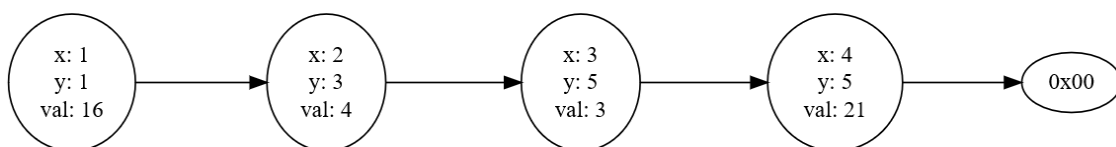


Sparse Matrices with Linked Lists

Project members: Nathan Fernandes, Michael Newton, Mathew Sturtevant, Dean Geraci

Sparse matrices are matrices where most of the elements are zero. Linked lists are a data structure where the values are linked using pointers. It contains a bunch of nodes and a pointer to the next node in the list. Sparse matrices are often represented by using a linked list because in many cases the sparse matrix is exceptionally large and would take up a lot of memory. In many cases, there can be sparse matrices with thousands of elements and just 10 non-zero elements. Representing this with a linked list uses a lot less memory because it does not include all the zero values. There are big differences between an array and a linked list. The output of a sparse matrix with a linked list is in a top-down order, with the first value being the x-coordinate, the second value being the y-coordinate, and the bottom value being the value of that coordinate, there is also a pointer to the next non-zero element.

$$A = \begin{bmatrix} 16 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 21 & 0 \end{bmatrix}$$



One example of a sparse matrix could be comparing all soccer teams in the world cup since 1930. The row and columns would contain all the teams around the world which have competed in the world cup. For example, row index 1 and column index 1 would be Germany, and column index 0 and row index 0 would be Brazil. The values of the matrix would be filled with the number of times those teams have played each other which is twice according to 11 v 11.com <https://www.11v11.com/teams/germany/tab/opposingTeams/opposition/Brazil/>. This translates to indices (1,0) and (0,1) being 2. Many of these values would be 0 however, since there are so many different teams that have played in the world cup. As a result, this type of plot would qualify as a sparse matrix. Another big application of sparse matrices is with machine learning. Machine learning uses often large sets of data, some of which are represented by sparse matrices, which the machine learning software can reference when executing a program. An example of sparse matrices in machine learning is counting the frequency of words in a text document. Some of the words that the exist will never be used, which would result in a sparse matrix.

Linked lists, arrays, and vectors are all ways you can store a matrix; however, each comes with certain advantages and disadvantages. Linked lists save a lot of memory, but most time heavy operations, such as matrix multiplication, will be far less time efficient. Because every time we want to index an element we need to iterate through every element in the linked list, that means we are automatically n times less efficient. That is to say, any ordinarily $O(n^i)$ is now automatically at least $O(n^{i+1})$. That is why using linked lists to represent any matrix is generally a poor idea, however, a very notable exception to that bad idea is if the matrix is sparse. Because

of how much data is essentially non-existent in a sparse matrix, using a linked list will save a lot of memory for your system, especially, when compared to a vector, which often uses even more memory than the number of elements it would normally hold. An array is a little bit better, but the larger a matrix is the more gaps in data there will be. Given this, bundled with the fact that most sparse matrix use cases require very large matrices, a linked list is the perfect way to run your computations.

Linked lists are often used with stacks and queues. With stacks and queues, you do not need to access elements in the middle of the list so that eliminates one of the main disadvantages of a linked list with the extra complexity needed to do that versus an array. Removing an element from the front as in a queue for example, does not shift the other elements because we just need to adjust where a few nodes point. This is unlike what would happen in an array or vector, both of which would require modifying the values of every element after the one we removed. This, of course, saves time when reordering the indexes of your elements, especially if the queue sizes were large.

Our SparseMatrix data structure contains a head pointer which points to our SparseNode structure. The SparseNode functions as a container for the non-zero elements in our matrix. It stores the x and y values of the corresponding element as well as what value is stored in that location and a pointer to the next SparseNode. The last node in the matrix will store a pointer to 0x00, the null pointer.

One of our functions implements matrix multiplication. Matrix multiplication is a process between a M by N matrix and an N by P matrix. If the columns of matrix A do not equal the rows of matrix B, matrix multiplication cannot occur. The resulting matrix has the dimensions M by P, also known as the rows of A by the columns of B. To define each element at its new index, you need to add up the products of each element A_{iy} by B_{xi} in the matrices, where “i” iterates P times. <https://www.desmos.com/matrix> was the main method used to confirm the results of multiplying smaller matrices.

