Sabancı University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Fall 2022

Homework 7 – Sparse matrix multiplication

Due: 08/01/2023, Sunday, 11:55

PLEASE NOTE:

Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; you are expected to take actions accordingly!

You HAVE TO write down the code on your own.
You CAN NOT HELP any friend while coding.
Plagiarism will not be tolerated!!

Submission rules (PLEASE READ, IMPORTANT)

Your submission will consist of only TWO code files:

1. `SparseMatrix.h`
2. `SparseMatrix.cpp`

Please don't change the names of these files.

Don't upload an XCode project or a Visual Studio project. Just the two code files above.

Place these two code files AND ONLY THESE TWO FILES inside a folder named with the following convention:

`SUCourseUserName_YourLastname_YourName_HWnumber`

Your SUCourse user name is your SUNet username that is used for your sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the folder name. For example, if your SUCourse username is cago, your first name is Taha Çağlayan, and your last name is Özbugsızkodyazaroğlu, then the folder name must be:

`cago_Ozbugsizkodyazaroglu_TahaCaglayan_1`

Compress this folder using a compression program such as WinZip or WinRAR. Please use "zip" compression. "rar", "7z" or any other compression mechanisms are NOT allowed. Our homework processing system only works with zip files. Therefore, make sure that the resulting compressed file has a zip extension. Check that your compressed file opens up correctly and it contains all your code files. You will receive no credits if your compressed

folder does not expand or it does not contain the correct files. The name of the zip file follows the same convention. The zip file for the homework submission by the student mentioned above would be:

```
cago_Ozbugsizkodyazaroglu_TahaCaglayan_1
```

Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.). Successful submission is one of the requirements of the homework. If for some reason you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Overview

We will provide you with the two files `SparseMatrix.h` and `SparseMatrix.cpp`, which contain a class for sparse matrices. The class will use the same data structure that was used in homework 2 to represent sparse matrices. Please refer to that document for a description of the data structure, as well as matrix files.

Your job will be to add to this class a function, **`mat_mul`**, that will multiply two matrices and return the result as a new matrix.

SparseMatrix class

The class given includes the following data members:
`row_node* head_`: a pointer at the head of the sparse matrix (0th row).
`int num_rows_`: number of rows in the matrix.
`int num_columns_`: number of columns in the matrix.

> Note: you'll notice that data members inside the class have the suffix "_" at the end of their name. This is the method followed by Google in their C++ codebase. It helps distinguish class data members from function parameters. Check the Google C++ Style Guide for more details.

The class given includes the following functions:

- `SparseMatrix(const string& filename)`: reads the matrix stored in the file `filename`. If the file has any problems, the constructor will throw an exception.
- `SparseMatrix(row_node * head, int num_rows, int num_columns)`: initializes the head pointer of the created matrix to `head`, the number of rows of the created matrix to `num_rows`, and the number of columns of the created matrix to `num_columns`.
- `SparseMatrix(const SparseMatrix& rhs)`: A copy constructor.
- `SparseMatrix& operator=(const SparseMatrix& rhs)`: An assignment operator.
- `~SparseMatrix()`: A destructor.

Additionally, the class has the following friend function:

- `ostream& operator<<(ostream& out, const SparseMatrix& rhs)`: prints the `rhs` SparseMatrix object to the output stream `out`.

All the functions above are already declared in the file SparseMatrix.h and defined in the file SparseMatrix.cpp. You don't need to implement any of them.

Your task for this homework is to add an additional function to the class. The function is described in the next section. You can add other helper functions as well to implement `mat_mul`.

Matrix Multiplication

Matrix multiplication is a simple process. It is actually an extension of the matrix-vector multiplication procedure you worked with in homework 1. Instead of the RHS being a single vector, it is a matrix.

When multiplying a matrix by another matrix, the number of columns in the left-hand side (LHS) matrix and the number of rows in the right-hand side (RHS) matrix must be equal.

