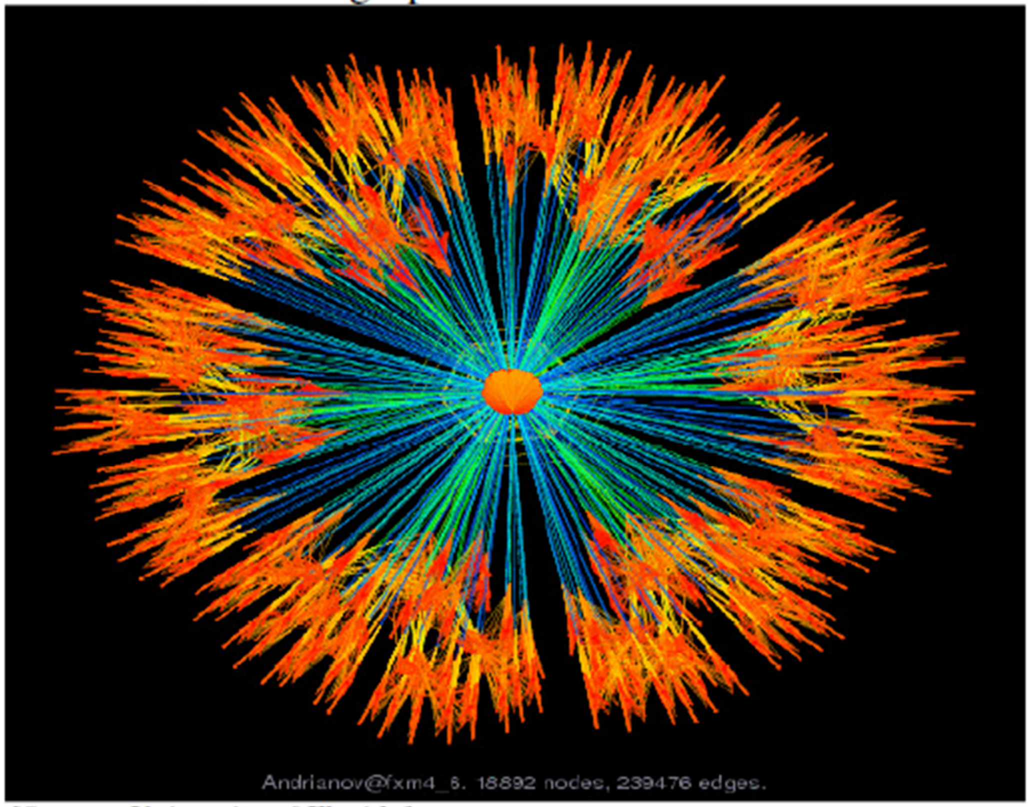
15-123 Effective Programming in C and UNIX

Lab 4 - Sparse Matrix Systems

Due Date: Thursday, February 24th 2011 by 11:59pm

Background:

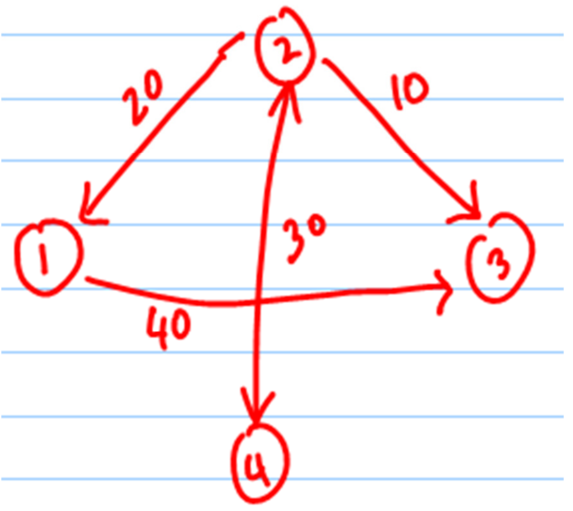
Sparse matrix is a matrix where many of its entries are undefined or zero is common in many modeling problems. Sparse matrices cover a wide spectrum of domains; include those arising from problems with underlying 2D or 3D geometry (such as structural engineering, computational fluid dynamics, model reduction, electromagnetic, semiconductor devices, thermodynamics, materials, acoustics, computer graphics/vision, robotics/kinematics, and other discretizations) and those that typically do not have such geometry (such as optimization, circuit simulation, economic and financial modeling, theoretical and quantum chemistry, chemical process simulation, mathematics and statistics, power networks, and other networks and graphs). Sparse graphs and sparse matrices are equivalent (that is, a sparse graph is represented by a sparse matrix). Here is a representation of a sparse graph with 18892 nodes and 239476 edges.



[Source: University of Florida]

We call the above graph “sparse” since many of its nodes are only connected to few other nodes. In general a graph can be represented by a matrix as follows.

