Sparse Matrices with Linked Lists

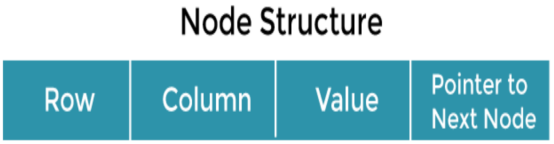
Isaiah Peralta, Edniel Campos, Damien Johnson, Cano Morales

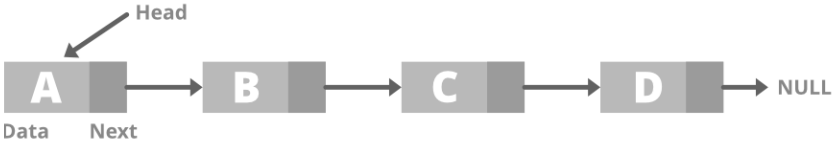
I. Introduction

The topic for our group project was to implement Sparse Matrices with Linked Lists. Initially, in our brainstorming process we decided the best way to show the implementation of sparse matrices with Linked Lists was by creating a visual representation or a visualizer to better show and explain the process of how Sparse Matrices and Linked Lists work individually and together. In doing so, we concluded with creating a simple console command sparse matrix calculator by implementing addition and multiplication, subtraction, and transpose. The visualization ends up just being through the console.

	0	1	2	3
0	0	0	1	0
1	3	0	0	0
2	0	4	5	0
3	0	6	0	0

For context, a sparse matrix is a two-dimensional grid of elements but the majority of the elements in the matrix are composed of zero values. A Linked List is a collection of nodes that holds different data types, but has to include a pointer to store the memory address of the next node. When using a Linked List to represent Sparse Matrices, it's important to understand the node structure. The first integer value stored by the nodes represents the row index while the second integer value represents the column index. The third integer value represents the value at that row and column index in the matrix. A node can usually just have a pointer to the next node in the matrix. To adjust the node data members so that we can better apply the methods to multiply. It was a lot harder to implement an algorithm with a node that points to just the next node. The last two data types for our nodes will both be of data type pointer. That way, the fourth data element will point to the next node that is in the next row but in the same column while the fifth will point to the





next node in the next column but in the same row. If a node is in the last column or row, the pointer values will be NULL respectively. Therefore, our implementation of a Sparse Matrix becomes represented through lists of linked lists. Where the matrix class holds two arrays of node pointers. One holding the first node in the respective row index. And the other holding the first node in the respective column index.

II. Methods/Implementation

Our matrix is represented as a Lists of linked lists (LIL). Figure 1 below is the best picture of a representation of how a matrix is represented as LIL. A matrix is represented through the matrix class, which has four members. Two of them are holding the dimensions as integers. There are also two arrays of node pointers as data members of the matrix class, with one holding the first node in the respective row index and the array holding the first node in the respective column index. The node in the array points to the next node in the row and the next node in its column creating the implementation of a linked list inside each array. Each node of the matrix has a non-zero value and zeros are not saved as nodes when the text file from the matrix is read.

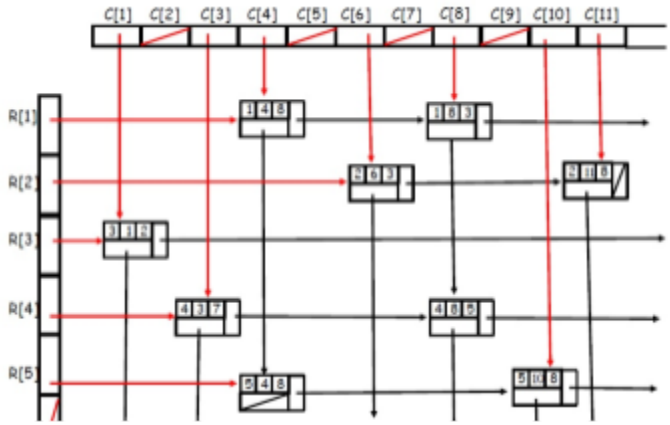


Figure 1
Source:
https://stackoverflow.com/questions/28000250/creating-a-sparse-matrix-using-linked-lists-in-c

Our matrix class has these methods implemented: create, read_file, add_node, print zeros, write, add, subtract, multiply, and transpose. These will be described in greater detail shortly. The main operations on matrices implemented were Addition, Subtraction, Multiplication, and Transpose.

