# Homework 2 – Operations on Sparse Matrices

## Overview

Matrix operations are computed frequently in machine learning, data science, and computer vision. This​ ​assignment​ ​involves​ ​using​ ​linked​ ​lists​ ​to​ ​represent​ ​sparse​ ​matrices​ ​and​ ​compute​ ​matrix operations.​ ​Your​ ​program​ ​will​ ​read​ ​two​ ​sparse​ ​matrices​ ​from​ ​files​ ​(see examples​ ​below),​ ​print​ ​both​ ​matrices,​ ​compute​ ​and​ ​print​ ​the​ ​transpose​ ​of​ ​both​ ​matrices,​ ​and compute​ ​and​ ​print​ ​the​ ​matrix​ ​product​ ​of​ ​the​ ​two​ ​matrices.

### Sparse Matrix

Let’s​ ​define​ ​a​ ​sparse​ ​matrix​ ​as​ ​a​ ​matrix​ ​where​ ​the​ ​majority​ ​of​ ​elements​ ​are​ ​zeros.

**Ex:**

| -1 | 0 | 0 | 0 | 0 |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 4 |
| 0 | 15 | 31 | 0 | 0 |
| 0 | 0 | 6 | 0 | 0 |

Consider​ ​storing​ ​a​ ​100,000​ ​by​ ​100,000​ ​matrix​ ​in​ ​an​ ​array​ ​representation.​ ​That​ ​would​ ​require​ ​storing 10​ ​billion​ ​values,​ ​most​ ​of​ ​which​ ​are​ ​negligible;​ ​a​ ​O(n2)​ ​memory​ ​complexity.

For​ ​reading​ ​the​ ​sparse​ ​matrix​ ​representation​ ​from​ ​file:

* the​ ​first​ ​line​ ​of​ ​the​ ​file​ ​is​ ​the​ ​number​ ​of​ ​rows
* the​ ​second​ ​line​ ​is​ ​the​ ​number​ ​of​ ​columns
* the​ ​remaining​ ​lines​ ​are​ ​<column>,<value>​ ​pairs​ ​that​ ​represent​ ​the​ ​elements​ ​in​ ​the matrix,​ ​with​ ​zero​ ​or​ ​more​ ​pairs​ ​per​ ​row
  + **Note:** row and column numbering starts at 1, not 0

### Matrix File Format

matrixA.txt:​ ​defines​ ​a​ ​4​ ​row​ ​x​ ​6​ ​column​ ​matrix

| 4  6  1,8​ ​6,60​ ​2,5  2,33​ ​4,36  5,18​ ​4,32​ ​3,31  6,98 |
| --- |

matrixB.txt​:​ ​defines​ ​a​ ​6​ ​row​ ​x​ ​6​ ​column​ ​matrix

| 6  6  5,50​ ​4,23​ ​3,3​ ​6,87  2,8​ ​1,90​ ​5,51  2,50  5,87​ ​6,100  2,42​ ​3,8​ ​4,20  6,33​ ​1,79 |
| --- |

### Matrix Transpose

The transpose of matrix **A**, denoted **AT**, can be computed by the interchange of row i with column i. That is, the element at row i and column j of **AT** is the element at row j and column i of **A**.

| **Ex:** | Let **A** = | 1 | 2 | 3 |  | then **AT** = | 1 | 4 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 4 | 5 | 6 |  |  | 2 | 5 |
|  |  |  |  |  |  |  | 3 | 6 |

