**605.202: Introduction to Data Structures**

**Christine Herlihy**

**Lab #2: Analysis Paper**

**Due Date: October 27, 2015**

**Dated Turned In: October 28, 2015**

Christine Herlihy
EN 605.202 Data Structures- 10/27/2015

**Lab #2 Analysis**

I. General Commentary

This project included four primary activities: (1) reading in a set of matrices, along with their dimensions, from an input file; (2) assessing the validity of each matrix (i.e., with respect to dimensions and the potential for missing values), (3) calculating the determinant of each matrix; and (4) sending the results to the output file in a clean and easy-to-read format.

I began by designing a Matrix class, so that I could ensure each n*n Matrix object I created would have a private data value indicating its order. I also wrote a function such that, given an integer value for *order*, a Matrix object's corresponding *order***order* matrix could be initialized with a simple function call. Within the Matrix class, I also included a function to recursively calculate the determinant of a Matrix object; my stopping case was a Matrix of order 1, where the determinant is equal to the sole Matrix element (i.e. tempMatrix[0][0]). I then wrote the main driver of my program, which included a method to read in the matrix input file, parse for integer values, and create Matrix objects containing the values specified—and of the orders specified—in the input text file. With the Matrix class, I created a constructor to allow for instantiation of an $n*n$ matrix vis-à-vis an integer parameter, $n$. Within the portion of the code where the Matrix objects are created, I skip any values of n that are $\leq 0$, as these would not represent valid dimensions. This way, any matrix that is successfully created must be an $n*n$. Each matrix that is successfully created is stored in an output array, which I then loop through to print each matrix, along with annotations (i.e. numbering the matrices, and indicating their dimensions and the value of their determinants in an easy to read format) to the output file. I included additional input matrices, including some with orders greater than 6, and I also display the processing time for each matrix's determinant calculation.

II. Justification for Design Decisions

An array is a natural data structure to use for implementing a matrix, as it allows for random access, which is necessary if we are to access specific elements in the matrix in order to calculate the determinant. While we were required to use an array to implement the matrices, I would have chosen an array even in the absence of such a requirement, as I feel it to be a logical and readily exploitable choice. I opted to design a separate Matrix class in the interest of encapsulation; additionally, this design allowed me to store each matrix from the input file in an output Matrix object array.

My solution to calculate the determinant is recursive per the requirements, and takes the form of the Laplace Expansion, in which the determinant of an $n*n$ matrix A can be found by the following formula: $\sum_{j=0}^{n}(-1)^{1+j}\, a_{0_j} M_{1_j}$, where M represents the minor, or matrix of one order less than A, that is formed by deleting the row and column that $a_{1_j}$ is in. An iterative solution is also possible, in which a formula for the determinant of a matrix A is the product of smaller determinants that are easier to compute (i.e. 2*2 or 3*3), but need not go as far down as the recursive base case of 1*1. The recursive solution quickly becomes quite computationally expensive, as its cost is O(n!); thus, iterative implementations, or a hybrid approach involving memoization, in which each 2*2 determinant (once calculated) is stored and referred to in a look-up table, may be preferable for larger order matrices.[1]

III. Lessons Learned

Designing and implementing this project helped me to get a better grasp on the mechanics of recursion. To understand how I could write a completely recursive function, without relying on a

[1] https://en.wikipedia.org/wiki/Determinant#Applications; http://www.vikparuchuri.com/blog/find-the-determinant-of-a-matrix/

stopping case of order = 2, in which the determinant of a 2*2 matrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is simply *ad-bc*, I calculated several of the determinants of the matrices in the input set recursively by hand. This helped me to understand how I would need to set up my function so that I would be breaking down my initial matrix into a set of smaller matrices each time, so as to finally wind up with a series of matrices of order 1, whose determinants were equal to the element each contained. I also learned that the determinant is undefined for non-square matrices, and can be used to assess linear independence.[2] Reading the matrices in from the input file was a bit more challenging than I initially expected, due to the need to parse the input txt file both within and between each line. In my first regex-based approach, I realized that because I was trying to read in each integer as a character, I had spliced the negative signs from their associated integers, and was thus erroneously excluding the negative signs from my matrices. Eventually, I realized I could use whitespace delimiters and store all of the integers in an array; I could then pull from this array to get a value for the dimension and then create each matrix.

## IV. Alterations to Consider in Future Iterations

In future iterations of this program, I would hope to include the option for a hybrid method to calculate the determinant using a closed formula for order 2 matrices. An additional feature that I think could also be helpful for instructional purposes would be to print out the intermediate calculations in a step-by-step, indented format, so that someone could trace the recursive steps the algorithm takes to arrive at the final answer. When I was trying to understand this process myself, I found that visualizing each step and seeing how each step related to all the subsequent steps was very helpful. A GUI component could also make this process more clear and user-friendly. Finally, my current code forces each matrix that is input to be an *n*n* matrix, and does include logic checks to ensure a matrix is not initialized with an $n \leq 0$, but the error checking mechanisms could be made robust, such that a matrix with improper dimensions could be read in, and the determinant calculation alone would return an error message.

## V. Issues of Efficiency

The cost of using a recursive algorithm to calculate the determinant of each matrix is O(n!), because to get our final answer recursively, the summation formula requires us to find and multiply by the minor of each matrix that is one order smaller than *n* (i.e. n*n-1*n-2…((n-n)+1). Thus, while this algorithm is relatively efficient for matrices of small dimensions, it quickly becomes increasingly expensive, as indicated by the processing time data in my output file. There are other alternatives, including Gaussian Elimination, which involves transforming the matrix to an upper triangular matrix with 0s in the bottom left corner. As the processing time of the test cases indicates, the cost of computing the determinant of such a matrix is lower than the cost of computing the determinant for a matrix in which all the elements are non-zero values. Such techniques have a cost of $O(n^3)$, and are thus preferable for larger order matrices.

[2] Fundamentals of Mathematical Physics Kraut, Edgar A. (2013) *Courier Corporation* p. 24