b.Elements[matrix2\_position][0] == c)

// Jump to next column

matrix2\_position++;

}

while (matrix1\_position < l && Elements[matrix1\_position][0] == r)

// Jump to next row

matrix1\_position++;

}

result.print();

}

**3. void TransposeTSMatrix(TSMatrix M, TSMatrix \*T)**

void TransposeSMatrix(SMatrix M, SMatrix \*T): Store the transpose of the sparse matrix M

into T. To Transpose a matrix, we can simply change every column value to the row value and vice-versa, however, in this case, the resultant matrix won’t be sorted as we require. Hence, we initially determine the number of elements less than the current element’s column being inserted in order to get the exact index of the resultant matrix where the current element should be placed. This is done by maintaining an array cpos[] whose j-th value indicates the number of elements in the matrix less than the column j. Refer to the lecture slides for algorithm details.

Furthermore,storing the transpose of the sparse matrix M into T. Then,we are going to Transpose a matrix, we can simply change every column value to the row value and vice-versa. However, in this case, the resultant matrix won’t be sorted as it will be required.

sparse\_matrix TransposeTSMatrix()

{

// new matrix with inversed row X column

sparse\_matrix result(column, row);

// same number of Elements

result.l = l;

// to count number of Elements in each column

int \*count = new int[column + 1];

// initialize all to 0

for (int i = 1; i <= column; i++)

count[i] = 0;

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

count[Elements[i][1]]++;

int \*index = new int[column + 1];

// to count number of Elements having

// column smaller than particular i

// as there is no column with value < 0

index[0] = 0;

// initialize rest of the indice

for (int i = 1; i <= column; i++)

index[i] = index[i - 1] + count[i - 1];

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

{

// insert a Elements at rpos and

// increment its value

int rpos = index[Elements[i][1]]++;

// TransposeSMatrix row=column

result.Elements[rpos][0] = Elements[i][1];

// TransposeSMatrix column=row

result.Elements[rpos][1] = Elements[i][0];

// same value

result.Elements[rpos][2] = Elements[i][2];