To quickly summarize what these operations do, matrix addition adds the corresponding i and j values of two matrices which should output a matrix of the same size that contains the resulting addition of the corresponding i and j values of both matrices. Subtraction is similar in this since. However, matrix multiplication is different so to start off; a matrix with the dimensions i x j and the other one being n x k. The resultant matrix's dimension will i * k. Therefore j and n must be equal to each other when multiplying two matrices. The process of matrix multiplication ends up being that for the entry in the ith row and the jth column of the resultant matrix, multiply each value in the ith row of the first matrix by the corresponding value in the jth column of the second matrix and adding the results. [1] Finally, to transpose a matrix we simply flip the entries of the matrix row and column position, so that the i of that entry now becomes the column position and vice versa. For example, the value 3 is in the 3rd row and 2nd column. The resultant matrix has that same entry value but in the 2nd row and 3rd column.

Add_node

The add node method of the matrix class takes the position of the row, and column and that a node will hold. It creates a new instance of a node and sets the corresponding values to the data members of the node. It then loops through the corresponding index of the column array to get the first node in the linked lists. It saves the node at the end of the linked list. It proceeds to do the same for the row index of the rows array.

Read_file

---

1 https://www.varsitytutors.com/hotmath/hotmath_help/topics/matrix-multiplication#:~:text=The%20definition%20of%20matrix%20multiplication,AB%E2%89%A0BA%20.

The read_file method takes in a string parameter of the file name that the function reads from. From there, the function traverses through the values in the text file and the non zero values are saved to the corresponding matrix class by its respective position using integers named i and j and incrementing i when a new line is read and resetting j after every line and incrementing j for every token read in the string stream line.

Write

The write method uses two nested for loops to write out all the values of the matrix. It sets a current row node to start off the node pointer from the first node in that row. If the pointer is not a null value it prints any zeros using an additional **printzeros** method that prints zeros based on the difference of two low and high integers. Then prints the corresponding node value and points to the next value in the node. But if the pointer is already null it just prints all the zeros based on the column size of the matrix and breaks the j loop to iterate to the next row by going to the ith array that holds the linked lists by row.

Addition/Subtraction

To comply with the rules of matrix addition. Our member function checks to see that both matrices are of the same dimensions, otherwise it will not even add the two matrices and if the code was changed to try to do it, it would break our algorithm anyways. Matrix A will be the matrix calling the function and Matrix B will be passed in as a parameter by value, while an empty Matrix C with zero nodes will also be passed as parameter by reference. Our function has a nested loop with the first going up to the number of rows and the nested loop going up to the number of cols. It then loops through Matrix A's first node that is at the ith index of its row array. If it finds a node in that linked list that has the same col value at current j value then it adds that node value to a hold integer. Then proceeds to do the same with Matrix B's arrays of linked lists.

In the end we can use Matrix C's add node member function to add a node to C with the parameters being passed with i and j from the nested for loops and the hold value. The Same works for subtraction very similarly but with subtraction and like two changes of code. It just sets temp to the value if it finds the corresponding node in the linked list for Matrix A. Then just subtracts the node's values to temp it when it traverses through Matrix B.

Multiply

In regards to the multiplication process of our project, the parameters are the same to the add member. There has to be a Matrix B and reference to an empty Matrix C passed. Two nested for loops are required to add_node into Matrix C with i and j being passed into the function. However, this time we have to traverse through the first node of the j index of B's column array and index i of Matrix A's row array. This is so we can multiply the values of the rows in Matrix A with its corresponding values in the same column of Matrix B.It then has a loop to accumulate the resultant value of adding the sum of the multiplication of the row values of A with the column values of B. Conditionals are used to traverse through the linked lists starting in a specific way. The conditionals are there so we can traverse through both linked lists of the A's row array and B's column array. If it finds that the column value of the current row node is greater than Matrix B's current column node's row value then it changes the current column node to the next node in the column of that node. While if it finds that it is less than the row node it is now pointing to the node next in the row. But if it finds a match then it multiplies both current nodes and have both nodes point to the next respective node next in the row or column. It does this for every iteration of i, and j so that nodes are added to Matrix C in order.