### Matrix Product

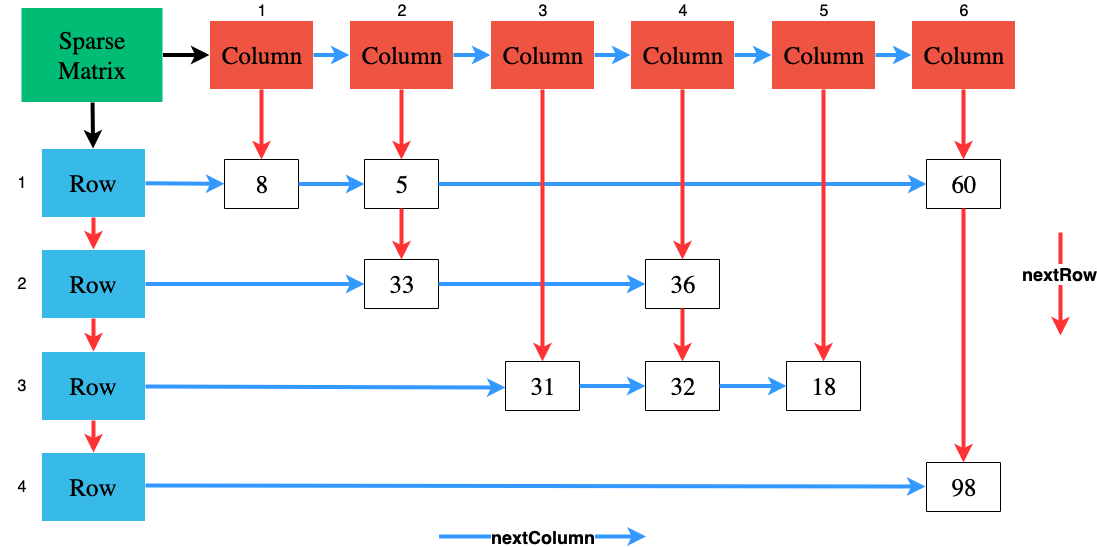
The matrix product, denoted **AB**, can be computed by evaluating the dot product of each row vector in **A** with each column vector in **B**. (**Note**: the number of columns in **A** must equal the number of rows in **B**.)

| **Ex:** | Let **A** = | A1 | A2 | A3 |  | and **B** = | B1 | B2 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | A4 | A5 | A6 |  |  | B3 | B4 |
|  |  |  |  |  |  |  | B5 | B6 |

| then **AB** = | A1\*B1 + A2\*B3 + A3\*B5 | A1\*B2 + A2\*B4 + A3\*B6 |
| --- | --- | --- |
|  | A4\*B1 + A5\*B3 + A6\*B5 | A4\*B2 + A5\*B4 + A6\*B6 |

## Approach

For​ ​this​ ​assignment,​ ​we​ ​will​ ​represent​ ​only​ ​the​ ​non-zero​ ​entries​ ​in​ ​the​ ​sparse​ ​matrix​ ​using​ ​linked lists. Each​ ​non-zero​ ​element​ ​will​ ​be​ ​represented​ ​by​ ​a​ ​node.​ Each​ ​node​ ​will​ ​not​ ​only​ ​need​ ​to​ ​keep track​ ​of​ ​its​ ​value,​ ​but​ ​also​ ​its​ ​row​ ​number​ ​and​ ​column​ ​number. As​ ​each​ ​node​ ​is​ ​in​ ​exactly​ ​one​ ​row and​ ​one​ ​column​ ​it​ ​will​ ​appear​ ​in​ ​exactly​ ​two​ ​linked​ ​lists;​ ​one​ ​linked​ ​list​ ​for​ ​the​ ​row​ ​and​ ​one​ ​linked​ ​list for​ ​the​ ​column. These​ ​row​ ​and​ ​column​ ​linked​ ​lists​ ​will​ ​be​ **​singly​ ​linked**​ ​(not​ ​doubly​ ​linked). ​A​ ​head node​ ​will​ ​be​ ​needed​ ​to maintain​ ​each​ ​column​ ​list​ ​and​ ​row​ ​list.​ Note that a head node points to the first node in the list, but is not part of the list of values. The​ ​head​ ​nodes​ ​are​ ​similarly singly​ ​linked lists;​ ​one​ ​for​ ​the​ ​rows​ ​and​ ​one​ ​for​ ​the​ ​columns. The​ ​sparse​ ​matrix​ ​will​ ​serve​ ​as​ ​the head​ ​node​ ​for​ ​both​ ​of​ ​these​ ​lists. As an example, the matrix from matrixA.txt above would be represented visually as:



## Design

The design of your program should take advantage of object-oriented programming concepts. Start with the following UML ([find skeleton code here](https://drive.google.com/open?id=121VAbNHT4PwwZbUzcEjqyqVqSAu5MxWx)):



You will need to alter the class design in the following ways:

* Appropriate constructors, accessor methods, & mutator methods should be added.
* Private methods should be added to ensure methods don’t become too long and complex.
* Constants should be added and used where appropriate.