}

// the above method ensures

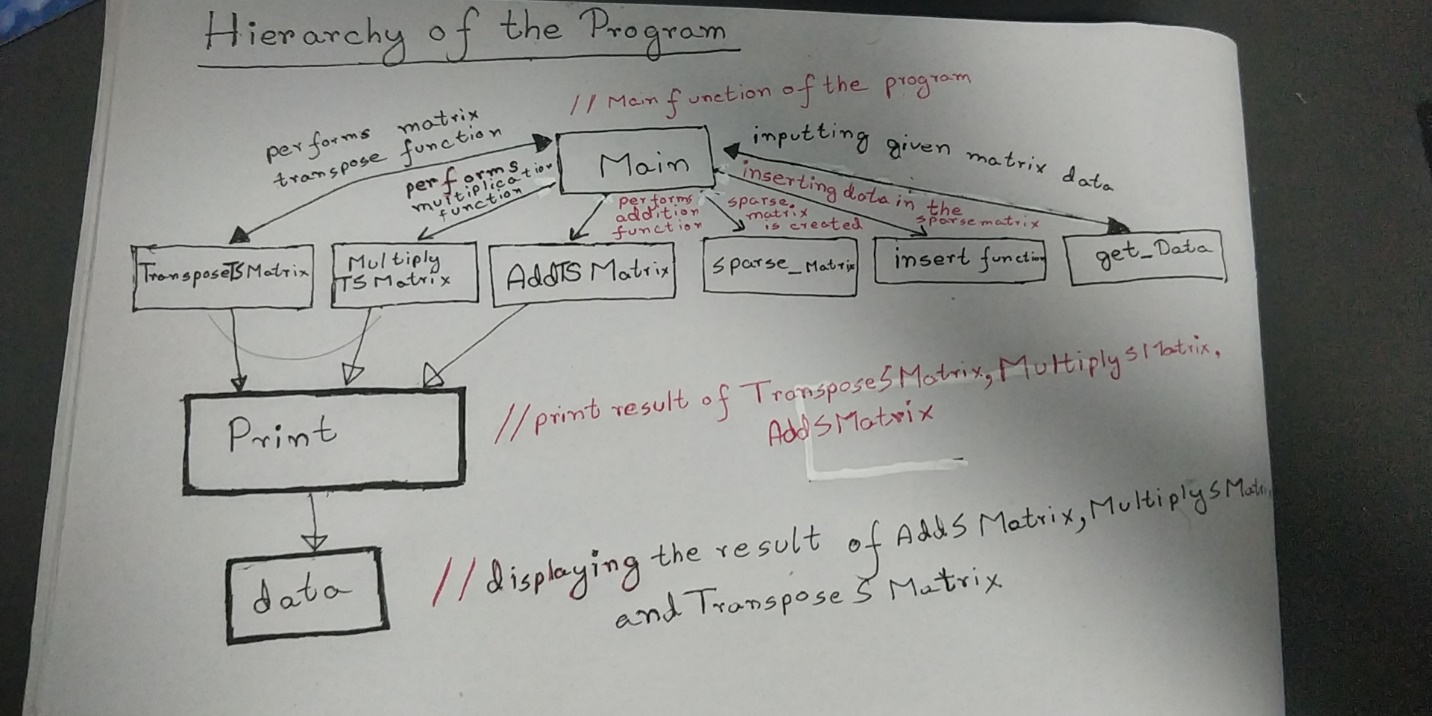
// sorting of Transpose SMatrix matrix

// according to row-column value

return result;

}

**Hierarchy of the program:**

****

**Code:**

/\*

\*Operation on the Sparse Matrix

\*dse\_assign05

\*Student Name :ABID ALI

\*Student ID :2019380141

\*E-Mail :abiduu354@gmail.com

\*/

#include<stdlib.h>

#include <iostream>

#include<bits/stdc++.h>

using namespace std;

class sparse\_matrix

{

// Maximum number of Elementss in this matrix

const static int MAX = 20000;

// Initialized double-pointer to store

// the triple-represented form

int \*\*Elements;

// This will be our Matrix dimension

int row, column;

// Total number of Elements in the matrix

//We are making a length of int type

int l;

public:

sparse\_matrix(int r, int c)

{

// initializing the row inputs

row = r;

// initializing the column inputs

column = c;

// initialize l to 0,So that our length start from first index.

l = 0;

//Array of Pointer to make a matrix

Elements = new int \*[MAX];

// Representation of the Array of Sparse matrix

for (int i = 0; i < MAX; i++)

Elements[i] = new int[3];

}

// Insert the elements into sparse matrix

void insert(int r, int c, int value)

{

// When there is a invalid entry

if (r > row || c > column)

{

cout << "An invalid entry";

}

else

{

// inserting the row value

Elements[l][0] = r;

// inserting the column value

Elements[l][1] = c;

// insert the following Element's value

Elements[l][2] = value;

// Increment number of Elements in matrix,So,that we can move to next elements.

l++;

}

}

void AddTSMatrix(sparse\_matrix b)

{

// If the matrices doesn't have same dimensions

if (row != b.row || column != b.column)

{

cout << "Matrix addition cannot be done.";

}

else

{

int mat1\_position = 0, mat2\_position = 0;

sparse\_matrix result(row, column);

while (mat1\_position < l && mat2\_position < b.l)

{

// If 2nd matrix row and column is smaller

if(Elements[mat1\_position][0] > b.Elements[mat2\_position][0] || (Elements[mat1\_position][0] == b.Elements[mat2\_position][0] &&

Elements[mat1\_position][1] > b.Elements[mat2\_position][1]))

{

// Insert smaller value into result

result.insert(b.Elements[mat2\_position][0],

b.Elements[mat2\_position][1],

b.Elements[mat2\_position][2]);

mat2\_position++;

