605.202 DATA STRUCUTRES

GUAN YUE WANG

JHU ID: gwang39

LAB 2 ANALYSIS

JULY 16, 2017

General Program Design

The program simulates a matrix determinant calculator to calculate the determinant of a given matrix and provide corresponding calculation summary information.

The overall program is controlled by a while loop to read input text file number by number. Based on the matrix size N taken, it runs a N*N nested loop to get each matrix element for each column at per row level. Once it gets a complete matrix, matrix determinant calculator is used to evaluate the matrix and calculate the determinant recursively. Then the summary information contains matrix itself and its determinant get export to output file. This 'read and calculate' process keeps going on until there's no more number left in the input file.

The program consists of the main class and one other class determinant calculator. Determinant calculator class provides several functions for given matrix, it can print original matrix in a formatted way, calculate determinant of a given matrix utilizing recursive calls, and provide summary information as formatted text for output. The main class contains a validation method to check whether everything in the input text file follows the correct patter (i.e. size N and then N*N matrix format). In addition, the main entry point can be ran with or without command line arguments ([input file] [output file]). If the user choose not to do so, the program console would prompt user to input I/O file locations.

Data Structure Implementation And Appropriateness

Array is used as the major data structure to store data and support recursive calculations in the program. It is considered a good fit with our program as our matrix size is relatively small and it works well with our determinant calculation method.

During the data input stage, a 2D N*N array is created depends on size N we get. Afterwards, a nested loop is initiated to feed numbers one by one into the 2D array. After the determinant calculation of the current matrix, this array is cleared and re-defined based on next N. This process goes on till the end of the file. In this case, 2D array is an intuitive way to represent the data as our matrix size is small and there is not many sparse matrices. In addition to calculation, it's easier to display 2D array in a formatted matrix view for users.

Regarding the calculation, after we have the matrix data ready, a recursive method comes into play to calculate the determinant utilizing the array structure. As this method implements the Laplace expansion, it expresses determinant of a matrix in terms of the determinants of smaller sub matrices. The Laplace expansion loops through one row, expresses the determinant as a weighted sum where each row element is multiplied by the determinant of the minor 'itself'. (Appendix 1)

Alternative Approach

The determinant calculation in this program leveraged recursion and Laplace expansion. This is an extremely inefficient approach for large matrix size considering it recursively calls minor itself and grows matrix operations. One of the alternative iteration solution can be Decomposition method. This includes lower upper decomposition and Cholesky decomposition which can decrease the complexity from O(n!) to O(n^3). For example, lower upper decomposition involves express matrix in terms of

lower triangular matrix L and upper triangular matrix U and permutation matrix P. As the determinant of these 3 matrix can be easily calculated, determinant of the original matrix can also be quickly obtained by multiplying these 3 determinants together. Both lower and upper triangular matrix can be found using nested for loop to iterate through columns at per row level.

Regarding the data structure, linked implementation can be a better choice compared to array implementation when we have a large matrix size or we have sparse matrix. In the event of large matrix size, the combination of array and recursion can cause significant inefficiency as we need to allocate time and space to calculate minor matrix whereas linked implementation would provide a better big O by breaking and re-linking nodes together. On the other hand, if we have a sparse matrix, a lot of space in the array would be 0/null and thus get wasted. Instead, if we use linked implementation, the space can be better used by only storing non-0 values in the matrix. The linked structure can be implemented with three kinds of nodes (see Appendix 2 for illustration):

- Header node would tell total number of rows, columns and non-zero values.

- Row node would store current row number and point to two places: next column node in this row that stores non-zero value as well as next non-empty row.

- Column node contains column number, the actual data value, and points to next column node.

Time And Space Efficiency

As recursion is used in Laplace expansion to accommodate determinant calculation, each method call must evaluate n with determinant of size n-1. Consequently, total number of recursive call is n*(n-1)*(n-2)*...*2*1 which leads to O(n!). Thus, the algorithm can become extremely slow when we have a large size matrix. Please see Appendix 3 for an demonstration of current time complexity using real run time records with different n size matrix. Therefore, as mentioned on the above, LU decomposition or Cholesky decomposition can be a better way to calculate determinant in order to decrease the time complexity to O(n^3) from O(n!).