The result matrix will have the same number of rows as the LHS matrix, and the same number of columns as the RHS matrix. This is shown in the following visualization:

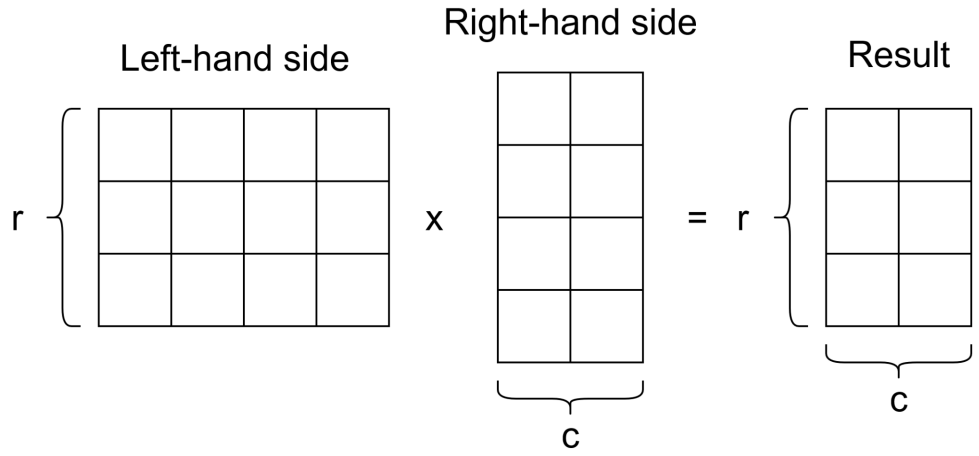

Figure 1: the resulting matrix of multiplying two matrices will have the same number of rows as the LHS, and the same number of columns as the RHS.

The best way to explain matrix-matrix multiplication is through visualization. First, let's calculate the 0th row in the result matrix. To calculate the element in the 0th column and the 0th row of the result matrix (Result[0][0]), we multiply each element in the 0th row in LHS, by an element in the 0th column in the RHS, and then sum up the multiplications and sum the results (as shown in the figure below).

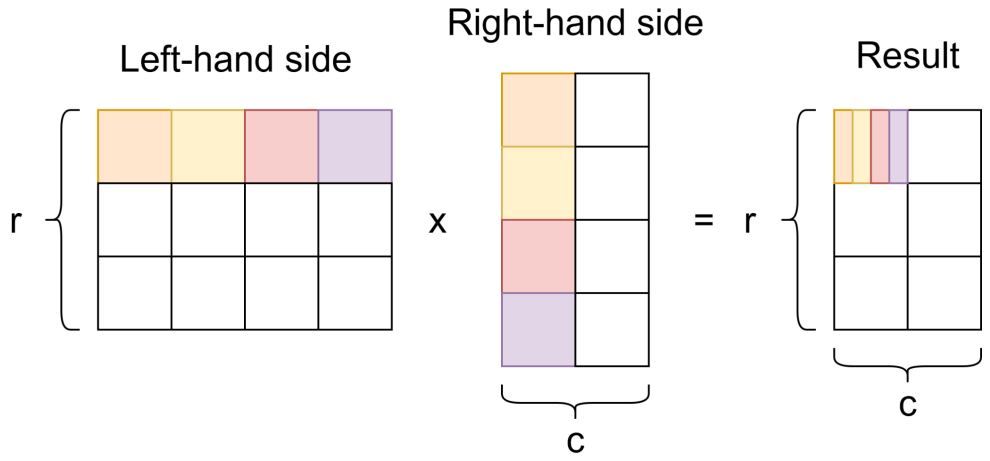

Figure 2: the value at Result[0][0] = LHS[0][0]*RHS[0][0] + LHS[0][1] * RHS[1][0] + LHS[0][2] * RHS[2][0] + LHS[0][3] * RHS[3][0]

Then, to calculate the element in the result matrix at row 0 and column 1 (Result[0][1]), we will multiply each element in the 0th row of LHS with a corresponding element in the *column at index 1* in the RHS matrix and sum the results (as shown in the figure below).

