Assignment 1 – Sparse Matrix

Due: Monday, April 1st, 2019 by 11:55PM

In this assignment, you will use dynamic array class from the lecture note to implement a sparse matrix data structure. You're welcome to read more about sparse matrix from other sources but be aware that these descriptions may be slightly different with the requirement in this assignment. Therefore, please read the entire handout carefully before starting your implementation.

1. Introduction

A sparse matrix or sparse array is a matrix in which most of the elements are zero. For example, the following is an example of a sparse matrix:

```
0 0 0 0 8
0 1 2 0 0
0 0 0 0 0
0 3 0 0 9
```

Advantages of using Sparse Matrix instead of simple matrix

Storage: There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.
Computing time: Computing time can be saved by logically designing a data structure traversing only non-zero elements.

2. Implementation

Representing a sparse matrix by a 2D array leads to wastage of lots of memory as zeroes in the matrix are of no use in most of the cases. So, instead of storing zeroes with non-zero elements, we only store non-zero elements. This means storing non-zero elements with triples - (row, column, value). In this implementation, we call it `Element` with the prototype defined as below.

Sparse Matrix Representations can be done in many ways. Following are two common representations:

- Array representation
- Linked list representation

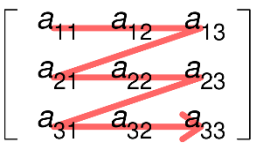

Figure 1. Row-major order for storing

In this assignment, you are required to use array representation (dynamic array) to represent the sparse matrix. In particular, you are required to use **row-major order** method for storing non-zero

elements of sparse matrix in a linear one-dimensional array. Figure 1 (above) shows an example of row-major order method. This storing method is very useful for finding non-zero elements of sparse matrix and also for other operations such as addition, point-wise multiplication.

For example, the sparse matrix (shown in the previous example) will be stored in the dynamic array in the following order:

```
(0, 4, 8)  (1, 1, 1)  (1, 2, 2)  (3, 1, 3)  (3, 4, 9)
```

If we use operator << to print the content of the sparse matrix, the result is as follow:

```
Sparse Matrix (4 x 5):
---------------------
(0,4) = 8
(1,1) = 1
(1,2) = 2
(3,1) = 3
(3,4) = 9
---------END----------
```

The implementation of the assignment is organised into one main() function and two classes: `Element` and `SparseMatrix` with four separate files, namely, `Element.h`, `SparseMatrix.h`, `SparseMatrix.cpp` and `main.cpp`. The detail of these files are as follows:

a. Node.h

This file contains the declaration and definition of `Element` data structure.

```
class Element {
public:
   unsigned int row, col;
   double value;
   Element(unsigned int r, unsigned int c, double val);
   friend ostream& operator<<(ostream& out, const Element& e);
};
```

b. SparseMatrix.h

The file contains the declaration of `SparseMatrix` data structure and the definition of several methods in this class. The prototype of this class is as following.

```
class SparseMatrix {
private:
       // data member for dynamic array
       Element* storage;
       unsigned size;
       unsigned capacity;
       unsigned int nrow, ncol; // the size of matrix is nrow x ncol

       void rangeCheck(unsigned row, unsigned col) const;
       void setCapacity(int newCapacity);
       void ensureCapacity(int minCapacity);
       void pack();
       void trim();
       void insertAt(int index, Element val);
       void add(Element val);
```

```
        void removeAt(int index);
public:
        SparseMatrix(unsigned int nr = 1, unsigned int nc = 1);
        SparseMatrix(const SparseMatrix& a);
        SparseMatrix& operator=(const SparseMatrix& a);
        ~SparseMatrix();
        void printFull();
        friend ostream& operator<<(ostream& os, const SparseMatrix& a);

        // Your tasks: complete the implementation of the below methods
        // in SparseMatrix.cpp
        void setAt(unsigned row, unsigned col, double val);
        double getAt(unsigned row, unsigned col);
        SparseMatrix* transpose();
        double trace();
        SparseMatrix* add(const SparseMatrix& b);
        SparseMatrix* multiplyPointWise(const SparseMatrix& b);

};
```

c. `SparseMatrix.cpp`

This file contains the prototypes for all the methods you need to implement in this assignment. The detail descriptions for all these methods are provided in the next section.

d. `main.cpp`

This file contains the main() function to test your implementation of the Sparse Matrix. It sequentially reads the commands from the `input.txt` to create objects of `SparseMatrix` and call methods of this class. **You can assume that the input always starts with the command c to create an object of Sparse Matrix and the parameter idx only contains two values 0 or 1**.

