**Adam Wexler**

**COP3530**

**Data Structures/Algorithms**

**Code Organization:**

**Documentation: sparseMatrix.cpp**

***Significance of Variables/Private variables:***

• Struct Node—This structures that we used in order to create a sparseMatrix class had a few elements:

* + - * 1. Node \*next—This next property is a pointer that is created to link the list together. This property is used to chain one node to another node and so on.
        2. T data—Variables that is used to store the item/elements in the matrix. This value is a template value because the user is required to create a boolean LinkedList, when performing the mask function.
        3. int col—Variable used to store the column of the element with respect to the Matrix structure.

• Node \*\*matrixRows—Double Pointed variables is created to initialize the LinkedList. This structures’ size and elements are not filled in until later during the read function, but this allows the variable to be global and can be used by any of the functions created.

• int numberOfTerms—This constant is created to find out how many times the user wants to cycle through the each array index (which is the row). This value is used as a counter for most situations in order to tell the program to stop performing an execution.

• int columns, int rows—This constant is created to store the value of the overall size of the matrix. The total size of the matrix will be of size (row x columns).

***Significance of Methods/Public Attributes:***

***1. sparseMatrix();***

This is the constructor class, which essentially lays the ground work for the structure of the SparseMatrix. This constructor is very small compared to most constructors, because the groundwork that we have to lay out is very minimal. In this method, the global variables are initialized and set to zero (0). The variables that are affected are: numberOfTerms, rows, and columns. The size of the Matrix and the terms that will be added will be specified in the read function. As a result, the time complexity of this function in the worst case and best case is both O(1), which is constant speed because you are just accessing random points in memory and setting those values.

***2. ~sparseMatrix();***

This is the destructor class, which essentially kills the class and allowed for the program to clean the SparseMatrix after use. This is done by delete the matrixRow and freeing all the node variables that are in the array. The values that are specifically affected are: delete [] matrixRows, numberOfTerms is set to zero (0), row is set to zero (0) and columns is set to zero (0). Since the matrix is essentially deleted because all the values that make up the structure are deleted Usually the time complexity of the destructor method can vary, since it depends on the number of for, while loops, or the total number of steps, which adds up all the loops in the function. However, this destructor runs in constant time complexity, which is O(1).

***3. void read();***

This is the read function of the SparseMatrix Class, which essentially allows the tester, user, or developer to add template items to an array that is filled with LinkedList. To begin the method, we will initialize the values that will be needed later on, such as the movingPoint Node and the count integer value. After the variable initialization is over, we will begin to ask the user, tester, or developer, a few inputs. These inputs (over the course of the method and not all at once) are the number of rows and columns that the matrix will have, the number of terms that will be added to each index of the matrixRow Array, and then we will ask the user to add elements and the specific column that the user wants to store. When the user provides the information, this structure will use that given information to find the best way at storing the data. The user starts by providing the row and column size of the Matrix. Shortly after, the program scans in the row variable, which is then used to initialize the Array filled with Nodes. This method checks whether the user has provided correct information/inputs. For example, this program catches out of range and negative inputs. The general time complexity of the program in Big-Oh Notation is:

Best Case: O(1) —No terms are added to the Matrix

Average Case: O((n^2)/2) — Half of the index is filled with values and half of the rows are filled

Worst Case: O(n^2) — Entire row is filled with all the columns are filled

***4. void print();***

This is the print function for the SparseMatrix Class, which essentially allows the tester, user, or developer to print all of the template items that are stored in the matrix. This function begins by printing out the total size of the matrix, which includes the row and column size. After this is printed to the screen, the program begins to traverse through the Nodes at each index, so that their values can be printed to the screen. First, the program checks whether the index has a null value. If this statement is the case, then the program will print an empty string and break out of the program. If this statement is not the case, then the program will check the index value to see the values that it holds. If more than one item is in that index, then the movingPoint Node will traverse through the “LinkedList” and print the values as the movingPoint pointer is moving to the last Node. The General Time Complexity of this Program’s print function is:

Best Case: O(1) — No elements to print

Average Case: O((n^2)/2) —Half of the row has to be printed and half of the columns has to be printed

Worst Case: O(n^2) — All of the matrix values are filled and have to be printed