CSE 220: Systems Fundamentals I

Stony Brook University
Homework Assignment #4
FALL 2023

Due: Sunday Oct 22 at 11: 59 PM EST

Learning Outcomes

- Mastering C programming for pointers and pointer dereferencing

Overview

In this assignment you will be implementing a program to perform simple linear algebra computations: sparse matrix calculation, matrix addition, multiplication and transpose. The matrices and formulas containing these operations will be implemented using pointer dereferencing. You are not allowed to use arrays to implement the operation. You can initialize the matrices using arrays before calling the functions. Assumption is that all matrices have been initialized before calling the required functions. However, after passing matrices to the operations/functions, all calculations within the functions will be done using pointer dereferencing. ***No marks will be given if direct arrays approach is used.***

Implement all operations in matrix.c file. During grading, only your `matrix.c` file will be copied into the grading framework's directory for processing. Make sure all of your code is self-contained in that file.

Mathematical Preliminaries

Matrices

A matrix is a rectangular array of numbers arranged in rows and columns. The size of a matrix is determined by the number of rows and columns it contains. Matrices are widely used in mathematics, physics, engineering, computer science, and many other fields.

Matrix Addition

Matrix addition is an operation that involves adding the corresponding elements of two matrices of the same size. Mathematically, if A and B are both m x n matrices (with m rows and n columns), then their sum, denoted as C, is an m x n matrix where the entry in the i-th row and j-th column of C is given by:

$$C_{ij} = A_{ij} + B_{ij}$$

In this formula, A_{ij} is the entry in the i-th row and j-th column of matrix A, and B_{ij} is the entry in the i-th row and j-th column of matrix B. Note that in order to add two matrices, they must be of the same size, i.e., they must have the same number of rows and the same number of columns. (You need to check this compatibility in your function as well)

Matrix Multiplication

Matrix multiplication is a way of combining two matrices together to produce a new matrix. The resulting matrix has the same number of rows as the first matrix and the same number of columns as the second matrix. Mathematically, if A is an m x n matrix and B is an n x p matrix, then the product of A and B, denoted as C, is an m x p matrix where the entry in the i-th row and j-th column of C is given by:

$$C_{ij} = \sum_{k=1}^{n} \left(A_{ik} B_{kj} \right)$$

In this formula, A_{ik} is the entry in the i-th row and k-th column of A, and B_{kj} is the entry in the k-th row and j-th column of B. In our scripts, matrix multiplication will be denoted C = A*B. (In your function, you need to check if the number of columns of A is equal to the number of rows of B).

Matrix Transpose

Matrix transpose is an operation that involves flipping a matrix over its diagonal. Mathematically, if A is an m x n matrix, then its transpose, denoted as A^T, is an n x m matrix where the entry in the i-th row and j-th column of A^T is given by:

$$\left(A^T\right)_{ij} = A$$

Sparse Matrix

A 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. For this homework, we will consider a matrix sparse if the numbers of non zero values are less than or equal to the largest dimension of the matrix.

Why to use Sparse Matrix instead of simple 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.

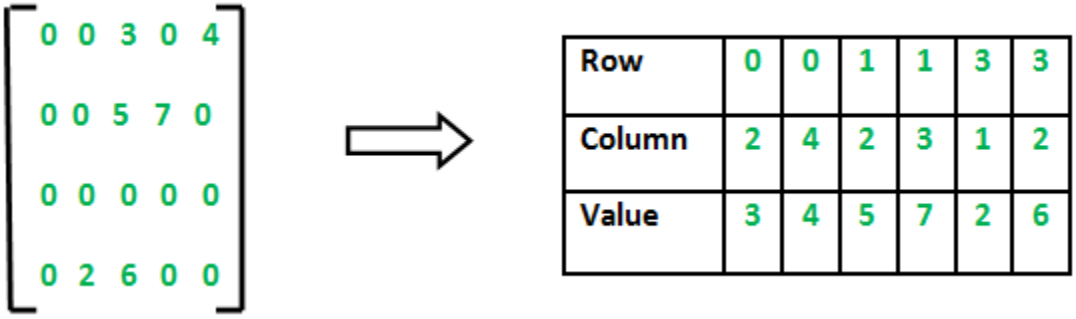

Figure -1 Sparse matrix representation. This figure is just to explain the concept of sparse matrix. It does not satisfy our constraint. We define that the number of non zero elements should

be equal to or less than the largest dimension. Here the largest dimension is 5. However, nonzero elements are 6.