"Dot Product"

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & \end{bmatrix}$$

<https://www.mathsisfun.com/algebra/matrix-multiplying.html>

Transposing a matrix involves switching the row and column values of all elements in a sparse matrix. The elements of row M become the elements of column M in the transposed matrix, and column N becomes row N of the transposed matrix. A transposed matrix can be denoted by $\langle \text{matrix_name} \rangle^T$, e.g., A^T . For sparse matrices

Transpose of a Matrix



$$A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}_{2 \times 3} \quad A^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}_{3 \times 2}$$

<https://www.cuemath.com/algebra/transpose-of-a-matrix/>

The matrix addition function adds together two matrices based on the rules of matrix addition and produces a single matrix output. Both matrices must have the same number of rows and columns for addition to be a possibility. If this is true, add both elements at the same coordinates. If matrix A has value 3 at $x = 1, y = 1$, and matrix B has 7 at $x = 1, y = 1$, the output matrix C would have the value 10 at $x = 1, y = 1$. This is true for all values in the matrix, including the addition between one zero element and one non-zero element. If matrix A has value 0 at $x = 3, y = 2$, and matrix B has value 5 at $x = 3, y = 2$, the output matrix will have value 5 at value $x = 3, y = 2$.

$$\begin{bmatrix} 4 & 8 \\ 3 & 7 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 4 + 1 & 8 + 0 \\ 3 + 5 & 7 + 2 \end{bmatrix}$$

<https://www.khanacademy.org/math/prec calculus/x9e81a4f98389efdf:matrices/x9e81a4f98389efdf:adding-and-subtracting-matrices/a/adding-and-subtracting-matrices>

Scalar multiplication takes a constant as a parameter and multiplies each element in the matrix by that constant. This is made more efficient by linked lists since having the non-zero

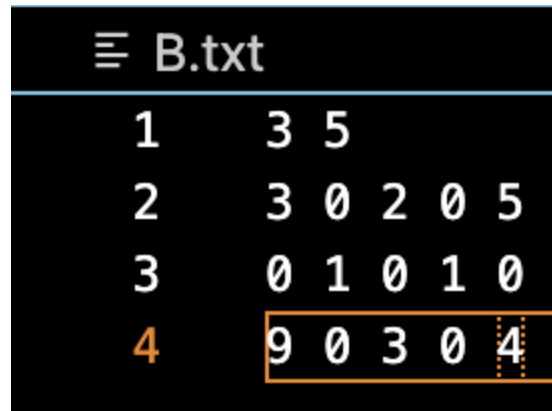
values stored, which are the only values that will be changed because of the scalar, uses less time to create the resulting matrix since all multiplication by zero is removed.

$$2 \cdot \begin{bmatrix} 10 & 6 \\ 4 & 3 \end{bmatrix} = \begin{bmatrix} 2 \cdot 10 & 2 \cdot 6 \\ 2 \cdot 4 & 2 \cdot 3 \end{bmatrix}$$

<https://www.khanacademy.org/math/prec calculus/x9e81a4f98389efdf:matrices/x9e81a4f98389efdf:multiplying-matrices-by-scalars/a/multiplying-matrices-by-scalars>

The Append node function is where the nodes are added to the end of the linked list. It starts off with a “temp” pointer that points to the head or first value of the linked list. There is an if statement that checks if the linked list is empty to add the first node. Then a new memory space is created as part of the structure “SparseNode”. Then it creates new values for the x and y coordinates and the value at that coordinate.

For our input, .txt files are used. For the .txt file to be read correctly based on the program, the first line must contain the dimensions (M and N) of the matrix. The actual matrix is started on the second line, and the read_file() function can correctly initialize the matrix if the first line’s dimensions match the actual dimensions of the matrix in the .txt file.



≡ B.txt				
1	3	5	0	0
2	3	0	2	0
3	0	1	0	1
4	9	0	3	0

Example of .txt file tested during debugging

In our group we all worked together to solve every problem we had. However the actual programming was split up. Dean worked largely on the main file, as well as, the functions for matrix transposition and scalar multiplication. Nathan worked on a lot of the core functions that we all relied on, including: One of the constructors, `append_node()` and the header file. Mike worked on matrix multiplication and one of the constructors, as well as all the operators. Matt worked on the matrix addition and file manipulation (read and save) functions. We collectively worked on the report and the PowerPoint at the same time and all brought images and ideas to the table.