}

else if (Elements[mat1\_position][0] < b.Elements[mat2\_position][0] || // if a's row and column is smaller

(Elements[mat1\_position][0] == b.Elements[mat2\_position][0] &&

Elements[mat1\_position][1] < b.Elements[mat2\_position][1]))

{

// Insert smaller values into result

result.insert(Elements[mat1\_position][0],

Elements[mat1\_position][1],

Elements[mat1\_position][2]);

mat1\_position++;

}

else

{

// AddSTMatrix the values as row and column is same

int AddTSMatrixedvalue = Elements[mat1\_position][2] +

b.Elements[mat2\_position][2];

if (AddTSMatrixedvalue != 0)

result.insert(Elements[mat1\_position][0],

Elements[mat1\_position][1],

AddTSMatrixedvalue);

// then insert

mat1\_position++;

mat2\_position++;

}

}

// insert remaining Elements

while (mat1\_position < l)

result.insert(Elements[mat1\_position][0],

Elements[mat1\_position][1],

Elements[mat1\_position++][2]);

while (mat2\_position < b.l)

result.insert(b.Elements[mat2\_position][0],

b.Elements[mat2\_position][1],

b.Elements[mat2\_position++][2]);

// print result

result.print();

}

}

sparse\_matrix TransposeTSMatrix()

{

// new matrix with inversed row X column

sparse\_matrix result(column, row);

// same number of Elements

result.l = l;

// to count number of Elements in each column

int \*count = new int[column + 1];

// initialize all to 0

for (int i = 1; i <= column; i++)

count[i] = 0;

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

count[Elements[i][1]]++;

int \*index = new int[column + 1];

// to count number of Elements having

// column smaller than particular i

// as there is no column with value < 0

index[0] = 0;

// initialize rest of the indice

for (int i = 1; i <= column; i++)

index[i] = index[i - 1] + count[i - 1];

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

{

// insert a Elements at rpos and

// increment its value

int rpos = index[Elements[i][1]]++;

// TransposeSMatrix row=column

result.Elements[rpos][0] = Elements[i][1];

// TransposeSMatrix column=row

result.Elements[rpos][1] = Elements[i][0];

// same value

result.Elements[rpos][2] = Elements[i][2];

}

// the above method ensures

// sorting of Transpose SMatrix matrix

// according to row-column value

return result;

}

void MultiplySMatrix(sparse\_matrix b)

{

if (column != b.row)

{

// Invalid multiplication

cout << "Error...Invalid dimensions";

return;

}

// Transpose SMatrix b to compare row and column value and to AddSMatrix them at the end

b = b.TransposeTSMatrix();

int matrix1\_position, matrix2\_position;

// result matrix of dimension row X b.column

// However b has been TransposeSMatrix,

// Hence row X b.row

sparse\_matrix result(row, b.row);

// iterate over all Elements of A

for (matrix1\_position = 0; matrix1\_position < l;)

{

// Current row of result matrix

int r = Elements[matrix1\_position][0];

// Iterate over all Elements of B

for (matrix2\_position = 0; matrix2\_position < b.l;)

{

// current column of result matrix

// Elements[,0] used as b is Transpose SMatrix

int c = b.Elements[matrix2\_position][0];

// temporary pointers created to AddSMatrix all

// Multiplied value to obtain current value

// Elements of result matrix

int tempa = matrix1\_position;

int tempb = matrix2\_position;

int sum = 0;

// Iterating over all Elements with

// Same row and column value then we are going calculate result[r]

while (tempa < l && Elements[tempa][0] == r &&

tempb < b.l && b.Elements[tempb][0] == c)

{

if (Elements[tempa][1] < b.Elements[tempb][1])

// Skipped a

tempa++;

else if (Elements[tempa][1] > b.Elements[tempb][1])

// Skipped b

tempb++;

else

// Same column, therfore Multiply TSMatrix and increment

sum += Elements[tempa++][2] \*

b.Elements[tempb++][2];

