CSCI 310 – Data Structures – Spring 2020
Sparse Matrix

Matrix Facts

An $m \times n$ *matarix* A is a rectangular array of mn real numbers arranged in m horizontal rows and n vertical columns.[1]

If A is an $m \times n$ matrix, the element at row i, column j, is denoted a_{ij}, for $1 \leq i \leq m$, $1 \leq j \leq n$.

If A and B are both $m \times n$ matrices, then the sum $A + B$ is an $m \times n$ matrix C where $c_{ij} = a_{ij} + b_{ij}$, for $1 \leq i \leq m$, $1 \leq j \leq n$.

If A is an $m \times n$ matrix and r is a real number, then the *scalar multiple* of A by r, written as rA, is the $m \times n$ matrix C, where $c_{ij} = r\ a_{ij}$, for $1 \leq i \leq m$, $1 \leq j \leq n$.

If A and B are $m \times n$ matrices, $A + (-1)B$ is written as $A - B$ and is called the *difference between A and B*.

If A is an $m \times n$ matrix, then the *transpose* of A, written as A^T, is the $n \times m$ matrix with $a^T_{ij} = a_{ji}$. In other words, A^T is obtained from A by interchanging the rows and columns of A.

An $n \times 1$ matrix is also called an *n-vector*.

If A and B are both n-vectors, the *dot product*, or *inner product*, $A \cdot B$, is the real number defined as

$$A \cdot B = a_{11}b_{11} + a_{21}b_{21} + \ldots + a_{n1}b_{n1} = \sum_{i=1}^{n} a_{i1}b_{i1}$$

If A is an $m \times n$ matrix and B is a $n \times p$ matrix, then the *product* of A and B, denoted AB is the $m \times p$ matrix C defined by

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj} \quad \text{for } 1 \leq i \leq m,\ 1 \leq j \leq p.$$

Two $m \times n$ matrices A and B are *equal* if, and only if, $a_{ij} = b_{ij}$, for $1 \leq i \leq m$, $1 \leq j \leq n$.

An $m \times n$ matrix is called a *square* matrix if $m = n$.

An $n \times n$ square matrix is call a *diagonal matrix* if $a_{ij} = 0$ for $i \neq j$. In other words, terms *off* the main diagonal are all zero.

A *scalar* matrix is a diagonal matrix whose diagonal elements are equal.

An *identity* matrix is a scalar matrix whose diagonal elements are all equal to one.

A matrix A is called *symmetric* if $A^T = A$.

A matrix A is called *skew symmetric* if $A^T = -A$.

[1] All definitions are from *Elementary Linear Algebra with Applications*, 9^{th} edition, by Bernard Kolman and David R. Hill.

Caution The Matrix Facts section above describes matrices in standard mathematical terms. Specifically, the row and column indices start at 1, rather than at 0.

Sparse Matrix Implementation

Introduction We will implement a `SparseMatrix` class and write a program that uses it. 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 provides a significant amount of savings in memory usage. More precisely, suppose we want to store an $m \times n$ matrix containing t 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.

Representing the Matrix (Instance Variables) Here is a detailed description of how we will represent an $m \times n$ matrix A. The class maintains two instance variables, `numRows` and `numCols`, that contain the values m and n respectively. In addition, there are two jagged lists called `indices` and `values`. Both of these lists have m rows (the same number of rows as the matrix A). These lists are used as follows:

indices If row i of the matrix no non-zero values, then `indices[i]` is empty. But if row i of the matrix does contain non-zero values, then the length of `indices[i]` is equal to the number of non-zero values in A. The values stored in `indices[i]` are the column indices where those non-zero values are located.

values As mentioned above, this list has the same shape as the `indices` list. Once again, if row i of the matrix no non-zero values, then `values[i]` is empty. But if row i of the matrix does contain non-zero values, the length of `values[i]` is equal to the number of non-zero values in A. The values stored in `values[i]` are the non-zero values from the matrix.

