Lab 3 Analysis - Noboru Hayashi

I. Data Structure

In this Lab assignment, I defined a Node class with a pointer to next node which is a singly linked list, to represent a matrix $\begin{matrix} A & B & C \\ D & E & F \\ G & H & I \end{matrix}$ as A →B →C →… →H → I → (null) . With this implementation, any square matrix with any order can be represented.

For calculating its determinant, the recursive function calDet() divides the matrix into submatrices until its order is 1. This base case returns the head node's data as sub-determinant, which is used to calculate the determinant of the upper order matrix.

The algorithm of the program is as following, assume Node mat is the top-left element of the matrix:

1. For order n = 1, calcDet(mat) = mat.data;
2. For order n > 1, calcDet(mat) = ∑(sign)^i * mat.get(i) * mat.reduce(i, n).
   - i ∈ [0, n-1],
   - mat.get(i) access i-th element from the head,
   - mat.reduce(index, order) creates a sub matrix contains all elements from the matrix except for the index i, i+n, i+2*n + …
     E.g. For the matrix above, mat.reduce(0, 3) creates a sub matrix: E→F→H→I → (null)

For implementing a matrix with some data structure, my first idea was a linked list in which each node has a pointer to the right and below, so the matrix can be represented as:

A→B→C
↓ ↓ ↓
D→E→F
↓ ↓ ↓
G→H→I

However this implementation adds complexities to codes to create a matrix, or reduce a matrix, for example, creating a new matrix requires keeping a pointer of the previous row since integers from the source files are read one by one. Then I found it is much easier and simpler to implement the square matrix with a singly linked list, since indices (# of parsing from the head needed to access the specific node) follows specific patterns for nodes at the same row or same column in the square matrix.

II. Cost Analysis

a. While reading the matrix with order n from a file, creating a new linked list to represent it costs O(n^2) time and space.

b. Each time reducing a matrix into a new lower-order submatrix: Node.reduce() needs to parse the entire matrix iteratively, therefore the time complexity of reduction is O(n^2) . All reducing a matrix needs to create a new sub matrix, so it takes O((n-1)^2) = O(n^2) space.
c. At the level n of the recursive function calcDet(), reduce method is called n times, hence each one-level recursive action requires O(n^3) time and O(n^3) space. And for a given matrix with order n, the calcDet() function requires n levels of recursive operations, so the total cost of time and space would be O(n^4). Since the matrix is square, it has total m = n^2 elements, and the program requires O(m^2) time and space.

III. Iteration & Recursion

This program has a combination of recursive and iterative approaches to calculate the determinant. The calcDet() function is recursive which reduces the main task into multiple sub-tasks: calculate the determinant of the sub matrices.

While creating sub matrices, the method Node.reduce() iteratively parses the linked list and takes required nodes to build a new sub matrix.

IV. Enhancement

This lab assignment provides a minimum test case that contains 8 matrices. I add two more with order 7 and 9, in order to examine whether the program is able to calculate any square matrix with any order. The program indeed calculates and prints determinants for 10 matrices.

V. Conclusion

In this lab assignment, I learned any square matrix (2D) can be implemented with a single linked list (1D) having information of its order. With this data structure, the singly linked list can be parsed back to a matrix, also can be used to calculate its determinant as well.

Perhaps next time I would try to implement a matrix with a linked list having right and below pointers, since this implementation is more intuitive and direct way to represent matrix.