COSC 320 - Advanced Data Structures and Algorithm Analysis
Project 1

Dr. Joe Anderson

Due: 14 March

1 Description

You will implement a recursive algorithm to compute the inverse of a matrix, then apply that solution to the problem of analyzing economical "Input-Output" models.

2 Specifications

Write a `c++` class, `Matrix` (possibly reusing work from Lab 4), to store a matrix of integers. Include methods to perform addition, subtraction, scalar multiplication, transpose, and inverse. Demonstrate the correctness of each of these routines in the program output. Details on each of these tasks is provided below.

Using the operations above, ~~write a program to perform linear regression with Ordinary Least Squares on a set of vectors.~~ The input for the program will be provided in a text file, which should be specified via command-line. For example, the command

```
./iomodel data
```

should run the program to read the data in `data` and store it in a data structure of your choice. The data file will contain a list of industries/sectors, a line of `---`, an $n \times n$ matrix in standard form with elements separated by a single space, and separate rows separated by a newline; following the matrix will be a line containing only `---`, then an n-dimensional vector in a similar format. An example (shortened) input file may look like:

```
Coal
Electricity
Produce
---
0.6 0.02 0.1
0.3 0.2 0.4
0.2 0.4 0.3
---
20
34
80
```

The input matrix will be of varying size, so your program will need to determine the correct n by analyzing the input file.

2.1 Input-Output Model

The Input-Output Model is a mathematical framework developed by Wassily Leontief to represent the interdependencies of various sectors in a national or regional economy. To understand the basics of the model, consider an economy in which multiple products are produced, and some products depend on the available of others. For example, to produce electricity, one would need a certain amount of coal or other fuel; and to acquire the fuel, one would need access to electricity. There could be many other products, which may or may not depend on each other. Finally, there is also likely to be an "external" demand for a product which accounts for commercial or other demands.

Leontief's idea was to model this mathematically by a labeling each product/sector by a number $1, \ldots, n$ and then constructing a matrix $A\mathbb{R}^{n\times n}$ where entry A_{ij} represents the amount of product i required to produce product j. The external demand is captured in a vector $d \in \mathbb{R}^n$ where d_i is the external demand for product i. Using this setup, one can model the total output, x_i, required of a sector i as:

$$x_i = a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n + d.$$

To write this in a much more compact form, we can view the model as:

$$x = Ax + d$$

and, by using some standard linear algebra, can be manipulated (check this!) into:

$$x = (I_n - A)x = d$$

where I_n is the $n \times n$ identity matrix. Finally, to conclude how much total of a single product will be needed to satisfy all demands of the economy, we need only compute

$$x = (I_n - A)^{-1}d.$$

The term $(I_n - A)^{-1}$ is the *inverse* of the matrix $(I_n - A)$, or the unique matrix B so that $B(I_n - A) = I_n$. To compute such a matrix, see the section below for one method (there are more efficient ones, but they are slightly more complex).

2.2 Matrix Inversion

Given an $n \times m$ matrix A, the *inverse* of A is the $m \times n$ matrix A^{-1} so that $AA^{-1} = I_n$, where I_n is the $n \times n$ identity matrix.

For your implementation of inverse, you may assume that A is square $(n \times n)$ and symmetric. To make sure your recursion reaches an acceptable base case, you need to pad the input matrix as

$$A \mapsto \begin{pmatrix} A & 0 \\ 0 & I_k \end{pmatrix}$$

so that $n + k$ is a power of 2. Then note that

$$\begin{pmatrix} A & 0 \\ 0 & I_k \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} & 0 \\ 0 & I_k \end{pmatrix}.$$

Then, the algorithm to compute A^{-1} is as follows:

1. Divide A into submatrices:

$$A = \begin{pmatrix} B & C^T \\ C & D \end{pmatrix}$$

where B, and D are also symmetric and are $(n/2) \times (n/2)$ matrices.

2. Recursively compute B^{-1}.
3. Compute $W = CB^{-1}$ and $W^T = B^{-1}C^T$.
4. Compute $X = WC^T$.
5. Compute $S = D - X$
6. Recursively compute $V = S^{-1}$.
7. Compute $Y = S^{-1}W$ and Y^T.
8. Set $T = -Y^T$ and $U = -Y$.
9. Compute $Z = W^T Y$ and set $R = B^{-1} + Z$.
10. Assemble the inverse of A as:

$$A^{-1} = \begin{pmatrix} R & T \\ U & V \end{pmatrix}.$$

Be sure to extract only the top-left $n \times n$ corner, if padding the input matrix was necessary.

Note that the base case of the recursion can be a 1×1 matrix, in which case the inverse is just the 1×1 matrix containing the reciprocal of the single element.

For more general use cases, you will need to add a test to verify that a matrix is symmetric, and add a case to invert a non-symmetric square matrix by computing $A^{-1} = (A^T A)^{-1} A^T$, where $A^T A$ is guaranteed to be symmetric.

For more information, see the course textbook, section 28.2.

Important caveats to consider:

1. The above algorithm description is obviously not fully detailed. Be sure to watch for edge cases, take into account matrix operation compatibility, matrix dimensions, and the assumptions needed by the algorithm.
2. **You may not use the c++ standard library data structures or algorithms.** If you want to use linked lists or related data structures, you should implement a specialized version for this project from scratch. If you are in doubt about whether importing specific outside code is acceptable, consult the instructor.

2.3 Output

Submit code that compiles into *two* executables:

- One "driver" program that demonstrates the correctness of the individual matrix operations used. Demonstrate that the various individual matrix operations are correct, by showing them on examples; for instance, give a matrix A, calculate A^{-1} and show that $A * A^{-1} = I_n$. Include different tests of the Input-Output model and some solutions. Vary the size of the input and make a note of how the running time changes with relation to the input size. Record both the time and number of operations performed by the algorithm; count an operation as being any single arithmetic operation on one of the data elements (comparison, addition, multiplication, assignment, etc.).
- One executable to read in the economy data and report the production needed. Your program should report to the user how much of each product to simultaneously satisfy all demands. In the example provided above, the solution would look like:

```
Amount of each product needed:
Coal:         133.72 units
Electricity:  236.45 units
Produce:      287.61 units
```

Be sure to fully document your code and include a `Makefile` to compile the code.

3 Submission

Submit a `.zip` file called `Project1[LastName].zip` (with `[LastName]` replaced with your own last name) containing your source code, `Makefile`, and documentation, then upload it to the course MyClasses submission page.

Second, print out your code and documentation, to be turned in on the due date (typically at the start of the following Lab).

4 Bonus

If any bonus are completed, be sure to note it in the README file and provide appropriate output to demonstrate its correctness.

1. (5 pts) Add a test to your `Matrix` class to test if a given matrix is non-singular. The simplest test is to compute the *determinant* of the matrix. Consult a linear algebra textbook or another resource to find the simple recursive formula for this.

2. (20 pts) Add an implementation of Strassen's algorithm for matrix multiplication. Include tests and thorough benchmarks (similar to a lab report) to compare it to the basic method using the definition.