Represent a graph with n nodes using a matrix of size n by n where each entry A[i][j] represents the value of the edge. For example the graph below:



Can be represented by a 4 x 4 matrix as follows

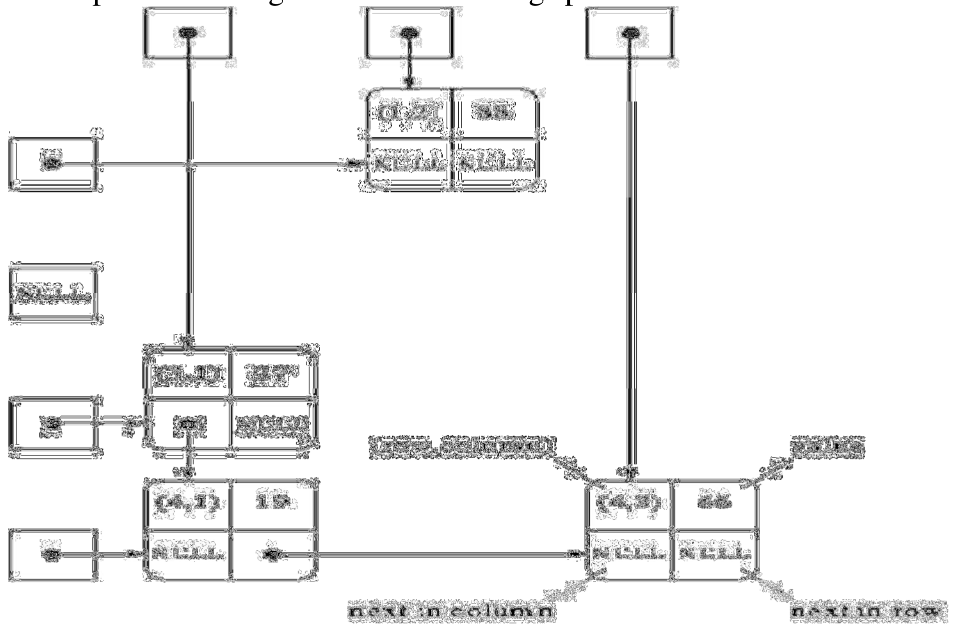
$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & x & x & 40 & x \\ 2 & 20 & x & 10 & 30 \\ 3 & x & x & x & x \\ 4 & x & 30 & x & x \end{array}$$

Although representing a matrix using a 2D array is convenient, this is often not practical for large matrices. For example, we note that if every node is connected to every other node in the above graph with 18892 nodes, you would have 356.907,664 possible edges or entries in the matrix. This is too much space to ask for since we know that the above graph has only 239476 edges that are useful. Therefore storing this graph in a typical 2D array is not practical.

We call this type of matrices sparse since they only have few meaningful values compared to all possible values, most of them don't exists. Hence storing data in a sparse matrix requires a creative way to construct a structure that can save space but still allow access to elements in an efficient way. C is an ideal language to code applications that require efficient memory management, optimization and other performance enhance techniques.

Other applications
C is also used by many programmers to solve problems that are numerically intense. One such problem is solving a large system of linear equations, a common problem in scientific computing. For example suppose we consider a system with two variables and two constraints.
ax + by = c
dx + ey = f
This system can be represented in matrix-vector form as AX = B
[a b][x] = [c]
[d e][y] = [f]
Where A = [a b]
                [d e]
and X = [x y] and B =[c f]
Finding solution (or approximate solution) X to a large linear system of the form AX = B requires some creative computational algorithms. Often A is a very large matrix which is common in many modeling problems with millions of variables and millions of constraints. These large systems lead to "sparse" systems. By a sparse system we mean that the coefficient matrix A has very few non-zero entries. For example, if a matrix A is of order n x n and n = 100,000 and no more than 50,000 entries may be non-zero (very typical of large systems), a traditional matrix storage system may require you to allocate space for 10 billion floating point numbers although only a fraction is actually used. Even if the memory is available, it seems that a better method may be utilized to store this sparse matrix. Typically we say a matrix of size n by n is sparse if the number of non-zero elements is less than $n \log_2 n$. Hence we are looking for a better way to store a sparse matrix. We discuss below a more efficient way to store a sparse matrix.

Storing Sparse Matrices
Here is a possible storage scheme for storing sparse matrices.



In the above scheme we maintain two arrays of linked lists, one for accessing the matrix from row side and one for accessing the matrix from column side. We only maintain nodes for entries that are defined. For example, the above data structure shows 4 x 3 matrix (4 rows and 3 columns) with only 4 entries such as (1,2, 88). Each entry is represented by a node that contains the (row,col, value), and pointers to the next defined entry in the same row or column. Note that some lists may be empty. For example, in the above structure, row with index 1 (second row) has no elements, but second column has one element (1,2,88)

Suggested Data Structure

In order to store such a graph, we define a node that contains 5 fields, row index, column index, value of the entry, pointer to the next non-zero column entry, and pointer to the next non-zero row entry. One suggested definition is the following.

```
typedef struct node {
      int row, column,
      double value;
      struct node*  rowPtr;
      struct node*  colPtr;
} node;
```

A matrix is defined as two arrays (row array and column array) of nodes. Hence we define a Matrix data structure as follows. A data structure of two arrays of Node*'s, each pointing to first element in a row or column. The definition of the struct is as follows.

```
typedef struct matrix {
    node**  rowList;     // rowList is a pointer to the array of rows
    node** columnList; // column list is a pointer to the array of columns.
    long dimension;       // assume the matrix is square, that is rows = cols
} matrix;
```

There are few things you need to do to build this structure.