The detail format of `input.txt` and the meaning of these commands are as follows:

No	Command	Parameters	Method called
1	c	row (int) col (int) idx (int)	`sm[idx] = new SparseMatrix(row, col)`
2	g	row (int) col (int)	`getAt(row, col)`
3	s	row (int) col (int) val (int)	`setAt(row, col, val)`
4	t		`transpose()`
5	r		`trace()`
6	+	idx (int)	`add(*sm[idx])`
7	*	idx (int)	`multiplyPointWise(*sm[idx])`
8	p		`operator<<`
9	f		`printFull()`

The below is one example of the input file:

```
c 4 5 0
s 3 4 10
s 0 1 2.5
s 2 0 -2
g 2 4
s 1 3 -1
p
```

and the corresponding output:

```
Get (2, 4) = 0
Sparse Matrix (4 x 5):
----------------------
(0,1) = 2.5
(1,3) = -1
(2,0) = -2
(3,4) = 10
---------END----------
```

Another example of the input file:

```
c 4 5 0
s 0 4 8
s 3 4 9
s 1 2 2
s 1 1 1
s 3 1 3
p
c 4 5 1
s 3 4 10
s 0 1 2.5
s 2 0 -2
g 2 4
s 1 3 -1
p
* 0
```

and the corresponding output:

```
Sparse Matrix (4 x 5):
----------------------
(0,4) = 8
(1,1) = 1
(1,2) = 2
(3,1) = 3
(3,4) = 9
---------END----------
Get (2, 4) = 0
Sparse Matrix (4 x 5):
----------------------
(0,1) = 2.5
(1,3) = -1
(2,0) = -2
(3,4) = 10
---------END----------
Sparse Matrix (4 x 5):
----------------------
(3,4) = 90
---------END----------
```

3. Your Tasks

Your tasks in this assignment is to implement the Sparse Matrix described in the previous section. In particular, you need to complete the following methods provided in the file "SparseMatrix.cpp". The initial code provides you the skeleton to help you to complete the requested data structure.

No	Method	Description	Points
1	`double getAt(unsigned row, unsigned col)`	return the value of the element at the position (row, col) in the Sparse Matrix if this element exists; return 0 otherwise; Parameters: - row, col: indices of element in the matrix. Throws: `MatrixSubscriptOutOfBoundsException`: if the indices (row, col) is out of range (row < 0 || row >= nrow || col < 0 || col >=ncol)	15
2	`void setAt(unsigned row, unsigned col, double val)`	replaces the value of the element at the specified position (row, col) in the matrix with the new value if this element is already existed; otherwise insert a new element to the dynamic array at a suitable position based on it position (row, col) followed the **row-major order** storage. Parameters: - row, col: indices of element in the array. - val: new value. Throws: `MatrixSubscriptOutOfBoundsException`: if the indices (row, col) is out of range (row < 0 || row >= nrow || col < 0 || col >=ncol)	20
3	`SparseMatrix* transpose()`	returns a pointer to the transpose matrix of the current matrix.	10
4	`double trace();`	return the trace value of the current matrix if it is square; otherwise throw exception Throws: `TraceOfNoneSquareMatrix`: if the matrix is not square	10
5	`SparseMatrix* add(const SparseMatrix& b);`	returns a pointer to the resulted sum matrix of the current matrix and the matrix b. Parameters: - b: a sparse matrix to be added Throws: `MismatchedDimensions`: if the dimensions of the current matrix and the matrix to be added are mismatched	25
6	`SparseMatrix* multiplyPointWise(const SparseMatrix& b)`	returns a pointer to the resulted point-wise multiplication matrix of the current matrix and the matrix b. Parameters:	20

		- b: a sparse matrix to be multiplied Throws: `MismatchedDimensions`: if the dimensions of the current matrix and the matrix to be added are mismatched	

4. Evaluation – How assignments are graded?

This assignment is graded using the mechanism of autotester. In specific, there are totally 100 testcases. A testcase is considered "passed" if your output is completely matched with the expected output (solution). The number of "passed" testcases will correspond to your grade for this assignment.

5. Submission

In this assignment, you are allowed to submit ONLY TWO files `SparseMatrix.h` and `SparseMatrix.cpp` which contains all your code. Therefore, you are NOT allowed to change the code in the other files including `Element.h`, and `main.cpp`. In addition, you should make sure that no other debugging printf or cout left behind in your code.

Instruction:

- Select two files: " `SparseMatrix.h` " and "`SparseMatrix.cpp` ", compress into "src.zip". Please **DO NOT** put them into any subfolder.

- Access website http://www.cse.hcmut.edu.vn/onlinejudge/ and sign in using your university account and password.

- Submit "src.zip" to the system .

There are total 100 testcases, each one corresponds to 1 points. You can check the result for each testcase whether it is pass or failed. You can submit 20 times per day. The timeout for each testcase is 3 seconds. Your final mark is based on the final submission.

The deadline for this assignment is **Monday, April 1st, 2019 by 11:55PM**

6. References

[1] Wikipedia "Sparse Matrix" accessed 24 February 2019, Online website: <https://en.wikipedia.org/wiki/Sparse_matrix>