605.202 DATA STRUCUTRES

GUAN YUE WANG

JHU ID: gwang39

LAB 3 ANALYSIS

AUGUST 04, 2017

General Program Design

The program simulates a matrix determinant calculator to compute determinants of all matrices in a text file and output all corresponding calculation summary.

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 elements per row for each column utilizing double linked list with header nodes. Once it gets a complete matrix, matrix determinant calculator is used to evaluate the matrix and calculate the determinant recursively. After that, summary information contains matrix itself and its determinant get exported to an 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, DeterminantCalculator class, LinkedList class, and Node class. The Node class constructs a generic node for LinkedList class. And LinkedList class provides double linked list data structure with deader node to help store matrix for manipulation. After the matrix is acquired in a form of combination of double linked list, Determinant calculator class comes into play to calculate determinant and provide calculation summary information. 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

Double linked list with header node is used as the major data structure to store data and support recursive calculations in the program. Node class acts as a base to this data structure by creating the generic node to support linked list structure. Each node contains the actual data, reference to previous node as well as reference to next node.

The LinkedList class constructs a double linked list, it contains multiple methods including add, setHeader, isEmpty, listLength, and nodeValue. To be specific, setHeader method creates a head node indicating the row index, nodeValue returns a value of node based on index provided, listLength gives us the length of the linked list, and isEmpty checks whether the list has any value. Last but not the least, add method adds node with data value to the list. During this process, it would update its previous and next node reference in order to accommodate the double linked list data structure. (Appendix 2)

During the data input stage, depends on the size N, a nested loop is used to loop through each row at per column, each row is put into a double linked list and then all rows are combined together into a list array. Regarding the calculation, after we have the matrix data ready, a recursive method comes into play to calculate the determinants. As this method implements the Laplace expansion, it expresses determinant of a matrix in terms of the determinants of smaller sub matrices. (Appendix 1) During this process, required elements in the matrix are located and re-linked together as a new linked list to form the minor matrix for recursive computation. In addition, header node is utilized to indicate which row the linked list represents.

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, compared to the array implementation in previous lab, the linked list implementation is harder to construct and utilize in the matrix manipulation. From a data access perspective, The cost of traversing a linked list is definitely higher than indexing elements in an array. However, as the major step in our computation is re-linking elements for minor matrix, linked list can actually provide a constant time by changing pointers. On the other hand, 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 provides a better big O using pointers. Furthermore, if we have a sparse matrix, a lot of space in the array would be 0/null and thus get wasted.

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!). In addition, as seen it Appendix 3, it looks like linked list takes less time when matrix size gets larger and more constant time for smaller matrix size compared to array.

From a space efficiency perspective, my linked list approach doesn't provide a significant improvement as it still allocate space to store re-linked list as minor matrix. However, if a better linked approach is used, it might take smaller space for breaking and re-linking different part of matrix. Also, linked implementation that skips zero could be better choice in the case of sparse matrix for space efficiency.

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, if I can do this lab again, I would definitely try decomposition method for determinant calculation. Although it's might take more time to figure out the structure and logic, it is really interesting to see the complexity difference between two approaches. 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|)

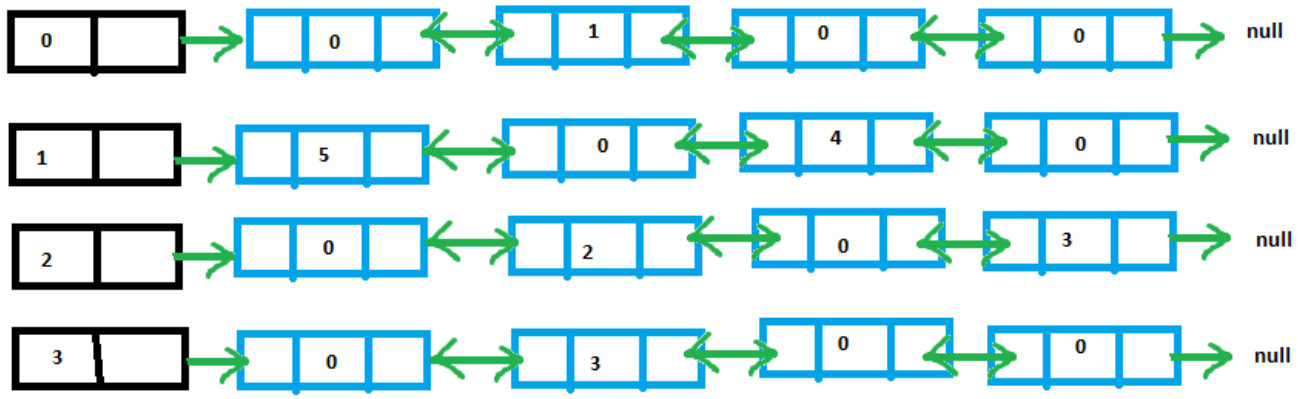
Result

= 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		



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

Matrix Size n	**Array Run time (microseconds)**	**Linked List Run time (microseconds)**
1	9	7
2	10	7
4	11	7
8	46	30
10	1270	1013