CS 2413 – Data Structures – Fall 2024 – Project One
Due 11:59 PM, September 6, 2024

Description: A rectangular array of numbers is called a *matrix*. Sometimes, a matrix is referred to as a table, 2-dimensional array, or array of arrays. Consider the following matrix below. It has 5 rows (denoted *n*) and 8 columns (denoted *m*). As you can see the row and column numbers start with a 0. Many entries might be zero in matrix computations, especially with large matrices. Storing all elements, including zeros, would be inefficient regarding memory and computational resources. The Sparse Matrix Representation (SMR) efficiently stores and operates on matrices where most the elements are zero. Instead of storing every element, we only store the non-zero elements along with their positions in the matrix, which reduces the amount of memory used and can make certain operations more efficient. This project will help you learn how to implement and work with this data structure using object-oriented principles by encapsulating it within a class and relevant operations.

This project focuses on manipulating and storing matrices using a specific data structure called Sparse Matrix Representation (SMR). We will encapsulate the SMR within a class, including a set of methods designed to operate on matrices stored in this format.

	0	1	2	3	4	5	6	7
0	100			900		500		
1					200			300
2		400					800	
3			200					
4	1600				700			

The empty cells have the same *common value* (for example, it can be 0). The above matrix is said to be *sparse* because the total number of common values is significantly large in comparison to other values in the matrix. In the example above, we have a total of 40 values in the matrix (8 × 5), 10 non-common values, and 30 common values.

A sparse matrix can be represented using sparse matrix representation. In a table format, the sparse matrix representation for the above matrix looks like the following

	Row#	Column#	Value
0	0	0	100
1	0	3	900
2	0	5	500
3	1	4	200
4	1	7	300
5	2	1	400
6	2	6	800
7	3	2	200
8	4	0	1600
9	4	4	700

In this project, you will create appropriate C++ classes (see below) to create the sparse matrix representation data structure along with matrix operations on the sparse matrix.

Your Project Implementation: As part of this project, you will create two classes as described below. The first class that you will create will be a SparseRow class. The second class that you will create will be the SparseMatrix class. A partial (you are responsible for completing this and writing all the methods) class definition for both classes is given below.

You will create two classes for this project, as described below. The first class that you will create will be a SparseRow class. The second class that you will create will be the SparseMatrix class. A partial (you are responsible for completing this along with writing all the methods) class definition for both classes is given below.

Programming Objectives:

1. All code must be in C++.You will create a class given below with appropriate fields.
2. The class will have several methods, including matrix addition, transpose, and matrix-matrix multiplication.
3. All input will be read via redirected input. That is, you will not open a file inside the program.
4. The class structure is shown below (you are responsible for fixing any syntax errors).
5. The structure for your main program is also provided. You will use that for your project.

```
class SparseRow {
      protected:
            int row; //Row#
            int col; //Column#
            int value; //We will assume that all our values will be integers
      public:
            SparseRow (); //default constructor; row=-1;col=-1;value=0
            display(); // print Row#, Column#, value
            ostream& operator<< (ostream& s, const SparseRow);
            //other methods that are necessary such as get and set
};

class SparseMatrix {
      protected:
            int noRows; //Number of rows of the original matrix
            int noCols; //Number of columns of the original matrix
            int commonValue; //read from input
            int noNonSparseValues;
            SparseRow* myMatrix; //Array
      public:
            SparseMatrix ();
            SparseMatrix (int n, int m, int cv);
            SparseMatrix* Transpose (); //Matrix Transpose
            SparseMatrix* Multiply (SparseMatrix& M);
            SparseMatrix* Add (SparseMatrix& M);
            ostream& operator<< (ostream& s, const SparseMatrix& sm);
            displayMatrix (); //Display the matrix in its original format
            //other methods that are necessary such as get and set
};
```

Your main program will have the following structure (changes may be required).

```
#include <iostream>
using namespace std;

//define all your classes here (as given above)
//write the methods after the class definition

int main () {

      int n, m, cv, noNSV;
      SparseMatrix* temp;

      cin >> n >> m >> cv >> noNSV;
      SparseMatrix* firstOne = new SparseMatrix(n, m, cv, noNSV);

      //Write the Statements to read in the first matrix

      cin >> n >> m >> cv >> noNSV;
      SparseMatrix* secondOne = new SparseMatrix(n, m, cv, noNSV);

      //Write the Statements to read in the second matrix

      cout << "First one in matrix format" << endl;
      (*firstOne).displayMatrix();

      cout << "First one in sparse matrix format" << endl;
      cout << (*firstOne);

      cout << "Second one in matrix format" << endl;
      (*secondOne).displayMatrix();

      cout << "Second one in sparse matrix format" << endl;
      cout << (*secondOne);

      cout << "Transpose of the first one in matrix" << endl;
      cout << (*(*firstOne).Transpose());


      cout << "Matrix Addition Result" << endl;

      temp = (*(*firstOne).Addition(secondOne));
      cout << temp;
      (*temp).displayMatrix();

      cout << "Matrix Multiplication Result" << endl;

      temp = (*(*firstOne).Multiply(secondOne));
      cout << temp;
      (*temp).displayMatrix();

}
```

