COMP 3512 Assignment 1

OOP in C++

Due February 10th at 11:59 PM NO EXCEPTIONS

1 Description

Well here it is, your first assignment is ready. We're going to create a matrix class and use it to implement a simple version of the Google PageRank algorithm. This is a very cool, very interesting example of how matrices, something we first encounter in a classroom, can actually be used for something meaningful and practical in the real world. You're not going to believe how easy this is.

2 Submission Requirements

1. No late submissions will be accepted for any reason.
2. Submit your assignment by inviting me to collaborate on Github. Remember that on Github I am known as jeffbcit. You will recognize my avatar.
3. Include a plaintext readme file which must include your full name on the first line and your student number on the second line. Leave line three blank.
4. On line four of your readme.txt file, write either "100% complete" or describe any outstanding issues.
5. Include a one-page PDF called design.pdf (all lower case) which must contain a UML class diagram of your object oriented design. I would like to see classes, important public members of classes, is-a associations, uses (has-a) associations, and any other non-trivial information about your submission.

3 matrix

Your matrix must meet the following requirements:

1. You must include a header file called matrix.hpp and a source file called matrix.cpp.

2. The matrix stores doubles.

3. Implement a default constructor which initializes a square 1 x 1 matrix that contains a 0.0.

4. Implement a constructor that accepts a positive integer n and creates a square n x n matrix that contains 0.0s. Throw an exception if the integer passed to the constructor is zero or negative.

5. Implement a constructor that accepts two positive integers r and c and creates a matrix with r rows and c columns that contains 0.0s. Throw an exception if either integer passed to the constructor is zero or negative.

6. You may use an array or vector for the following constructor. Implement a constructor that accepts an array of double. The size of the array must have an integer square root, i.e., the size of the array must be 1, or 4, or 9, or 16, etc. Create a square matrix of size 1 x 1, or 2 x 2, or 3 x 3, or 4 x 4, etc., and populate it using the values from the array. If the size of the array does not have an integer square root, throw an exception.

7. Implement a 3-parameter mutator called set_value that accepts two integers representing row and column and a double representing the new value for the specified location. Throw an exception if the integers are negative or too large.

8. Implement a 2-parameter accessor called get_value that accepts two integers representing row and column and returns the value in the matrix from the specified location. Throw an exception if the integers are negative or too large.

9. Implement a function called clear which sets all values in the matrix to 0.0.

10. Implement a destructor called ~matrix.

11. Implement an overloaded insertion operator so we can print the matrix to std::cout or other streams.

12. Implement the == and != operators. The equality operators must check if the matrices are the same size and contain the same values in the same locations. Since we are dealing with doubles, it may be sensible to set a tolerance, i.e., if two doubles are within TOLERANCE of one another, then they are essentially the same. If the matrices are not the same size, and they do not contain the same values, they are not equal.

13. Implement the unary increment and decrement operators. You will need a prefix and a postfix version of each. The operators should (respectively) increment or decrement each value in the matrix by 1.0.

14. Implement the assignment operator using the copy-and-swap algorithm introduced in class.

15. Overload operator+=, operator+, operator-=, and operator-. These will only work if the matrices are the same size. If the operands passed to the operator are not the same size, throw an exception.

16. Do not overload operator/ or operator/=.

17. Overload operator* and operator*=. For operator* and operator*=, be careful. You will need to remember that for 2 matrices a and b, the product matrix c is possibly a different size. If matrix a is 1 x 4 and matrix b is 4 x 3, the size of the product matrix c will be 1 x 3. If the number of columns of the first operand is not equal to the number of rows of the second operand, throw an exception.

4 Google PageRank

Here's an implementation-free description of a simple version of the Google PageRank algorithm. This is the version of the PageRank algorithm which you will use in your assignment. Don't be alarmed. Read it carefully. This is surprisingly and pleasantly easy:

1. Let W be a set of webpages (a web) of size n.

2. Let G be a square connectivity matrix. A connectivity matrix is a square matrix of 0s and 1s used to represent the connections in a graph (or web). If there is a hyperlink from page j to page i, there is a 1 at location [i][j]. If there is no link from page j to page i, then there is a 0 at location [i][j].

3. Suppose we have a tiny web W of 4 pages: A, B, C, and D, and apart from page D which has no links to it or from it, each page contains one link to each of the other pages, but not to itself. That is, page A contains a link to page B, and to page C, but not to page A or to page D. Page B contains a link to page A, and to page C, but not to page B or page D. And so on. We say that the jth column of the connectivity matrix contains the outbound links from page j. Our connectivity matrix will look like this:

```
G =   a b c d
    a 0 1 1 0
    b 1 0 1 0
    c 1 1 0 0
    d 0 0 0 0
```

4. Let's define the importance of each page by the importance of the pages that link to it. (Hey wait, that sounds a little like recursion!). If we do

this, we can use the connectivity matrix as a starting point for determining the relative importance of each page in our web. Note that a typical connectivity matrix G is probably very big (how many billions (trillions) of web pages are there these days), but very sparse (lots of zeros). Its jth column shows the links on the jth page, and the total number of nonzero entries in the entire matrix is the total number of links in our web W.

5. We can define r_i and c_j to be the row and column sums of G respectively. These quantities are the in-degree and out-degree. The in-degree is the number of pages that have links to our ith page, and the out-degree is the number of links on our jth page to other pages.

