22C:21 Project 1

Due date and time: Friday October 1st 5 pm

Introduction. This is the first in a series of 3 programming projects that will build on each other. In this project you will implement a `SparseMatrix` class and write a program that tests and evaluates this class. The `SparseMatrix` class will have all of the functionality of the `Matrix` class (pages 60-63 in your textbook), but the underlying representation of the matrix will be quite different. Large matrices that occur in practice, say in civil engineering applications, are usually quite sparse, (i.e., most of their entries are zeroes). Therefore storing just the non-zero entries could provide a significant amount of savings in memory usage and possible improvement in the running time of some of the operations as well. More precisely, suppose we want to store an $m \times n$ matrix containing t non-zeroes. If t is very small as compared to $m \cdot n$, we could save a significant amount of space if the amount of space we used was $O(t)$ rather than $O(mn)$. The `SparseMatrix` class will take advantage of the sparsity of a matrix and use space that is proportional to the number of non-zeroes in the matrix.

Sparse matrix representation. Here is a detailed description of how I want you to represent an $m \times n$ matrix M so as to take advantage of its sparsity. As in the `Matrix` class, two protected variables `height` and `width` contain the values m and n respectively. In addition, there are three protected `Vector` objects called `values`, `columnPointers`, and `rowIndices` that together represent M. In the following, I will assume that the rows of M are labeled 0 through $m-1$, and the columns are labeled 0 through $n-1$. The three vectors mentioned above, play the following role in representing the matrix M.

values: This is a `Vector` whose size equals the number of non-zero entries in M. It contains the non-zero entries of M, column by column, starting with column 0 and ending with column $n-1$. For this project assume that we are only dealing with integer matrices.

columnPointers: This is a `Vector` of size n such that if `columnPointers[i]` contains the value c, then it means that entries of column i in M start at location c in the `Vector` `values`. If a column i contains only zeroes, then assume that `columnPointers[i]` is equal to -1.

rowIndices: This vector has the same size as `values`. If the entry in `rowIndices[i]` is r, then it means that the value in `values[i]` is in row r in M. Thus entries in `rowIndices` are in the range $0..m-1$.

An example will help illustrate this further. Suppose M is the matrix

$$\begin{pmatrix} 1 & -3 & 0 & -1 & 0 \\ 0 & 0 & -2 & 0 & 3 \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & -4 & 0 \\ 5 & 0 & -5 & 0 & 6 \end{pmatrix}$$

Then the `Vector values` is

1	2	5	-3	4	-2	-5	-1	-4	3	6

The `Vector columnPointers` is

0	3	5	7	9

The `Vector rowIndices` is

0	2	4	0	3	1	4	0	3	1	4

Methods in the `SparseMatrix` class. I want the `SparseMatrix` class to provide all the methods that the `Matrix` class provides, plus a few more. The methods `get`, `set`, `height`, and `width` retain the same meaning in the `SparseMatrix` class that they had in the `Matrix` class, though their implementations will be different. The class constructor and the methods `addRow` and `addCol` have slightly different meaning. The `SparseMatrix` class constructor takes integers h and w and constructs an $h \times w$ matrix of zeroes (rather than nulls). The methods `addRow` and `addCol` respectively add a row of zeroes or a column of zeroes to the matrix. The methods `removeRow` and `removeCol` need more explanation. I'll explain how the `removeRow` method works – `removeCol` is similar. In the `Matrix` class, the `removeRow` method returns a `Vector` containing the removed row. In the `SparseMatrix` class, `removeRow` returns should return just the non-zero entries in the row because the row size may be huge as compared to the number of non-zero entries in the row. To do this, `removeRow` returns a `Matrix`, let us call this X, with two rows. Row 0 in X is just the non-zero values in order from the removed row and the Row 1 of X is the column indices of these non-zero entries. For example, if we asked `removeRow` to remove row 0 from the matrix M shown earlier, it would return

1	-3	-1
0	1	3

Similarly, if we asked `removeCol` to remove column 1 from the matrix M, it would return a `Matrix` with two columns:

1	0
2	2
5	4

In addition to the methods described above, I would like the `SparseMatrix` class to provide the following methods.

(a) A second constructor

```
public SparseMatrix(Matrix x)
```

that takes a `Matrix` x and constructs an equivalent `SparseMatrix` representation of x.

(b) Two functions, `getRow` and `getCol`:

```
public Matrix getRow(int r)
public Matrix getCol(int c)
```

where `getRow` returns the row r without removing it from the matrix. It returns the row, the same way that `removeRow` does – as a matrix with two rows. The method `getCol` returns the column c without removing it from the matrix. It returns the column, the same way that `removeCol` does – as a matrix with two columns.

Testing and Evaluating the `SparseMatrix` class. To test and evaluate your implementation of the `SparseMatrix` class, I want you to perform matrix multiplications on matrices represented as `SparseMatrix` objects and separately on matrices represented as `Matrix` objects. Specifically, for an integer n and a real number p, $0 < p < 1$ construct two $n \times n$ random matrices A and B, in which each entry is non-zero with probability p. Then compute the product of A and B. Do this once using the `SparseMatrix` representation for A and B and then again using the `Matrix` representation for A and B. For each representation, measure the time taken for (i) matrix construction and (ii) matrix multiplication. To get more accurate measurements, for fixed n and p, repeat the experiments 10 times and record mean times. Perform the experiment for $n = 1000, 2000, 3000, \ldots, 10,000$ and $p = 0.05$. Plot your times as a function of n. You should get 4 plots, since the matrices have two possible representations, and there are two tasks we want timed.

Write a few paragraphs (at most one page) explaining your timing data and plots. Address the following issues:

1. In which representation is it faster to construct the random matrices? In which representation is it faster to do matrix multiplication?

2. What can you say about the running time of matrix multiplication, as a function of n, using the two different representations? Asymptotically, do the functions seem to grow at the same rate or not? Why?

3. Set $p = \frac{10^4}{n^2}$ for each choice of n. Again, generate pairs of random matrices for $n = 1000, 2000, 3000, \ldots, 10,000$, but with this new choice of p. What can you now say about the running time of matrix multiplication, as a function of n, using the two different representations? Asymptotically, do the functions seem to grow at the same rate or not? Why?

What to submit? You should submit two files with the names: `SparseMatrix.java` and `MatrixMultiplication.java` respectively containing the class `SparseMatrix` and the class `MatrixMultiplication`. The class `MatrixMultiplication` should contain functions for generating random matrices and functions for multiplying matrices. These files should be submitted electronically – more information on how to do this will be made available on the course web-page soon. You should submit a hardcopy of tour timing data/plots and the one-page writeup that goes with it.