}

if (sum != 0)

result.insert(r, c, sum);

while (matrix2\_position < b.l &&

b.Elements[matrix2\_position][0] == c)

// Jump to next column

matrix2\_position++;

}

while (matrix1\_position < l && Elements[matrix1\_position][0] == r)

// Jump to next row

matrix1\_position++;

}

result.print();

}

// Printing the matrix

void print()

{

cout <<"(" << row << "x" << column<<")"<<endl;

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

{

cout << Elements[i][0] << " " << Elements[i][1]

<< " " << Elements[i][2] << endl;

}

}

};

int main()

{

int p,q,n1,n2,x,y,z,i,j,k;

cin>>p>>n1;

// Creating 1st sparse matrices and insert value

sparse\_matrix a(p, n1);

while(1)

{

cin>>x>>y>>z;

if(x==0 && y==0 && z==0) break;

a.insert(x, y, z);

}

cin>>q>>n2;

// Creating 2nd sparse matrices and insert value

sparse\_matrix b(q, n2);

while(1)

{

cin>>i>>j>>k;

if(i==0 && j==0 && k==0) break;

b.insert(i, j, k);

}

cout << "Result of the Addition:";

a.AddTSMatrix(b);

cout << "Result of the Multiplication:";

a.MultiplySMatrix(b);

cout << "Result of Transpose on the first matrix:";

sparse\_matrix aTransposeTSMatrix = a.TransposeTSMatrix();

aTransposeTSMatrix.print();

}

**Test Description and Results:**

**Test 1**

Sample input:

4 4

1 2 10

1 4 12

3 3 5

4 1 15

4 2 20

0 0 0

4 4

1 3 8

2 4 23

3 3 9

4 1 20

4 2 -20

0 0 0

Sample output:

Result of Addition:(4×4)

1 2 10

1 3 8

1 4 12

2 4 23

3 3 14

4 1 35

Result of Multiplication:(4×4)

1 1 240

1 2 -240

1 4 230

3 3 45

4 3 120

4 4 460

Result of Transpose on the first matrix:(4×4)