Transpose

The transpose method takes in an empty matrix named B that is passed by reference. The function then loops through the array of node pointers by the ith value leading max of the row size. If it finds the row pointer to be null then it skips because the zeros will be printed in the right position. But, it loops through the end of the linked lists by going to the next node in the row then just adds the node that exists to Matrix B but switches the column and j values does it till it finds the last node.

Our main.cpp includes the user interface implementation through the console asking the user what it wants to do to the two matrices. However, the user has to compile the program first doing “g++ main.cpp Matrix.cpp Node.cpp -o run” and to run the program through the console the input should be “./run.exe <name of file 1> <row size> <column size> <name of file 2> <row size> <column size>. The names of the files for the two matrices must be given as a console command and the size of the matrices in the respective order.

III. Conclusion

Our program efficiently maximizes the storage of matrices. It does so by not saving the non zeros which is the premise of representing a sparse matrix with a linked list. It may use a little more memory than using a singly linked list because all nodes are pointing to the node next in the row and next in the column. However, we were able to figure out an algorithm for matrix multiplication because of reading those two node pointers as data members to the node class.

Some limitations where the users must know the 2 file names that hold the matrices inside of them. Along with knowing the size of each matrix, such as the number of rows and columns. This could be implemented by having the read_file method get the name of the data files by user input and setting the row and column data members of the matrix after it computes

the dimensions after reading. We found that our transpose method would break if we tried to transpose a matrix that is not a square matrix (dimensions differ in value). This is probably because our add_node member must add nodes by row-major order so if we created another add_node that does it by column-major order this could be fixed. Also, we default the array of linked lists to size of 100 in the matrix, when we would probably be changed based on the size of the row and column of the matrix on the creation of the matrix object.

IV. Contributions

Isaiah:

Isaiah's contributions to the project were crucial, being the "lead" of the group. Isaiah managed who will be writing specific code contributions and how the sparse matrix will be represented. This helped in other members on how our matrix is being represented as a class and how a node is represented as well. He assisted everyone in developing their functions. But, mainly stuck to figuring out how matrix multiplication could work and adjusting the main.cpp for user input and creating the flow of the program in general.

Edniel:

Edniel contributed in helping Isaiah figure out the representation of the sparse matrix. As this was the foundation for our program. He was helpful by assisting Cano with the implementation of matrix addition. Moreover, he contributed to the write member function by creating a helper function to print the zeros that are not stored in the matrix. In general, planning and documentation was improved because of Edniel. On his own he figured out how we could add matrix subtraction in our program by deriving off of addition.

Damien:

I created the Creations and implication of the Create method as well as the Transpose method. Including but not limited to helping with the implications of other methods. Working on the project report and presentation slides. I created the create method functions to set a list of all null pointers to all multipliers. I created the Transpose method that included the implementation of accepting an matrix object called B that is passed in by reference. Then would loop through a linked list until the rows for the class node. Once the node is received it will create a new node using B and the add_node button will take in the current. The node's rows, columns and the actual values.

Cano:

My contribution to this project in regards to the program includes the initialization of the "add"(matrix addition) and "read_file" method in the matrix.h file where the matrix class is defined. From there I was able to implement the actual functionality of the methods in the matrix.cpp file. Besides our program, I contributed to the editing of this report as well as the powerpoint presentation. More specifically, I was able to contribute and elaborate on matrix addition in the addition/subtraction section written above as well as some of the slides in the slide show.

*V. **Sources***

"Sparse Matrix and Its Representations: Set 1 (Using Arrays and Linked Lists)." *GeeksforGeeks*, 25 Nov. 2022, https://www.geeksforgeeks.org/sparse-matrix-representation/.

"Sparse Matrix - Javatpoint." *Www.javatpoint.com*, https://www.javatpoint.com/sparse-matrix.

"Matrix Multiplication." *Matrix Multiplication*, https://www.varsitytutors.com/hotmath/hotmath_help/topics/matrix-multiplication#:~:text=The%20definition%20of%20matrix%20multiplication,AB%E2%89%A0BA%20.