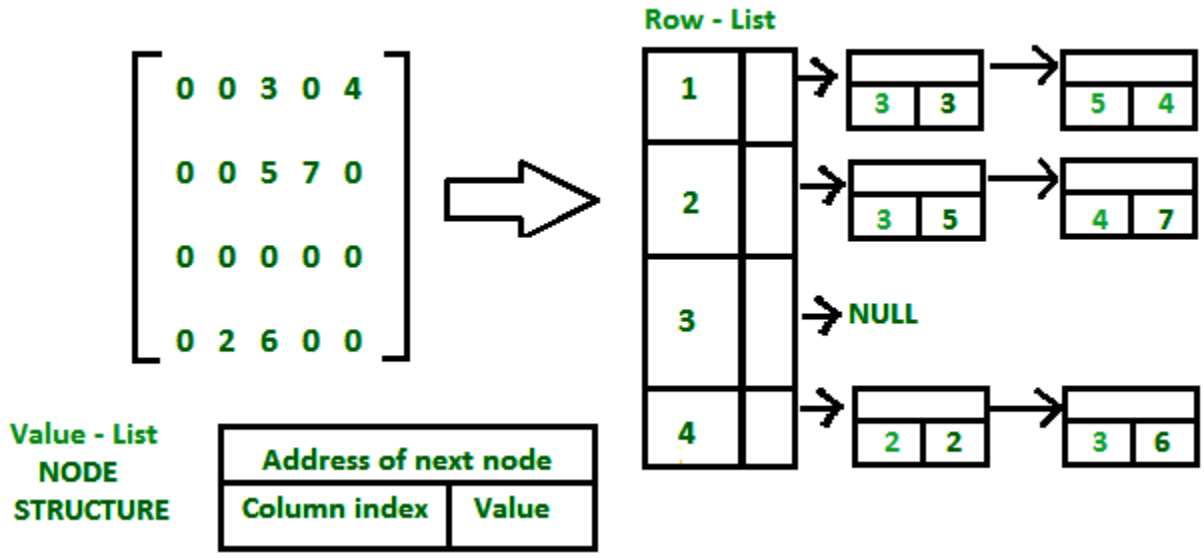
Data Structures BCS-4F
FAST-NU, Lahore, Spring 2021

Homework 3

Due: Thu, April 22, 1155 PM **100 pts**

Many computing applications produce matrices of numbers – integers – in which most of the numbers are 0. Such matrices are called Sparse Matrices.

In an NxN matrix, the total number of entries is N^2. So it requires $O(N^2)$ space. However, in a Sparse Matrix most of these entries are 0. It is possible that only O(N) entries are non-zero. So, if we stored only the non-zero entries, we could store the entire matrix in only O(N) space. Such a matrix can be stored in only O(N) space by using a scheme which only stores the non-zero elements of each row in a separate linked list (one linked list per row). The following diagram shows the scheme:



Note: both row numbers and column numbers start from 1.

The structure of each node is shown above. It contains the value of the matrix entry, its column number and a pointer to the next node in the row. Note that the size of each linked list may be different, depending upon how many non-zero entries exist in that row.

Also pay attention to the Row List which contains the head pointers of all the lists. In our implementation, we will use the STL vector to maintain these head pointers. The following code shows the basic definition of the class SparseMatrix and the struct Node. Integers M and N contain the number of rows and columns of the matrix respectively.

Note: you absolutely must **not** store the entire M x N matrix in a 2D array at any time in your program. Doing so will result in a 0 in the assignment.

```
class SparseMatrix{
    struct Node{
        int value;
        int colIndex;
        Node* nextInRow;
        Node(int v):value(v){}
    };
    int M, N;//dimensions
    vector<Node*> rowList;
};
```

You are required to add the following public methods to this class:

1. **read(string *filename*)**:
   This method should be able to read an M x N matrix from a text file. Each line in the file contains all the entries in that row (including the zeros) separated by spaces. There is one row per line. Following is how the matrix shown in the previous picture will be stored in the file:

   0 0 3 0 4
   0 0 5 7 0
   0 0 0 0 0
   0 2 6 0 0

   This file contains 4 lines. The characters in each line are separated by a space. Your function should be able to load this matrix as an object of SparseMatrix, as described above.

2. **SparseMatrix(*string filename*)**
   Constructor. Simply uses read to read the matrix from the specified file.

3. **SparseMatrix(const SparseMatrix& obj)**
   A regular copy constructor creates a deep copy.

4. **const SparseMatrix & operator = (const SparseMatrix & obj)**
   A regular assignment operator creates a deep copy.

5. **~SparseMatrix()**
   Destructor cleans up all the allocated memory.

6. **bool operator == (const SparseMatrix & obj)**

Returns **true** if two Sparse Matrices are of same dimensions and contains the exact same elements. Returns false otherwise.

7. **SparseMatrix operator + (const SparseMatrix & obj)**
   Adds two Sparse Matrices of same dimensions and returns the result in a third Sparse Matrix.

8. **SparseMatrix transpose()**
   Returns an N X M transpose of an M X N matrix.

9. **SparseMatrix operator * (const SparseMatrix & obj)**
   Multiplies two Sparse Matrices with compatible dimensions for multiplication; say, M1 X N1 and N1 X L1, and returns their M1xL1 resultant product Sparse Matrix.

10. **bool isSubMatrix(const SparseMatrix & obj)**
    Returns true if the matrix obj is completely contained in the current matrix. Returns false otherwise.

    For example, the matrix $\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$ is a sub-matrix of $\begin{bmatrix} 2 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$.

    Again, note that your matrices are stored in the space efficient manner described above and must never be reconstructed into MxN arrays at any point in your program.

*** END ***