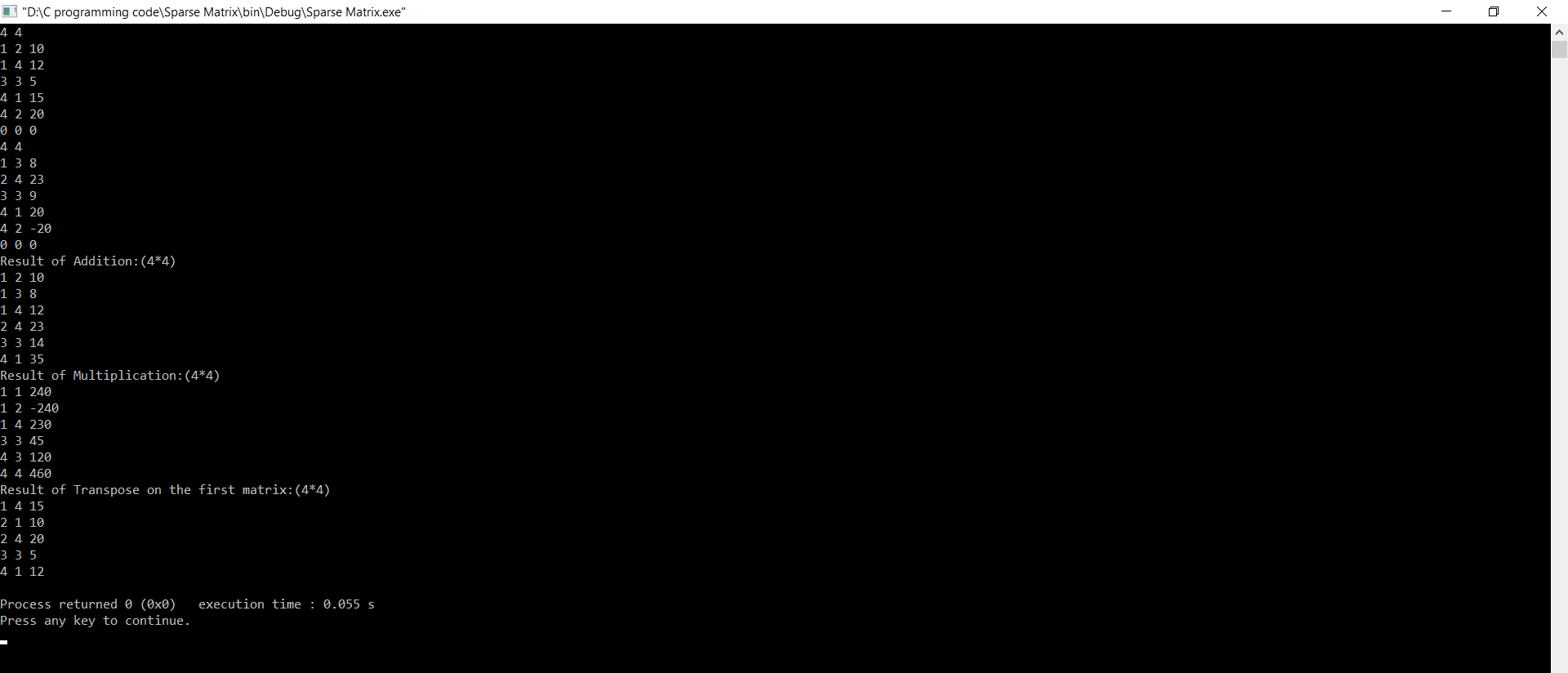
1 4 15

2 1 10

2 4 20

3 3 5

4 1 12



**Test 2**

Input:

10 10

1 3 13

2 1 6

2 8 15

3 1 19

3 6 41

4 9 20

5 2 7

6 10 3

7 6 5

8 9 20

9 2 12

0 0 0

10 10

1 3 8

2 2 6

3 5 19

3 6 4

5 6 7

6 1 14

6 5 2

7 6 45

8 9 1

9 2 22

0 0 0

Output:

Result of Addition:(10×10)

1 3 21

2 1 6

2 2 6

2 8 15

3 1 19

3 5 19

3 6 45

4 9 20

5 2 7

5 6 7

6 1 14

6 5 2

6 10 3

7 6 50

8 9 21

9 2 34

Result of Multiplication:(10×10)

1 5 247

1 6 52

2 3 48

2 9 15

3 1 574

3 3 152

3 5 82

4 2 440

5 2 42

7 1 70

7 5 10

8 2 440

9 2 72

Result of Transpose on the first matrix:(10×10)