Below is an example to help illustrate how this works. Suppose A is the 7×11 matrix filled with zeros except for the following:

$$\begin{aligned}
a_{02} &= 7 \\
a_{0\ 10} &= 9 \\
a_{33} &= 6 \\
a_{46} &= 3 \\
a_{49} &= 8 \\
a_{53} &= 2 \\
a_{65} &= 4
\end{aligned}$$

Matrix A is show below, with zeros replaced dashes to make it easier to read.

$$A = \begin{array}{c} \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \end{array}
\begin{array}{c}
\begin{array}{ccccccccccc} \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} & \mathbf{9} & \mathbf{10} \end{array} \\
\begin{pmatrix}
- & - & 7 & - & - & - & - & - & - & - & 9 \\
- & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - \\
- & - & - & 6 & - & - & - & - & - & - & - \\
- & - & - & - & - & - & 3 & - & - & 8 & - \\
- & - & - & 2 & - & - & - & - & - & - & - \\
- & - & - & - & - & 4 & - & - & - & - & -
\end{pmatrix}
\end{array}$$

Then the list `indices` is

	0	1
0	2	10
1		
2		
3	3	
4	6	9
5	3	
6	5	

Then the list `values` is

	0	1
0	7	9
1		
2		
3	6	
4	3	8
5	2	
6	4	

Constructors The `SparseMatrix` class includes the following constructors:

- A default (no-argument) constructor that creates a 1×1 matrix with the single matrix element initialized to zero.
- A constructor that takes a single integer n and creates an $n \times n$ matrix initialized to all zeros.
- A constructor that takes two integers m and n and creates an $m \times n$ matrix initialized to all zeros.
- A constructor that takes a 1-dimensional array of integers and creates a corresponding vector.
- A constructor that takes a 2-dimensional rectangular array of integers and creates a corresponding matrix.
- A constructor that takes a `SparseMatrix` object x and makes an equivalent `SparseMatrix` representation of x.

Accessors The `SparseMatrix` class includes the following accessor methods:

- `getMax` that returns the maximum matrix element.
- `getMin` that returns the minimum matrix element.
- `getM` that returns the number of rows in the matrix.
- `getN` that returns the number of columns in the matrix.
- `get` that takes two integer parameters i and j and returns the matrix element at row i and column j.
- `toString` that returns a string with the dimensions of the matrix and the number of non-zero entries.
- `toString1` that returns a *formatted* string showing the matrix in tabular form. The width of the table columns should all be the same and should be set according to the largest number of characters required to show any element in the matrix. Zeros should be shown as dashes. This method would normally not be called on very large matrices.

Mutators The `SparseMatrix` class includes the following mutator methods:

- `set` that takes three parameters, i, j, and $value$, and sets the matrix element at row i column j to $value$.

Predicate Methods The `SparseMatrix` class includes the following predicate methods:

- `isVector` returns true if the matrix is a vector, returns false otherwise.
- `isSquare` returns true if the matrix is a square matrix, returns false otherwise.
- `isDiagonal` returns true if the matrix is a diagonal matrix, returns false otherwise.
- `isScalar` returns true if the matrix is a scalar matrix, returns false otherwise.
- `isIdentity` returns true if the matrix is an identity matrix, returns false otherwise.
- `isSymmetric` returns true if the matrix is symmetric, returns false otherwise.
- `isSkewSymmetric` returns true if the matrix is skew symmetric, returns false otherwise.

Boolean Operations Only one Boolean operation is required for the `Matrix` class:

- `equals` takes another matrix as a parameter and tests to see if the two matrices are equal.
  The method call `A.equals(B)` returns true if $A = B$ and returns false otherwise.

Matrix Operations The `SparseMatrix` class includes the following matrix operations:

- `add` Takes another matrix and performs matrix addition.
  The method call `A.add(B)` returns the matrix $A + B$.
- `scalarMultiply` Takes an integer r and performs scalar multiplication.
  The method call `A.scalarMultiply(r)` returns the matrix rA.
- `difference` Takes another matrix and performs the difference operation.
  The method call `A.difference(B)` returns the matrix $A - B$.
- `transpose` Returns the transpose of the matrix.
- `dotProduct` Takes another matrix and performs the dot product operation.
  The method call `A.dotProduct(B)` returns the integer $A \cdot B$.
- `multiply` Takes another matrix and performs matrix multiplication.
  The method call `A.multiply(B)` returns the matrix AB.
- `subMatrix` Takes two integer parameters i and j and returns the sub-matrix with row i and column j removed.

Static Methods The `SparseMatrix` class includes one static method:

- `identity` Takes one integer parameter n and returns the $n \times n$ identity matrix.

Tester

We would want a tester program that thoroughly tests the `SparseMatrix` implementation.