6. Take a break and this about this. This is neat. There's a lot of math in search engines, and there's some pretty interesting vocabulary involved. We're using matrices. We're using a special kind of matrix called a connectivity matrix which is all zeros and ones. The matrix can be sparse, and has row and column sums. The sum of 1s in a row is the in-degree because it is a count of the links to the page at row i. The sum of 1s in a column is the out-degree because it is the number of links out from the page at column j.

7. Mathematically-speaking, the "importance" of page i (x_i) equals the sum over all pages j that link to i of the importance of each page j divided by the number of links in page j:

$$x_i = \sum_{j \in W} \frac{1}{N_j} x_j$$

Figure 1: The importance of a page

8. We can modify our connectivity matrix to show us this "importance" if we divide each value in each column by the sum of each column. Since it is a connectivity matrix, the values in the cells are either 0 or 1, so each cell containing a 1 is divided by the number of 1s in the column. For example, in our connectivity matrix G, the sum in the first column (column A) is 2, so we divide each element by 2 to get 0.5. We can observe that every nonzero column now adds up to 1. Let S be the matrix constructed according to this rule:

```
S =   a   b   c   d
    a 0   0.5 0.5 0
    b 0.5 0   0.5 0
    c 0.5 0.5 0   0
    d 0   0   0   0
```

9. In matrix S, the value in S[i][j] is the "probability" of going from page j to page i. Note that every non-zero column adds up to 1, but we have that pesky final column of zeros to worry about. There are no links to page D, and there are no links away from page D. We have to do something about this, otherwise page D is going to end up at the bottom of the PageRank no matter how interesting its contents. We want to replace every column of zeros with a column whose elements equal 1/n (recall n is the dimension of our square connectivity matrix):

```
S =   a   b   c   d
    a 0   0.5 0.5 0.25
    b 0.5 0   0.5 0.25
    c 0.5 0.5 0   0.25
    d 0   0   0   0.25
```

10. Are you still with us? Good! The matrix S is now a stochastic matrix, or to be more specific, a left stochastic matrix because the columns all sum to 1. In a left stochastic matrix, the elements are all strictly between 0 and 1 and its columns all sum to 1. We're almost finished. We can also call S a probability matrix, and we will use our probability matrix to construct a transition matrix for a Markov process. This sounds much more complicated than it really is. There are two steps:

11. We need to introduce the notion of a random walk. We need to multiply our probability matrix by a random walk probability factor. For our lab, we will designate this variable p, and set p = 0.85.

12. The next step is the most complicated step (and it's not complicated at all). We need to add (1 - p) multipled by a matrix Q, whose every element = 1 / n. (recall that a rank 1 matrix has a single linearly independent column). Let's call the transition matrix that results M. Use the formula:

```
M = 0.85 * S + (1 - 0.85) * Q
```

13. Now comes the fun part, the "dynamical system" portion of our algorithm. Start with a matrix of size n x 1, a column matrix of 1s called rank:

```
rank = 1
       1
       1
       1
```

14. The final 2 steps are the Markov process (aka the dynamical system aka the power method):

15. Multiply the transition matrix M by our column matrix rank, and then multiply M by the result and then keep doing this until the rank stops changing (result converges), e.g., M * rank = rank. In this case, we get:

```
rank = 1.2698
       1.2698
       1.2698
       0.1905
```

16. Finally (last step!) divide each element in rank by the sum of the values in rank (scale rank so its elements sum to 1):

```
rank = 1.2698 / 3.999 = 0.3175
       1.2698 / 3.999   0.3175
       1.2698 / 3.999   0.3175
       0.1905 / 3.999   0.0476
```

5 Requirements

1. Write a C++ program that opens a connectivity matrix and calculates the Google PageRank algorithm for the web described by the connectivity matrix. The connectivity matrix will be in a plaintext file called connectivity.txt. It will be square, and composed entirely of space separated 0s and 1s (this is an example):

```
0 1 1 0
1 0 1 0
1 1 0 0
0 0 0 0
```

2. Your program must print the result to cout like this:

```
Page A: 31.75%
Page B: 31.75%
Page C: 31.75%
Page D:  4.76%
```

6 Grading

Your assignment will be marked out of 40:

- Your code must compile without any warnings using g++. If your program does not compile, you will earn zero. If your code compiles with warnings, you will lose marks.
- Your code must assume the connectivity matrix file is called connectivity.txt and it is in your CLion project. If I have to modify your code to edit a file path, you will lose marks.
- Your code must be commented and must follow consistent and correct formatting norms.

- Each class you create must be in its own header and source file.
- Do not use capital letters for identifiers.
- Make function parameters const when the function should not modify the parameter. Make member functions const when they do not modify the state of the object. Use const (or constexpr) to define constants (do not use magic numbers).
- Your program must use encapsulation and object oriented programming. Encapsulate all logic in classes, member functions, friend functions, and (possibly) a few non-member functions.
- I think there is some inheritance here. A connectivity matrix seems to be a specialized matrix that only contains 0s and 1s, for example.
- Your main method must be short and it must be in a separate file called main.cpp. If the main method is not in a separate file called main.cpp, you will lose marks.
- No global variables.

Please remember that this is an individual assignment. I strongly encourage you to share ideas and concepts, but sharing code or submitting someone else's work is not allowed.

Good luck, and have fun!