605.202: Introduction to Data Structures

Thomas Muamar

Project 2: Determinant of Matrix Analysis

Due Date: July 20, 2021

Dated Turned In: July 19, 2021

Project 2: Determinant of Matrix Analysis

Justification for data structures/reasons for choices:

In order to get the determinant of a matrix I just have two classes one of which to find the determinant and another to read input file and display the matrix and determinant of that matrix. When storing the matrix for this problem you can take few different approaches like the different linked list structures or an array structure. The structure I choose to store the matrix was in a two-dimensional array. The time complexity of a two-dimensional array would be O(NxM). Where if the values of n and m are equal to each other then the complexity would be O(n^2). For the space complexity it would be O(n^2) since the matrixes being calculated for the determinant are square. Array structure might take up more space, but it has better access within the array to allows for an easier access unlike a linked list structure were u have to traverse through the array and do not have access to other positions.

In order to calculate the determinant of a matrix you have to make sure that it is a square matrix, or it is unable to calculate the results of that matrix. In order to calculate a larger matrix, it is easier to dividing up that matrix into different portions and then adding them together to produce the result. When creating the calculation for the determinant I use a recursive process where it will divide the matrix into small parts and calculates the determinant for each of these parts by repeating the process. So basically, for larger matrixes it will go through each row by one until it has reached the end of the row and start increasing the column by one. The base case in the determinant class would be the 1x1 matrix which basically calls on itself as the result. The recursive portion for the larger matrixes would be the equation det = (int) (det+mat[0][n] * Math.pow(-1, (int) n) * determinant(newMat, mat.length-1)) which gets repeated depending on the length of the matrix.

In the main class of my program this is where the matrix is read in the inputted text file. In order to read each matrix, the program starts by reading the order of the matrix and then it will input it into the array of that sizes. When printing out the matrix I basically only checked the rows because we are conducting an NxN matrix. If the inputs are unequal which will produce a non-square matrix it will give an error saying it cannot calculate the determinant because it is impossible to calculate one for a non-square matrix. As soon as it recognizes a non-square matrix it errors out and does not calculate the determinants of the remaining inputs.

What I Learned:

This project has taught me a lot more about how to calculate the determinant because I watched YouTube videos to be able to get a better refresher of how to calculate them. This allowed me to produce a code that allowed me to calculate the determinant of the matrix making it easier to produce results for larger matrixes.

Also learned that this two-dimensional array will have good time complexity, but space complexity might not be the best case if the input matrix gets bigger. Doing this project has also got me a better understanding of how to create a recursive function.

Got a better understanding of how to do a recursive setup for different problems.

What I might do differently next time:

If I did not need to do it in a recursive format, I would probably have tested an iterative program to see which would perform better when running matrixes that are limited to an order of 6.

Another thing that I would do differently that I had a lot of trouble trying to figure out is when there is not a square matrix it would error out and then go on an calculate the next matrix in the input

text file. I could not figure out a way to have the error to end and then continue to calculate the next matrix in the text file.
One more thing that I would do differently would be to differentiate the row an column instead of making it the same variable for each. This would allow an easier time in printing out errors when it is not a square matrix.

Issues of Efficiency:

With a recursive setup for calculating the determinant is not as efficient as having a non-recursive way of doing it. The reason for that is because as the matrix gets larger it will end up requiring a lot of memory space to hold the values on the systems stacks. The reason for this is because every time it calls the recursive call it will have to keep track of that and in order to do that it puts it in a stack. Not only does a recursive setup have space issues but it also is slower. The time complexity of a recursive function is o(n) but that depends on how many times that recursive function gets called.

Considering an iterative implementation:

With an iterative approach I would make a case for each order since we only need to go up to the 6 order. Taking this iterative approach would require a lot more code because you would have to create a different formula for each order that is needed. For example in each order the code would contain the equation in this format det = mat[0][0] * mat[1][1] - mat[0][1] * mat[1][0]. As the order gets larger than there would have to be separate variables to break up the matrixes. Like in a 4x4 it would have to be broken to 3x3 matrixes and then to calculate that to break it into a 2x2 matrix to get the value of each of the 3x3 matrixes within that 4x4. Which ends up causing 10 times longer code than using a recursive implementation. With recursion we just have one equation that gets repeated each time which makes a lot simpler and easy to understand.