1. Allocate memory for matrix structure
2. Read the dimension (from the data file) and assign two dynamic arrays of node*s.
3. As you read the entries in the file (given as (row,col,value)), create a node and add it to the matrix from row side and column side.
4. Eg: suppose you read: (2, 3, 890) from the file. Create a node with these values. Find rowList[2] and insert the node. Also find colList[3] and insert the *same* node. Note that although we are making the node part of two lists, we only have allocated memory for one node. It is connected from both sides.

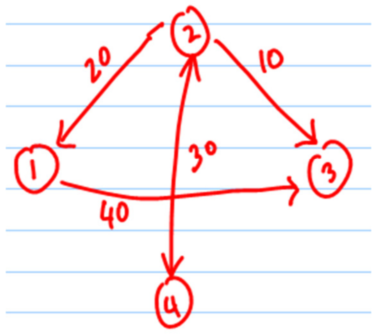
The Assignment

PART 1: First part of the assignment is to build a structure to store a sparse matrix A. You must take the name of the file as a command line argument. Entries of the sparse matrix are given in the file in the following format. First line of the file is an integer that is the dimension of the matrix. You may assume that the matrix is square. That is, the row and column dimensions are the same. Each of the subsequent lines contain two integers, row-index, column-index and value of the matirx entry(double). For example, smallA.txt looks like this.

```
10
0     1      -23
0      6       34
2     3      -56.6
2      8      89.0
3      1      15
5      2       17.34
6      7       13.32
8      5       10.12
8      8        10
8      9        20
9      9      10.34
```

You may assume row and column indices run from 0...n-1. Also assume that files are well-formed. You may not assume that Row and column indices are sorted in the input file. Nodes however must be inserted in order.

PART II: Second part of the assignment is to find out about the underlying graph structure by performing some matrix operations using the sparse structure. For example, we define indegree of a node N as the number of edges that are coming into the node N. In the graph below indegree(1) = 1 and indegree(3) = 2. We also define outdegree of a node N as the total number of edges that are going out of the node N. In the graph below outdegree(3)=0 and outdegree(2)=3 etc.



For example, if you think of the facebook relationship graph, your outdegree may be how many friends you have. Since you assume your friends also call you their friend, your indegree is also the same as your outdegree. This leads us to discuss another useful statistic about a matrix. A matrix is "symmetric" if (i,j)-th entry of the matrix is equal to (j,i)-th entry of the matrix for all i and j. That is A[i][j]=A[j][i] or friendship is mutual and has the same value to both parties for example. You may assume that missing entries of the matrix has a symbolic constant value -infinity(set to smallest double). Note that we are reluctant to use zero for a missing entry since zero may indicate an edge with zero value. We are not showing these missing entries in our structure anyway.

Examples of symmetry: if

```
A = [ 1  2 ]      B = [ 3   4 ]
      [3   4]             [4    3]
```

Matrix A is NOT symmetric

Matrix B is symmetric

You are to write *at least* the following functions. See starter code for function descriptions.

```
node* createNode(int row, int col, double value);
node* insertInOrderRow(node* N, node* head);
node* insertInOrderCol(node* N, node* head);
int is_sorted_Row(node* head);
int is_sorted_Col(node* head);
int initializeMatrix(matrix*, long , FILE*);
int insertNode(matrix*, node*);
void printSubMatrix(matrix* ,long , long , long , long );
long indegree(matrix*, long);
long outdegree(matrix*, long);
double rowSum(matrix*, long);
double columnSum(matrix*, long );
int isSymmetric(matrix* );
double getValue(matrix*, long, long);
int freeAll(matrix*);
```

Running the program: In order to test your program we provide you with the file main.c. To receive full credit, you must pass all assertions. Note that there are number of test data files in the datafiles folder that you can test with. We will continue to update the main to cover as many cases as possible. Hence when you run the code, compile with the latest main.c available.

- **gcc -std=c99 –pedantic –Wall -ansi matrix.c /afs/andrew/course/15/123/downloads/lab4/sparsematrixtester.c**

Downloading Files: Download from **/afs/andrew.cmu.edu/course/15/123/labs/lab4/**

Files to Submit: Three files needs to be submitted. main.c, matrix.h and matrix.c

FAQ: We will leave Frequently asked questions and answers (if any) in the course web https://www.cs.cmu.edu/~ab/15-123S11/Labs/lab4/

Commenting your code: Please comment your code to show what you are doing. The least amount of comments is a short description at the beginning of each function that explains what it does, what parameters it takes, and what the expected output or return value is.

Handing in your Solution

You should just copy the files to handin including the selflog.txt file. **Don't hand in input, obj or exe files. handin the files to : /afs/andrew.cmu.edu/course/15/123/handin/lab4**

How we grade

Your program will be graded by testing functionality of each function. Assertions must pass for all possible datafiles given in the datafiles folder. We will release more descriptive grading scripts. The following 10 points are allocated for style and log files.

```
Style - 7 pts
Selflog.txt - 3 points
```