1 2 6

1 3 19

2 5 7

2 9 12

3 1 13

6 3 41

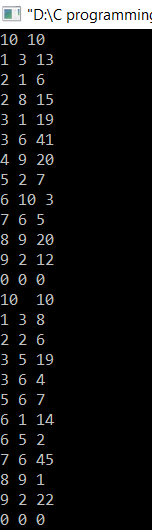
6 7 5

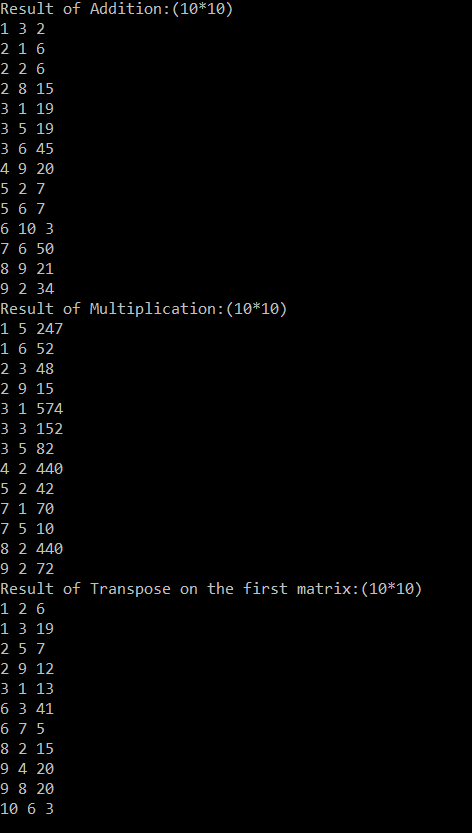
8 2 15

9 4 20

9 8 20

10 6 3





Bug:I used different inputs then I noticed for lesser amount of rows and columns the program works perfectly.When the number of rows and columns increases the program starts to work lag and my OS system can handle and finally crashed but when I used online compiler then my program worked correctly.