Sparse Matrix Representations can be done in many ways following are two common representations:

1. Array representation
2. Linked list representation

Since we have not yet covered the topic of advanced pointer usage with dynamic memory allocation, we will use the array representation for sparse matrix representation in this homework.

2D array is used to represent a sparse matrix in which there are three rows named 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)
- See Figure 1.

Remember

In array notation A[row][i] is equivalent to (*(*(A + row) + i)) and A[j] means *(A+j); You will be using such expressions in your homework.

You will implement the following functions:

int SparseMatrix (int** M, int** S, int* D)

M : 2D matrix which has more zero elements than non zero elements

S : 2D sparse matrix. However, 3 x m. m is equal to the largest dimension of M

D: the 1D matrix contains the dimensions of M. It's a two element array. First element gives the dimension along the x-axis while the second element gives the dimension along the y-axis.

The function will return the number of non-zero elements if the number of non-zero elements are equal to or less than m (the largest dimension of M). In this case, Sparse matrix S will be created. Otherwise, the given M matrix is not sparse and function will return -1.

Note: In case the number of non zero elements is less than the maximum dimension of M, the remaining elements of S will be filled with zeros. See Figure 2.

For this homework, the sparse matrix will have 3 x m dimension. Where m is the maximum dimension of the given matrix. For example, if we have 5 non zero elements and maximum dimension is 7

rows	1	2	3	4	5	0	0
Column	1	2	3	4	5	0	0
values	3	3	3	3	3	0	0

Figure 2: Yellow part is the real sparse matrix.

int Addition(int** M, int** N, int** A, int* D) (A= M +N)

M, N and A are 2D matrices. D is a 1D array with 1x6 elements with the first two elements giving the dimensions of M (first element gives x axis and the second element gives y axis), the next two elements giving the dimensions of N, and the last two elements giving the dimension of A. Matrix A contains the outcome of the addition operation.

Potential Cases

- Dimensions of M and N are compatible (M and N have same dimensions) for algebraic addition of matrices:
    - A is compatible (A is the exact size of result of addition). Straight forward operation. Matrix A will have the answer. **Return 1** (success).
    - If A is not compatible, it is oversized enough to contain the actual result. Matrix A will have the answer. **Return 2**.
    - If A is not compatible and does not have enough space to contain the whole result. In this case it will contain part of the result. The one that overlaps with the space of the result/answer with the space of A. **Return -3**

- Dimensions of M and N are not compatible for algebraic addition (M and N have different dimensions). We can not do the addition. However, we move on with what we have and try to do the addition of elements that are possible.
    - Step 1: The space of M and N that overlaps should be added. You can call it intersection of the elements that are common to both matrices
        - M= [[1,2,3] , [4,3,6], [7,8,9]] → 3 x 3 matrix
        - N= [[1,2], [3,4]] → 2 x 2 matrix
        - Both are not compatible. However, the first two rows and columns overlap (call it intersection of the two matrices). The elements of the common space will be added.
    - If A has enough space or dimension to hold the result of addition from the previous step, A should hold it and the function should **return -1**.
    - If A does not have enough space or dimension, then A should hold the elements of the results from step 1 that overlap / intersect its dimension and the function should **return -2**.

Note:

Matrix A should be initialized to zero. No garbage values.

int Multiplication (int** M, int** N, int** A, int* D): A = M x N

M, N and A are 2D matrices. D is a 1D array with 1x6 elements with the first two elements giving the dimensions of M (first element gives x-axis and the second element gives y-axis), the next two elements giving the dimensions of N and the last two elements giving the dimensions of A respectively. Matrix A contains the outcome of the multiplication operation.

Potential Cases

- Dimensions of M are N are compatible (#cols of M = #rows of N) for algebraic multiplication of matrices:
    - If A is compatible (A is the exact dimension of the result of multiplication). Straight forward operation. Matrix A will have the answer. **Return 1** (success)
    - If A is not compatible, it is oversized enough to contain the actual result. Matrix A will have the answer. **Return 2**.
    - If A is not compatible, not enough space to contain the whole result. In this case it will contain part of the result. The one that overlaps with the space of the result/answer. **Return -3.**

- Dimensions of M and N are not compatible for algebraic multiplication. However, we move on with what we have and try to do the multiplication (of elements) that is possible.
    - Step 1:
        - M= [[1,2,3],
          [4,3,6],
          [7,8,9]] → 3 x 3 matrix
        - N= [[1,2], (Hint consider N tobe [[1,2,0], [3, 4, 0]])
          [3,4]] → 2 x 2 matrix
        - MxN → [(1+6=7) (2+8 = 10)
          (4+9=13) (8+12=20)
          (7+24=31) (14+32=46)] will result 3 x 2 matrix
        - matrix A will hold this result
    - If A has enough space or dimension to hold the result of multiplication from the previous step, A should hold it and the function should **return -1**.
    - If A does not have enough space or dimension to hold the result of multiplication from the previous step, then A should hold the elements of the results from step 1 that overlap / intersect its dimension and the function should **return -2**.

Note:

Matrix A should be initialized to zero. No garbage values

int Transpose (int** A, int** AT, int* D)

A, AT are 2D matrices. D is a 1D array with 1x4 elements with the first two elements giving the dimension of A (first element gives x-axis and the second element gives y-axis), the next two elements giving the dimension of AT. Matrix AT contains the outcome of the Transpose operation.

- The function will **return 1** if the dimension of matrices are compatible (AT has the dimension of transpose of A),
- If dimensions are not compatible then:
    - If AT has enough space or dimension to hold the result of transpose, AT should hold it and the function should **return 2**.
    - If AT does not have enough space or dimension, then AT should hold the elements of transpose that overlap / intersect its dimension and the function should **return -1**

Note:

Matrix AT should be initialized to zero. No garbage values

Test cases:

int SparseMatrix (int** M, int** S, int* D)

Case 1:

M= $\begin{bmatrix} 0 & 4 \\ 0 & 5 \\ 0 & 0 \\ 2 & 0 \end{bmatrix}$

4×2

Sparse Matrix:
So Sparse Matrix Dimension, S= $3 \times m$, hence $m=4$ and the number of non-zero elements is 3 that is less than m. So this matrix satisfies the sparse matrix condition and it's compatible for creating sparse matrices. **Its return 3**.

Hence, the sparse matrix dimension is S= 3×2.

Row	0	1	3	0
Column	1	1	1	0
Value	4	5	2	0

Note that: Last column 3 all elements are 0 as no non-zero value is available to put there.

Case 2:

M= $\begin{bmatrix} 2 & 3 & 0 & 5 \\ 3 & 0 & 3 & 1 \\ 4 & 0 & 3 & 0 \\ 0 & 0 & 6 & 0 \end{bmatrix}$

4×4

Sparse Matrix:

Sparse Matrix Dimension= 3×m, hence m=4 and the number of non-zero elements is 9 that is higher than m. Therefore, this matrix does not satisfy the sparse matrix condition. **So the function returns -1.**

int Addition(int** M, int** N, int** A, int* D)

Case 1:

M	N	A	D	Compatible/Return
$\begin{bmatrix} 2 & 3 \\ 1 & 3 \\ 4 & 5 \end{bmatrix}$ 3×2	$\begin{bmatrix} 3 & 3 \\ 7 & 2 \\ 1 & 5 \end{bmatrix}$ 3×2	$\begin{bmatrix} 5 & 6 \\ 8 & 5 \\ 5 & 10 \end{bmatrix}$ 3×2	[3 2 3 2 3 2]	Yes/Return 1 M, N, and A are compatible

Case 2:

M	N	A	D	Compatible/Return
$\begin{bmatrix} 2 & 3 \\ 1 & 3 \\ 4 & 5 \end{bmatrix}$ 3×2	$\begin{bmatrix} 3 & 3 \\ 7 & 2 \\ 1 & 5 \end{bmatrix}$ 3×2	[5 6 0 8 5 0 5 10 0] 3×3	[3 2 3 2 3 3]	Return 2 M and N are compatible but not A. However, A can hold the actual addition

Case 3:

M	N	A	D	Compatible/Return
$\begin{bmatrix} 2 & 3 \\ 1 & 3 \\ 4 & 5 \end{bmatrix}$ 3×2	$\begin{bmatrix} 5 & 6 \\ 8 & 5 \\ 5 & 10 \\ 5 & 10 \end{bmatrix}$ 4×2	[7 9 9 8] 2 x 2	[3 2 4 2 2 2]	No/Return -2 A can hold part of the final addition

Case 4:

M	N	A	D	Compatible/Return
$\begin{bmatrix} 2 & 3 \\ 1 & 3 \\ 4 & 5 \end{bmatrix}$ 3×2	$\begin{bmatrix} 5 & 6 \\ 8 & 5 \\ 5 & 10 \\ 5 & 10 \end{bmatrix}$ 4×2	[7 9 9 8 9 15 0 0 0 0] 5 x 2	[3 2 4 2 5 2]	No/Return -1 A can hold the complete addition

Case 5:

M	N	A	D	Compatible/Return
$\begin{bmatrix} 2 & 3 \\ 1 & 3 \\ 4 & 5 \end{bmatrix}$ 3×2	$\begin{bmatrix} 3 & 3 \\ 7 & 2 \\ 1 & 5 \end{bmatrix}$ 3×2	[5 6 8 5] 2×2	[3 2 3 2 2 2]	Return - 3 M and N are compatible but not A. However, A can hold part of the actual addition

int Multiplication (int** M, int** N, int** A, int* D)

Case 1:

M	N	D	A	Compatible/Return
$\begin{bmatrix} 2 & 3 \\ 5 & 4 \\ 2 & 3 \end{bmatrix}$ 3×2	$\begin{bmatrix} 3 & 2 & 1 \\ 1 & 3 & 4 \end{bmatrix}$ 2×3	[3 2 2 3 3 3]	$\begin{bmatrix} 9 & 13 & 14 \\ 19 & 22 & 19 \\ 9 & 13 & 14 \end{bmatrix}$ 3×3	Yes/Return1

Case 2:

M	N	D	A	Compatible/Return
$\begin{bmatrix} 2 & 3 \\ 5 & 4 \\ 2 & 3 \end{bmatrix}$ 3×2	$\begin{bmatrix} 3 & 3 \\ 7 & 2 \\ 1 & 5 \end{bmatrix}$ 3×2	[3 2 3 2 2 2]	[27 12 43 23] 2 x 2	No/ Return -2 A can hold part of the final result

Case 3:

M	N	D	A	Compatible/Return
$\begin{bmatrix} 2 & 3 \\ 5 & 4 \\ 2 & 3 \end{bmatrix}$ 3×2	$\begin{bmatrix} 3 & 3 \\ 7 & 2 \\ 1 & 5 \end{bmatrix}$ 3×2	[3 2 3 2 4 3]	[27 12 0 43 23 0 27 12 0 0 0 0] 4 x 3	No/ Return -1 A can hold the complete result

Case 4:

M	N	D	A	Compatible/Return
$\begin{bmatrix} 2 & 3 \\ 5 & 4 \\ 2 & 3 \end{bmatrix}$ 3×2	$\begin{bmatrix} 3 & 2 & 1 \\ 1 & 3 & 4 \end{bmatrix}$ 2×3	[3 2 2 3 4 4]	[9 13 14 0 19 22 19 0 9 13 14 0 0 0 0 0] 4×4	Return 2 M and N are compatible A will have the right/complete result

Case 5:

M	N	D	A	Compatible/Return
$\begin{bmatrix} 2 & 3 \\ 5 & 4 \\ 2 & 3 \end{bmatrix}$ 3×2	$\begin{bmatrix} 3 & 2 & 1 \\ 1 & 3 & 4 \end{bmatrix}$ 2×3	[3 2 2 3 2 4]	[9 13 14 0 19 22 19 0] 2×4	Return -3 M and N are compatible A does not have the right result dimensions

int Transpose(int** A, int** AT, int* D)

Case 1:

A	AT	D	Compatible/Return
$\begin{bmatrix} 9 & 9 \\ 1 & 2 \\ 9 & 3 \end{bmatrix}$ 3×2	$\begin{bmatrix} 9 & 1 & 9 \\ 9 & 2 & 3 \end{bmatrix}$ 2×3	[3 2 2 3]	Yes/ Return 1

Case 2:

A	AT	D	Compatible/Return
$\begin{bmatrix} 9 & 9 \\ 1 & 2 \\ 9 & 3 \end{bmatrix}$ 3×2	[9 1 9 2 0 0] 3 x 2	[3 2 3 2]	No/ Return -1 Does not have the correct result

Case 3:

A	AT	D	Compatible/Return
$\begin{bmatrix} 9 & 9 \\ 1 & 2 \\ 9 & 3 \end{bmatrix}$ 3×2	[9 1 9 0 9 2 3 0 0 0 0 0] 4 x 4	[3 2 4 4]	No/ Return 2 Has the correct result

Running, Testing and Submitting Your Code

Running test cases using Criterion is optional for this homework. You can have your way of testing the code. However, as we move to the next assignment we will adopt this framework for testing the homework. The testing framework is provided in the starter code. Try it and familiarize yourself.

To run the provided unit tests (written using Criterion):

- `make` to build your code
  - The directory structure for this homework assignment is different from HW #3. Binaries are now located in the `bin` directory. If you want to understand more, feel free to explore the provided `makefile`.
- `make test` to test your code
- `make gcov` to check the test coverage (see below)

The provided test cases are nowhere near comprehensive. It is your responsibility to write additional test cases to verify the correctness of your code. If you create new test cases using Criterion (instead of `printf`), then both `make tests` and GitHub will run them automatically (GitHub only when you `git push` your work).

Remember to regularly `git commit` your work. Occasionally, `git push` your work to run the same tests on the git server ***and to submit your work for grading*** by the due date.

Grading Notes

During grading, only your `matrix.c` file will be copied into the grading framework's directory for processing. Make sure all of your code is self-contained in that file.

Extensions, resubmissions and regrades will not be granted because a student did not use git properly to submit the assignment.

Academic Honesty Policy

Academic honesty is taken very seriously in this course. By submitting your work for grading you indicate your understanding of, and agreement with, the following Academic Honesty Statement:

1. I understand that representing another person's work as my own is academically dishonest.
2. I understand that copying, even with modifications, a solution from another source (such as the web or another person) as a part of my answer constitutes plagiarism.
3. I understand that sharing parts of my homework solutions (text write-up, schematics, code, electronic or hard-copy) is academic dishonesty and helps others plagiarize my work.
4. I understand that protecting my work from possible plagiarism is my responsibility. I understand the importance of saving my work such that it is visible only to me.
5. I understand that passing information that is relevant to a homework/exam to others in the course (either lecture or even in the future!) for their private use constitutes academic dishonesty. I will only discuss material that I am willing to openly post on the discussion board.
6. I understand that academic dishonesty is treated very seriously in this course. I understand that the instructor will report any incident of academic dishonesty to the University's Academic Judiciary.
7. I understand that the penalty for academic dishonesty might not be immediately administered. For instance, cheating on a homework assignment may be discovered and penalized after the grades for that homework have been recorded.
8. I understand that buying or paying another entity for any code, partial or in its entirety, and submitting it as my own work is considered academic dishonesty.
9. I understand that there are no extenuating circumstances for academic dishonesty. 8