**However, keep the class names, method names, return types and parameters the same.**

## Requirements

Below are descriptions of functional requirements that solutions must provide:

* Each​ non-zero matrix value​ ​must ​be​ ​represented​ ​by​ ​a​n instance of ValueNode.
  + No zero values should be represented by nodes
* The linked lists for rows, columns, and values must all be singly linked (not​ ​doubly)
  + You must implement your own Linked List; not use Java’s
* Each instance of ValueNode must have a reference to the next ValueNode in its row (nextColumn), and a reference to the next ValueNode in its column (nextRow)
* The insert method on MatrixRow and MatrixColumn should insert a ValueNode in **sorted** order
  + Based on the row / column of the ValueNode being inserted
* The get method on MatrixRow and MatrixColumn should return 0 if there is no ValueNode at the specified position
  + **Note** that row and column positions start at 1, not 0
* The getFirst method of MatrixRow and MatrixColumn should return the first ValueNode in its row or column
* The getNext method of MatrixRow and MatrixColumn should return the next MatrixRow or MatrixColumn in the list
* A SparseMatrix must have a reference to the head of the list of rows (firstRow), and a reference to the head of the list of columns (firstColumn)
* The constructor of SparseMatrix should create a MatrixRow for each row and a MatrixColumn for each column
* The getValue method on SparseMatrix should return 0 if there is no ValueNode at the specified row and column
* The print method on SparseMatrix should print the values of each column in each row, including any 0 values
* The transpose method on SparseMatrix should compute the transpose and return the result as a new SparseMatrix
* The product method on SparseMatrix should compute the matrix product and return the result as a new SparseMatrix
* The run method on Homework2 should perform the following:
  + Read a file named matrixA.txt to build matrix A
  + Print matrix A
  + Read a file named matrixB.txt to build matrix B
  + Print matrix B
  + Compute the transpose of matrix A and print it
  + Compute the transpose of matrix B and print it
  + Compute the product of matrix A x matrix B and print it
  + End

### Other Considerations

* The main method should have **no** program logic. It should only start the program. The main method should be placed by itself in a Main.java class.
* Do NOT make all of your methods and fields static. Limit usage of static to only constants and utility methods that do not involve program state
* Implement the program incrementally; adding code, compiling, and debugging a little bit at a time. This will help ensure your group’s program is always functional (can "do something"), even if it is not complete.
* I recommend groups follow a bottom up approach. Start by implementing and debugging the ValueNode, MatrixRow and MatrixColumn classes, before moving on to top level classes like SparseMatrix that combine the other classes for the final program.
  + Start by assuming inputs are ordered, so values can be inserted to the end. Wrap back around later and ensure they insert in sorted order.
* Break up the program’s logic into smaller pieces to keep methods from getting too long and complex.
* Use encapsulation, providing accessor and mutator methods instead of exposing public fields.
* Use good programming style when creating the program: descriptive names for classes, methods, variables, and constants; proper indentation for code blocks; etc. It is also advisable to save backup copies (e.g. gitlab.cs.wwu.edu) of the program periodically as your group work, so that you can restore to stable build if you want to experiment with something new or to avoid losing your work if something goes wrong.
* Once implemented, test the program to verify that it satisfies all of the requirements listed above, including error handling. Be sure to test using several different sets of input values.
* Verify the results of the matrix operations using online calculators (e.g. [Transpose](http://matrix.reshish.com/transpose.php), [Product](http://matrix.reshish.com/multiplication.php)).

## Submission

Team collaboration is required for project assignments. A team allows for a maximum of three students. Each team must join a “HW2” group on Canvas with each of the members (whether 2 or 3 students).

After your group has completed and thoroughly tested the program, upload a .zip file containing ONLY your group’s source code files to Canvas.