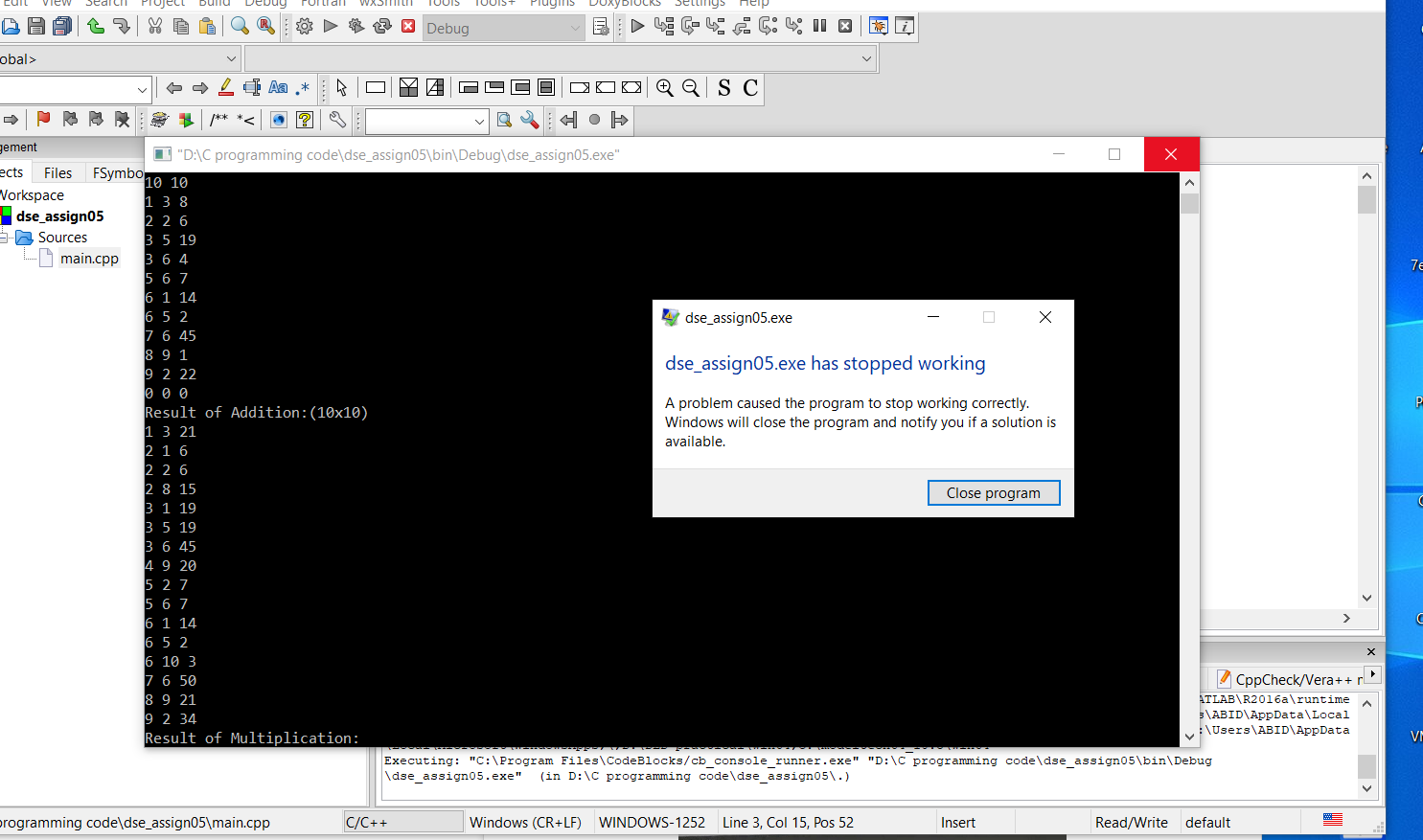


Fig:CodeBlocks implementation of (10\*10) Matrix crashing

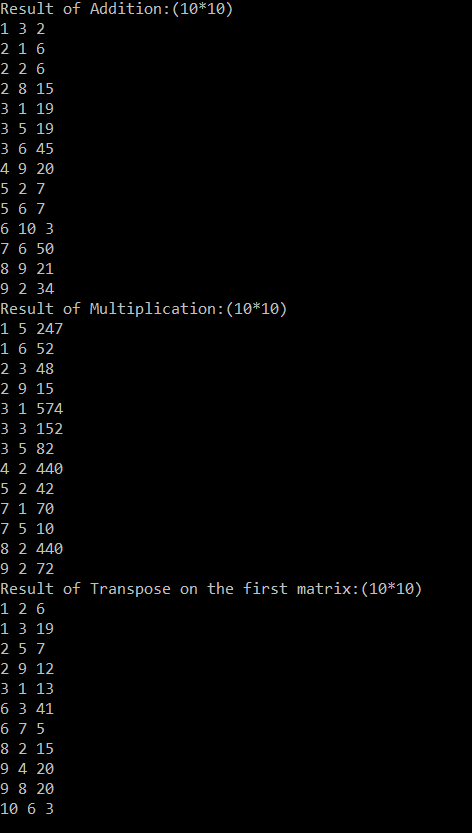


Fig: Using online compiler for implementation of (10\*10) Matrix

**Epilogue:**

While doing this assignment I had three big bugs which had influence on program’s working and many troubles with compilation programs, because of syntax and different kinds of functions and using all those functions together. At first,I draw the picture and watch some video about the topic and then I started to Implement the design in code. The hardest part was understand the concept of **Operations on Sparse Matrices.**Such as:**MultiplySMatrix ,**I faced a problem that for (10\*10) that it takes long time to iterate over all the elements provided to us.I searched in the internet and read different articles.I came to the conclusion,if I want to do this function using this program we cannot do it because the memory cannot store large value and it crashes.When,I used online compiler to do this program ,I got our desired result.

Second problem, “TransposeSMatrix”, we were having an opportunity to become more acquainted with classes and their constructors and chance to write simple operational method.

In this semester,I am learning Object Oriented Programming,I used the the concept that I learned in that class .Such as:How to classes and object of those classes,how to implement those concept in code.In C++,OOP system is available.So,I used it to implement this code.

I used online resource for example:

https://www.geeksforgeeks.org/operations-sparse-matrices/

Watch some video about the topic and then I started to Implement the design in code. Then,I go through my code code again to remove any unused code or function.At last,I was able to solve it correctly.

The most interesting thing in that assignment for me was to implement all those new idea together and create a program. I got more acquainted with **Operations on Sparse Matrices**.

**Attachments:**

1)dse\_assign05.cpp

2)dse\_assign05.txt

3)dse\_assign05.pdf

**Acknowledgements:**

I complete this assignment by myself by using online videos and different books discussing about Algorithm and Data structure. Also,used my knowledge about Object Oriented Programming to solve this problem.It was very useful and helpful for me to increase knowledge for solving complex problem.

**Remarks and Grade (by the instructor)**

Instructor Signature:

Grading Date: