CS 435 Programming Project (Fall 2014)
Solving a System of Equations using Gaussian Elimination

A **system of linear equations** is a set of linear equations involving the same *n* variables x_1 ... x_n. A system of linear equations can be represented using a matrix equation in the form $\boldsymbol{Ax} = \boldsymbol{y}$, where $\boldsymbol{A}$ is a coefficient matrix, $\boldsymbol{x}$ is a column vector with *n* variable entries x_1 ... x_n, and $\boldsymbol{y}$ is a column vector storing the constants. For example, given the system of linear equations shown in **(a)**, their matrix equation representation is **(b)**.

$$\begin{aligned} 3x_1 + 5x_4 &= 8 \\ 2x_2 - 7x_3 &= 9 \\ 5x_1 - 3x_2 - 10x_4 &= -8 \\ 6x_2 - 5x_4 &= 1 \end{aligned}$$

(a)

$$\begin{bmatrix} 3 & 0 & 0 & 5 \\ 0 & 2 & -7 & 0 \\ 5 & -3 & 0 & -10 \\ 0 & 6 & 0 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 8 \\ 9 \\ -8 \\ 1 \end{bmatrix}$$

(b)

A solution to a system of linear equations is the assignment of *n* values to x_1 ... x_n such that each equation is satisfied. For example, **(a)**'s solution is: $x_1 = 1, x_2 = 1, x_3 = -1, x_4 = 1$.

In this multi-step programming project you will implement a program that solves systems of linear equations using **Gaussian Elimination** (also known as **Row Reduction**). For simplicity, we will only consider systems of equations that have *n* linear equations in terms of *n* variables. Thus, their coefficient matrix $\boldsymbol{A}$ is *n* x *n* in size.

A **sparse matrix** is a matrix where a large proportion of the elements have a value of zero. In a linked list representation of such a matrix zero value elements are not represented, saving space.

In this assignment you will use the following linked-list-based implementation of a sparse matrix. Each non-zero entry in the matrix will be represented using a node with five fields: ***row*** (integer), ***col*** (integer), ***value*** (real number), ***rowLink*** (pointer), and ***colLink*** (pointer).

Each node will belong to exactly two singly linked lists: a row linked list and a column linked list. The nodes of each row are organized into a linked list via their ***rowLink*** pointers, with a head pointer containing a link to the first node of the list. Similarly, the elements of each column are organized into a linked list via their ***colLink*** pointers. Each column has a head pointer that points to the first node in the list or *null* if the column is empty.

You will use this sparse matrix implementation to represent the coefficient matrix $\boldsymbol{A}$ and the column vector $\boldsymbol{y}$. The values of $\boldsymbol{y}$ will be represented using an extra column with a column field indicated with a ***col*** of "-1." Additionally, for this additional column only, zero values will be explicitly represented using a node with a ***value*** of 0. Thus, to represent an *n* x *n* system of equations you will create a sparse matrix with *n* rows and *n* + 1 columns.

This implementation is shown explicitly for (a) in Figure 1.

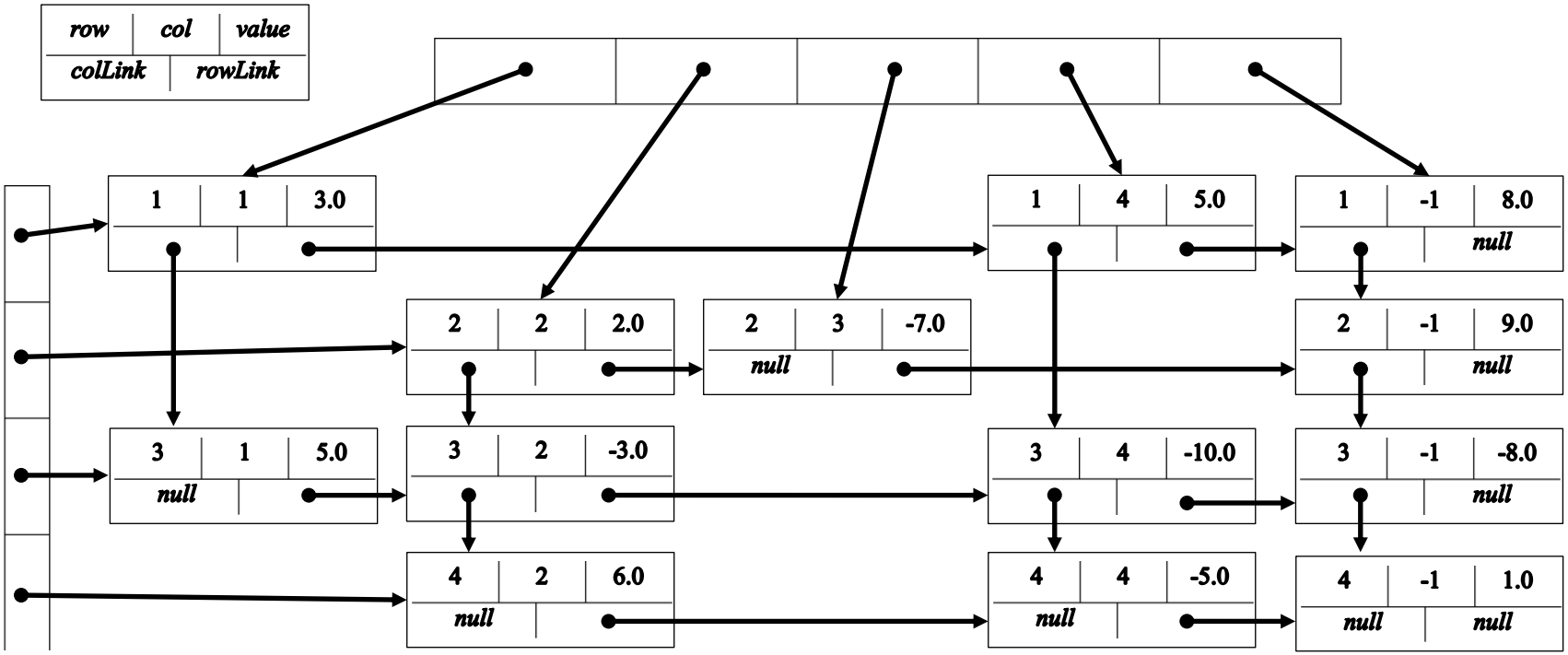

Figure 1. The linked sparse matrix representation of (a).

Step 1 (30 points)	**Implement Linked Sparse Matrix**	**Due Date: 10/30**

The first step of this project is to implement the linked sparse matrix data structure described on Page 1 and shown on Page 2. This includes correctly implementing the node structure and linked spare matrix data structure **(5 pts)**. Additionally, implement the following:

Initialize(n, S[]) (8 pts): Construct a new sparse matrix of size *n* x (*n*+1) to represent the linear equations stored in S. S is an array of *n* strings containing linear equations in the form "c_1x1 $\pm$ c_2x2 $\pm \cdots \pm$ c_nx$n = y$" where c_1 ... c_n and *y* are real numbers and x*i* correlates to a variable $x_i, 1 \leq i \leq n$. There is no order to the equations and no bound on *n*. A subtracted term implies a negative constant. Any terms with a zero constant are ignored (though the test input does not explicitly include zero terms). For simplicity you can assume there is no whitespace in the strings and the variables are input in order according to *i*. For example, the term $2x_2 - 7x_3 = 9$ from (a) would be represented using the string "2x2-7x3=9" (without quotes). If the constant is 1 then it will not be included, e.g., $-x_1 + x_2 = -3$ will be input as "-x1+x2=-3" (without quotes).

Insert(i, j, c) (8 pts): Insert a node with real value *c* into row *i* and column *j* of the matrix. If there is already a non-zero element in this position of the matrix it **cannot** be overwritten.

Output() (5 pts): Print the matrix. The entries in each row/column should be evenly spaced. The entire matrix should be printed out (including zero values).

Test Cases (4 pts): Implement and execute all test cases listed for Step 1 on Page 5.

Step 2 (40 points)	**Sparse Matrix Row Operations**	**Due Date: 11/16**

AddRow(i, c, j) (7 pts): Add the row i, where each row value is multiplied by a real constant c, to row j. Any row value in j that becomes zero must be removed.

SubtractRow(i, c, j) (7 pts): Subtract the row i, where each row value is multiplied by a real constant c, from the row j. Any row value in j that becomes zero must be removed.

SwapRows(i, j) (14 pts): Swaps the rows i and j. This can only be implemented by updating nodes' pointers. No new nodes can be created and no nodes can be deleted.

Delete(i, j) (6 pts): Delete the node at position (i, j) in the matrix.

Test Cases (6 pts): Implement and execute all test cases listed for Steps 1-2 on Page 5.

Step 3 (30 Points)	**Solving Systems of Linear Equations**	**Due Date: 12/4**

Triangulate() (10 pts): Triangulate the matrix, using Add, Subtract, and Swap, according to the algorithm described in class.

Solve() (8 pts): Find the solution to the system of linear equations for a triangulated matrix to solve the equation ***Ax = y***. Returns an array of size *n* that stores the values of x_1 ... x_n.

IsSolution(x) (7 pts): Checks if **x** solves the equation. ***x*** is an array of size *n*. Multiplies the matrix with column vector ***x*** to determine if ***x*** stores the correct solution to ***Ax = y***.

Test Cases (5 pts): Implement and execute all test cases listed for Steps 1-3 on Page 5.

This project is separated into three steps. Each step is broken down into several parts. You cannot move on to the next step of the project until every part of the previous step is implemented correctly or you are cleared to move onto the next step by the TA (you will be explicitly told that you cannot move on when you receive a grade for a step). Correcting a part of a step after a grade was given will result in no credit.

Implementation Guidelines

You will receive a grade of **zero** for your submission if you do not follow these guidelines.

1. You must implement the linked sparse matrix data structure described on Page 1 and shown on Page 2. No other implementations of a sparse matrix will be allowed.
2. You must implement all of your own data types and data structures.
3. You cannot convert a sparse matrix into another data structure at any point.
4. The correct implementation of the required functions is dependent on the correct implementation of the specified data structure. If you have the required functions working with an incorrect data structure you will receive no credit.
5. A sparse matrix can **NEVER** be converted into a string. **No exceptions**.
6. The sparse matrix cannot contain any nodes that store a value of "0", except for the last column which represents ***y***.
7. The sparse matrix functions must be implemented using the algorithms described in class. No other implementations will receive credit. Additionally:
    a. AddRows and SubtractRows must be implemented as separate functions.
8. You can implement additional functions if needed, as long as they do not violate any of the above guidelines. You can also name any function as you wish, as long as it's a meaningful name.
9. This project lends itself to object oriented design and implementation. A procedural implementation is also acceptable.
10. The preferred language for this project is C++. If you do not know C++ you can use C or Java with no penalty. No other languages are permitted.

Submission Guidelines

- Email your submission to cro3@njit.edu with the subject "CS 435 Project" followed by the step you are submitting, e.g. "CS435 Project Step 1".
- Submit only what is required for the current step.
- Submit **ONLY** the source code for your submission. Only submit .cpp/.cc/.h/.hh/.c/.java files. If you have more than one source file you can combine them into a single Zip.
- Your code **MUST** compile and execute on NJIT's AFS computers (afsconnect1.njit.edu or afsconnect2.njit.edu) using GCC, G++, or javac.
    - This **WILL** be a problem if you are using Microsoft Visual Studio. I guarantee it.
    - Do not use any non-standard libraries (e.g., conio.h).
    - If your Step 1 submission does not compile on AFS you will be given 24 hours to correct the problem with no penalty. After 24 hours it will be considered late.
    - In all other situations, if your submission does not compile it will receive a zero.
- You can modify a previous step's code for a future step, if necessary.

Academic Honesty Policy

Copying code from the internet, another student, or from any other source (including books, YouTube videos, 8-tracks, etc.), **even if it is just a single line of code**, is **strictly prohibited**. Any cases of copying will result in everyone involved being reported to the Dean of Students and everyone involved **will receive a zero for the entire project** (not just the copied part/step).

- Your code is yours and yours alone. You are solely responsible for it.
- Do not share your implementation or code with other students.
- Do not look at each other's code.
- Do not share your code as a "guide" for other students.
- Do not put your code in a publicly accessible place (e.g., Github, Google Code, etc.).

Test Cases

Your project **MUST** automatically run all of the test cases for the current step and all previous steps. For each step, consider the following systems of linear equations labeled (a), (b), and (c):

$$\begin{aligned} 3x_1 + 5x_4 &= 8 \\ 2x_2 - 7x_3 &= 9 \\ 5x_1 - 3x_2 - 10x_4 &= -8 \\ 6x_2 - 5x_4 &= 1 \end{aligned}$$

(a)

$$\begin{aligned} -3x_1 + 16x_3 &= 6 \\ 3x_1 + 2x_2 &= -3 \\ -x_1 - x_2 - x_3 &= 10 \\ x_2 + 2x_3 + 10x_4 &= 7 \\ 2x_5 + x_6 &= 0 \\ 8x_4 + 3x_6 &= -1 \end{aligned}$$

(b)

$$\begin{aligned} 2x_1 - 4x_4 + 5x_6 &= 32 \\ 3x_1 + 4x_3 + 2x_5 + x_7 + 8x_8 &= 31 \\ x_2 + 2x_4 + 3x_7 &= -7 \\ x_1 + x_4 + 2x_5 + 2x_8 &= 10 \\ -x_1 - 2x_2 + 5x_8 &= 8 \\ x_2 + 2x_4 + 3x_6 &= 8 \\ x_3 + 2x_5 + x_7 + x_8 &= 8 \\ x_1 - 3x_5 + x_8 &= -5 \end{aligned}$$

(c)

Step 1: Construct sparse matrices from (a), (b), and (c) using initialize(n, S). These inputs can be hardcoded into your program. Print (a), (b), and (c) as matrices to standard output.

Step 2: For Step 2 all operations are cumulative. After each of the following operations print the resulting matrices. For all three matrices: (1) Add the last row multiplied by 2 to the first row. (2) Add the third row multiplied by -3 to the second row. (3) Subtract the first row from the second row. (4) Subtract the last row multiplied by 3 from the second to last row. (5) Swap the first and last rows. (6) Swap the second and second to last rows. (7) Delete all of the nodes in column 1. (8) Delete all but the last two nodes in row 2 (columns 1 … n-1). Remember, column 1 was already deleted in (7).

Step 3: For Step 3's test cases you will first have to reinitialize the sparse matrices for (a), (b), and (c). Triangulate each matrix and print the result. Use the triangulated matrix to solve each system of linear equations. Print the results. Verify that your solutions are correct using IsSolution(x).

--- START THIS PROJECT TODAY! ---