You must ensure that your program outputs the correct results. Make sure that you know matrix transpose, addition, and multiplication. Write those methods. Your input for each matrix will have the following structure. Note that in the first line, we have 5 rows, 8 columns, 0 being the common value and 10 being the number of non-sparse values.

5 8 0 10
100 0 0 900 0 500 0 0
0 0 0 0 200 0 0 300
0 400 0 0 0 0 800 0
0 0 200 0 0 0 0 0
1600 0 0 0 700 0 0 0

After the first input matrix you will read the second input matrix which will have the similar format.

You must ensure that your program outputs the correct results.

Redirected Input: Redirected input provides you a way to send a file to the standard input of a program without typing it using the keyboard. To use redirected input in Visual Studio environment, follow these steps: After you have opened or created a new project, on the menu go to project, project properties, expand configuration properties until you see Debugging, on the right you will see a set of options, and in the command arguments type "< **input filename**". The < sign is for redirected input and the **input filename** is the name of the input file (including the path if not in the working directory). A simple program that reads a matrix can be found below.

```
#include <iostream>

using namespace std;

int main () {

        int r,c,nsv;
        cin >> r >> c >> cv;
          cout << r << c << cv << endl;
        for (int i=0; i < nsv; i++) {
           cin >> rn >> cn >> val;
           cout << rn << cn << val << endl;
        }
        return 0;
}
```

Constraints

1. In this project, the only header you will use is #include <iostream> and using namespace std.
2. None of the projects is a group project. Consulting with other members of this class our seeking coding solutions from other sources including the web on programming projects is strictly not allowed and plagiarism charges will be imposed on students who do not follow this.

Project Submission Requirements:

1. **Code Development (75%):** Implement the provided class structure for the Sparse Matrix by writing the necessary methods to manipulate sparse matrices. Your implementation must be fully compatible with the main program provided, which is designed to create and manipulate Sparse Matrice objects. Specifically, your code should include methods for adding, multiplying, and transposing matrices and any other operations specified in the project guidelines. Your code must run successfully with the main program and the corresponding input, demonstrating the correct functionality of the Sparse Matrix class and its methods.

    - **LLM/AI Tool Usage:** You can use Large Language Models (LLMs) or AI tools, such as GitHub Copilot, to assist in writing and refining your classes and their methods. If you did not use LLM or AI tools to write your project, you still have to show for 2. below how you would have used it to find a solution to the project. **You need to make sure that you use the class structure provided to you as the basis, and failure to do that will result in zero points for this project.**

2. **LLM and GitHub Copilot Usage Documentation (15%):** If you choose to use LLM tools or GitHub Copilot, you must document your usage. This documentation (in PDF Format) should include:

    - **Prompts and Suggestions:** Provide the specific prompts or suggestions you used, such as "Generate a method for adding two sparse matrices in C++" or "How can I implement a transpose function for a sparse matrix?"
    - **Rationale:** Explain why you chose these prompts or suggestions and how they contributed to the development of your classes. For instance, you might describe how a particular suggestion helped you structure the addition method or handle edge cases in matrix multiplication.
    - **Incremental Development:** Detail how you used the tools to build and refine your classes and methods incrementally. For example, you might start by generating a basic structure for the classes and then refine individual methods, ensuring each one integrates smoothly with the main program.

3. **Debugging and Testing Plan (10%):** Submit a comprehensive debugging and testing plan. This should include:

    - **Specific Tests:** Describe the tests you conducted on your class methods, such as checking that matrix addition works correctly when combining matrices of different sparsities or ensuring that the transpose operation correctly swaps rows and columns.
    - **Issues and Resolutions:** Document any issues you encountered, such as handling zero values or optimizing for performance, and how you resolved them.
    - **Verification:** Explain how you verified that your classes work correctly with the provided main program. This could involve running a series of test cases the main program provides or creating additional test cases to ensure robustness.