From a space efficiency perspective, as the recursion requires keeps allocating space to store minor matrix for Laplace expansion, the space used is heavily wasted as we allocate space for data that we don't need in current operation. Thus, an alternate approach should be considered in order to achieve better space efficiency. Also, as mentioned above, a linked implementation would be better choice in the case of sparse matrix.

Learning And Takeaway

There are many takeaways from this lab. One of the most important ones is how the selection of data structure and algorithm can lead to huge impact on time and space complexity. It is surprised to see how Laplace expansion's O(n!) compared to Decompositions' O(3) as well as how space is affected by the matrix size and density. In summary, a combination of data structure and algorithm should be considered at the planning stage to evaluate the most efficient and suitable solution to program things out.

As mentioned above, I look forward to the next lab where I can use linked structure and decomposition method for determinant calculation. Although it's might take more time to figure out the structure and logic, it would be really interesting to see the complexity difference in the actual program. On the other hand, things I would do the same is the same is to keep the flexibility in the program I/O process. In this lab, both command line arguments and console input is available as options for user file to specify input and output locations. I believe all these user-friendly features would be valuable for user experience enhancement and adoption of the program.

Enhancement

There are several enhancements in the program to improve the process flow in the aspect of general design, user I/O, validations, and user display

To begin with, the general design of the program implements OOP concept by having a determinant calculator available to take the matrix, calculate determinant, and generate summary information. Compared to putting all methods in main class, it brings encapsulation to make sure the structure and creates the flexibility for people to modify calculation or display format later.

As a part of the design, there's also enhancement at user I/O level, it grants user the option to choose whether they want to provide input and output locations as command line arguments or not. To be more specific, they can state input and output file locations as command line arguments so the program gets the I/O information directly from there. Meanwhile, they can also run the program without providing any command line arguments and the program would prompt user for I/O locations in the console and read from there.

In addition to general design, the program also has an extra validation method to make sure the input file follows the correct matrix format (size N followed by N*N matrix) so no processing power is wasted prior to we confirm the completeness of the data in the file. Besides, output is formatted in a way that matrix has border around it so people can easily tell whether they are looking at the matrix or output summary information.

Appendix

Appendix 1. Laplace expansion

Matrix

|1 2 3|

|4 5 6|

|7 8 9|

Taking the Laplace expansion along the first row leads to below

= 1* |5 6| - 2 * |4 6| + 3 * |4 5|
|8 9| |7 9| + |7 8|

This repeats till we get the smallest sub matrices to evaluate determinant

= 1*(5*|9|-8*|6|)-2*(4*|9|-7*|6|)+3*(4*|8|-7*|5|)

Result

= 1*7 - 2*(-6) + 3*(-3) = 7 + 12 - 9 = 10

Appendix 2. Linked implementation to store matrix information

Let's say we have below matrix

RC	0	1	2	3
0		1		
1	5		4	
2		2		3
3		3		

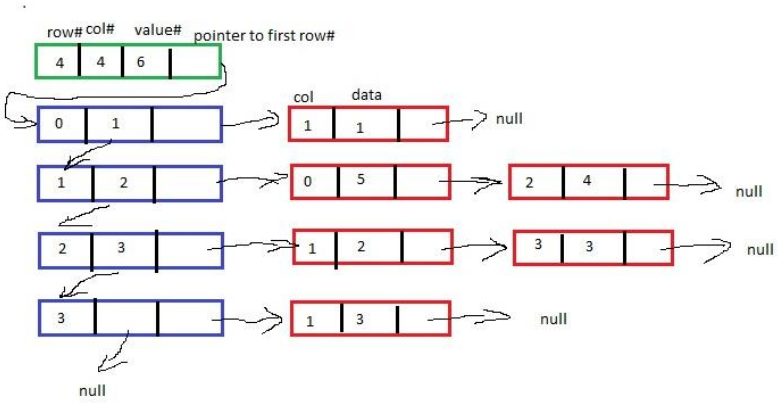
- Header node will show total 4 rows, 4 columns, 6 non-zero values, and point to next non-empty row

- Row 1 - Row node will show current row #1, next non-empty row #, and first column # contains value

- Row 1 columns, Column node will show current column number, data value, and point to next column node

- when finish row 1, follow row 1 pointer to next non-empty row to find column values

Graphically



Appendix 3. Current Time Complexity O(n!) with real program run time tracking

Matrix Size n	Run time (microseconds)
1	9
2	10
4	11
8	46
10	1270
20	Takes forever...