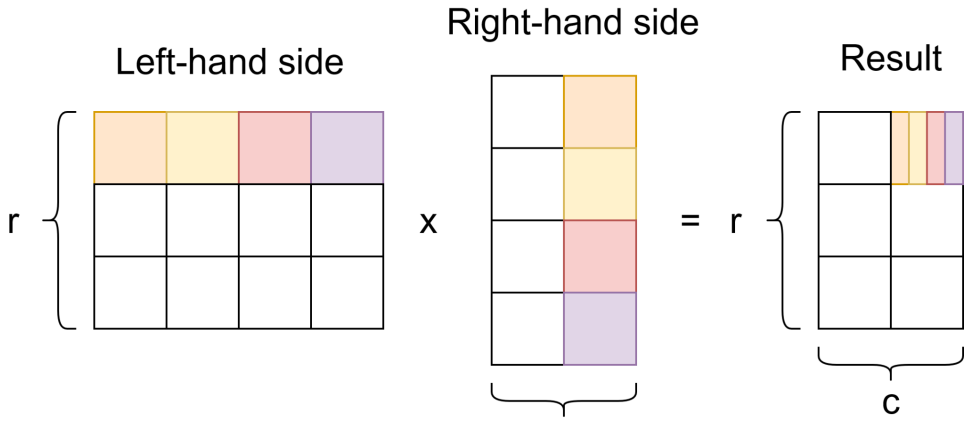

Figure 3: the value at Result[0][1] = LHS[0][0]*RHS[0][1] + LHS[0][1] * RHS[1][1] + LHS[0][2] * RHS[2][1] + LHS[0][3] * RHS[3][1]

Now, let's calculate row 1 in the result matrix. To calculate the result element at row 1 and column 0 (Result[1][0]), we multiply each element in row 1 of the LHS matrix, with the corresponding element in column 0 of the RHS matrix and sum the results (as shown in the figure below).

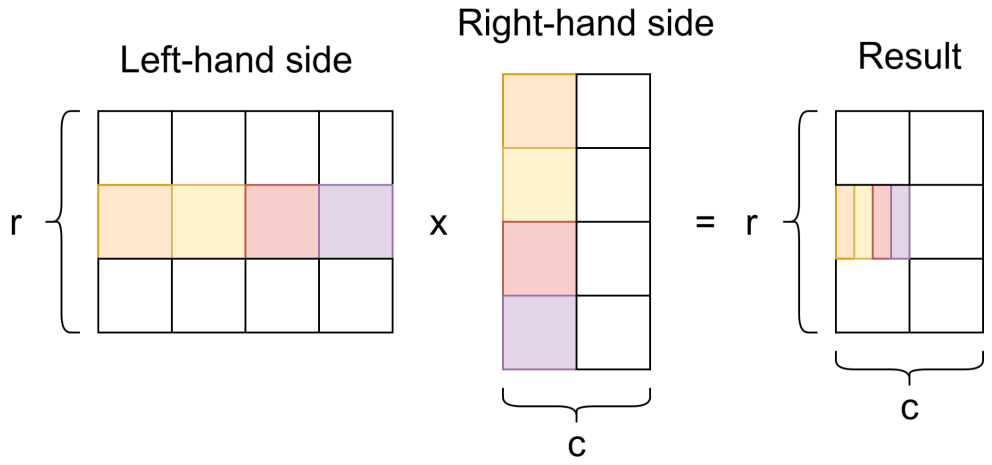

Figure 3: the value at Result[1][0] = LHS[1][0]*RHS[0][0] + LHS[1][1] * RHS[1][0] + LHS[1][2] * RHS[2][0] + LHS[1][3] * RHS[3][0]

And to calculate the result at column 1 of row 1 in the result matrix (Result[1][1]), we will multiply each element in row 1 of the LHS matrix with a corresponding item in column 1 in the RHS matrix and sum the results (as shown in the figure below).

