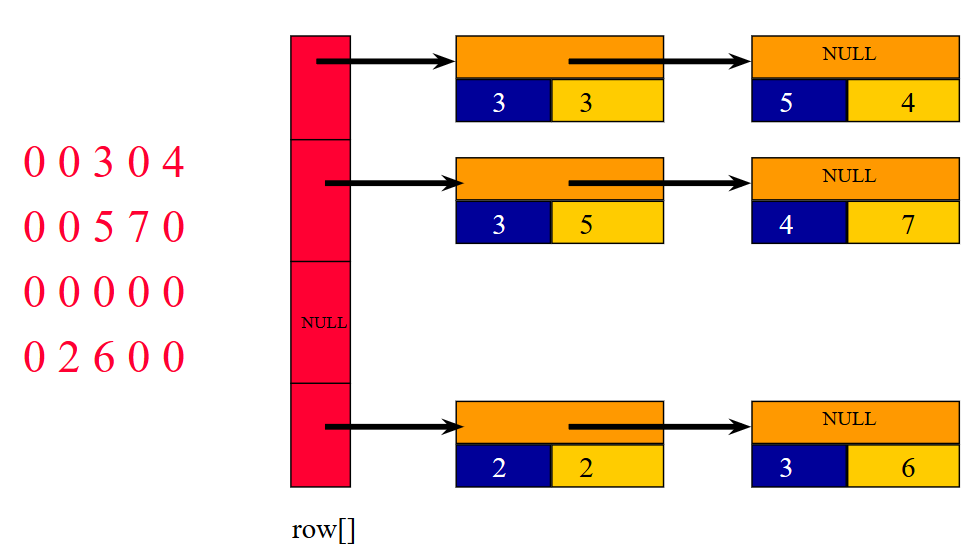
**COP 3530, Fall 2015**  
Data Structures and Algorithms  
Assignment 2: Masking Sparse Matrices  
**Due: October 12th, 2015, 11:59PM**

**Description:**  
 Sparse matrices are matrices populated primarily with zeros (lecture 11). When storing and manipulating sparse matrices on a computer, it is beneficial and often necessary to use specialized algorithms and data structures that take advantage of the sparse structure of the matrix. You are required to implement the **sparseMatrix** data structure described as follows. Your **sparseMatrix** should consist of an array of chains (see figure below). Each chain holds the non-zero elements of the corresponding row in the matrix. These non-zero elements can be of type **boolean or integer**. All of the elements of a **sparseMatrix** must be of the same type, i.e. a **sparseMatrix** cannot contain both integers and booleans. Thus, your **sparseMatrix** class **must be templated.**



For your **sparseMatrix** data structure, you must implement a constructor, destructor, print, read, and mask methods. You may also implement any helper function(s) at your own discretion.

* The **read** function receives input from standard input (**stdin**) and creates a sparse matrix by assigning values to the internal structures of the matrix.
* The **print** function prints out the elements of the matrix, in the format shown in the example below.
* The **sparseMatrix** method **mask(b,c)**, has an implicit third operand. The operand is **this** (the object invoking the method). The **mask**(b,c) method stores the result of masking t**his** and ***b*** into the **sparseMatrix** ***c***.
* Masking a matrix is defined as follows:

c(i,j) = a(i,j) if b(i,j) is true, and

c(i,j) =0 otherwise,

where **b** is a boolean sparse matrix, and **a** and **b** are integer sparse matrices.

* You need to explicitly instantiate two matrices with type **int**, and one matrix with type **boolean**. Your main matrix and result matrix will contain integers. Your mask matrix will contain booleans.
* You may assume that all matrices will have the same dimensions.
* **Note:** As you read in matrix **b**, you will read integer values from the input, but store boolean values into **b**. As you read in matrix **a**, you will read (and store) integer values.

**Sample test:**

int main()

{

sparseMatrix<int>\* a = new sparseMatrix<int>();  
 sparseMatrix<bool>\* b = new sparseMatrix<bool>();  
 sparseMatrix<int>\* c = new sparseMatrix<int>();

cout << "Reading Matrix A" << endl;  
 a->read();

cout << “Matrix A:” << endl;  
 a->print();

cout << "Reading Matrix B" << endl;  
 b->read();

cout << “Matrix B, the boolean mask matrix:” << endl;  
 b->print();

// Masking

a->mask(\*b,\*c);

cout << “Matrix C, result:” << endl;  
 c->print();

return 0;

}

**Sample input and output (user input is shown in RED):**

Reading Matrix A

Enter number of rows, columns

3 4

Enter number of terms in row 1

1

Enter element's column, and value of each term in row 1

1 111

Enter number of terms in row 2

2

Enter element's column, and value of each term in row 2

2 222 3 233

Enter number of terms in row 3

0  
  
Matrix A:

rows = 3 columns = 4

row 1[ col:1 val= 111]

row 2[ col:2 val= 222, col:3 val= 233]

row 3[]

Reading Matrix B

Enter number of rows, columns

3 4

Enter number of terms in row 1

1

Enter element's column, and value of each term in row 1

1 1

Enter number of terms in row 2

1

Enter element's column, and value of each term in row 2

3 1

Enter number of terms in row 3

0

Matrix B, the boolean mask matrix:

rows = 3 columns = 4

row 1[ col:1 val= 1]

row 2[ col:3 val= 1]

row 3[]

Matrix C, result:

rows = 3 columns = 4

row 1[ col:1 val= 111]

row 2[ col:3 val= 233]

row 3[]

We will test your submissions by following these steps (commands on thunder.cise.ufl.edu):

1. tar xvf “<LastName\_FirstName>\_UFID.tar”

2. make

3. ./sparseMatrix < our\_input

**Deliverables:**

* Your submission tarball(.tar archive file) should be named <LastName\_FirstName>\_UFID.tar and must contain the following files: sparseMatrix.cpp, Makefile, and <LastName\_FirstName>\_UFID\_report.pdf
* A PDF document containing your own test cases, test results, and any special diagnostics you utilized. This file shall be named <LastName\_FirstName>\_UFID\_Report.pdf

**PLEASE NOTE** that **ALL** submissions **MUST** compile on **thunder.cise.ufl.edu** by using your Makefile. It is highly recommended that you upload your source code to your CISE account and test it on the thunder. Secure Remote Access to CISE machines is available. Please visit this link for more information, <https://www.cise.ufl.edu/help/access/remote>. We will not debug your source code or makefile. Please refer to and make note of the submission rules and policies before submitting, <http://cise.ufl.edu/class/cop3530fa15/SubmissionRules.htm>. After submitting on Canvas, you should verify that your submission was successful by downloading it from Canvas (to a separate location), and successfully un-tarring, compiling, and running it. YOU ARE NOT DONE UNTIL YOU DO THIS.