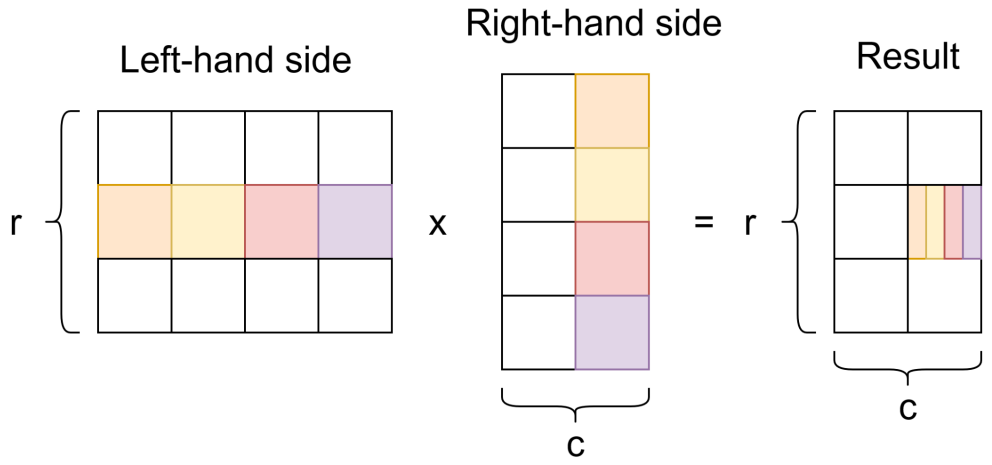

Figure 3: the value at Result[1][1] = LHS[1][0]*RHS[0][1] + LHS[1][1] * RHS[1][1] + LHS[1][2] * RHS[2][1] + LHS[1][3] * RHS[3][1]

We can summarize the pattern of the calculation like this: to calculate the result at the cell in row R and column C in the result matrix, we need to multiply each element in row R of LHS with an element in column C in RHS, and sum the results of these multiplications.

The following is an implementation of matrix multiplication in which the LHS and RHS matrices are represented using vectors of vectors, and in the returned matrix is also a vector of vectors.

```
vector<vector<int>> mat_mul_vec(vector<vector<int>>&lhs,
vector<vector<int>>& rhs){
   int num_rows_result = lhs.size();
   int num_columns_result = rhs[0].size();
```

```
    vector<vector<int>> result(num_rows_result,
vector<int>(num_columns_result));
    for (int result_row = 0; result_row < lhs.size(); result_row++) {
        for (int result_column = 0;
             result_column < rhs[0].size(); result_column++) {
            int sum = 0;
            for (int i = 0; i < lhs[0].size(); i++) {
                sum += lhs[result_row][i] * rhs[i][result_column];
            }
            result[result_row][result_column] = sum;
        }
    }
    return result;
}
```

You may use this website for an additional visualization:
https://www.mathsisfun.com/algebra/matrix-multiplying.html
And you can use this website to calculate matrix multiplications and see the solution steps:
https://matrix.reshish.com/multiplication.php

Your job in this homework is to add and implement the following member function to the SparseMatrix class:

`SparseMatrix mat_mul(const SparseMatrix& rhs) const`

This function will multiply the calling object (`*this`) with the matrix `rhs`, and will return the resulting matrix by value. **Your function must carry out this multiplication in a multi-threaded fashion.**

Example usage

```
#include "SparseMatrix.h"

int main(){
    // reads the matrix stored in m1.txt
    SparseMatrix lhs("m1.txt");
    // reads the matrix stored in m2.txt
    SparseMatrix rhs("m2.txt");
    // Stores in the SparseMatrix "result" the result of the following
operation
    // lhs * rhs
    SparseMatrix result = lhs.mat_mul(rhs);
    return 0;
}
```

Hint: it is better if you first implement this function using a single thread, make sure it is correct by checking if it does the multiplication correctly on a few sample matrices, and then convert it to a multi-threaded implementation.

Rules and assumptions

When implementing the function mat_mul, you can make the following assumptions:

1. You may assume that the calling object (`*this`) and the RHS matrix (`rhs`) will have at least 1 row and at least 1 column.
2. You may assume that the number of columns in the calling object (`*this`) is equal to the number of rows in the RHS matrix (`rhs`).
3. You may assume that your code is going to be tested on valid matrix files suitable for the matrix multiplication operation (two samples are given with this homework bundle).

The following are the rules you must follow in your solution:

1. If your solution does not do the multiplication using multiple threads, you will get 0 points.

Good Luck!
Amro