Representation of a 2-Dimensional Matrix using Linked List

1 Thought Process

To maintain a 2D linked list, we will require two pointers for a node namely 'right' and 'down', instead of a single pointer as in the case of a normal linked list.

2 Abstract Data Type

```
2DNode      {
      2DNode* right;
      2DNode* left;
      int data;
}
```

3 Input

For the input, it is easier to accept the numbers in the form of a (n x n) matrix and then perform the required operations. The elements are entered according to their position in the 2D linked list. This is similar to insertion in a linked list, but here it is 2-dimensional, as shown in the figure(Figure 1 & 2).

4 Necessary Precautions

In this assignment, the pointers need to be used carefully to avoid memory leaks and dangling pointers which need be optimized for efficient handling of data for large applications.

5 Algorithms

5.1 Sum of Diagonal

1. Start from first node(pointer)
2. Add element to sum.
3. Visit the node to its right(link)
4. Visit the node to its bottom(link)
5. Repeat until pointer is NULL.

5.2 Perimeter

1. Start from first node.
2. Add the element.
3. Visit the node to its right(link).
4. Repeat until pointer is NULL.

5. Add the element.
6. Visit the node to its bottom(link).
7. Repeat until pointer is NULL.
8. Start from first node.
9. Add the element.
10. Visit the node to its bottom(link).
11. Repeat until pointer is NULL.
12. Add the element.
13. Visit the node to its right(link).
14. Repeat until pointer is NULL.

5.3 Transpose

1. Replace every Mat(i, j) for Mat(j, i)
2. Create a new linked list using the transpose

5.4 Output

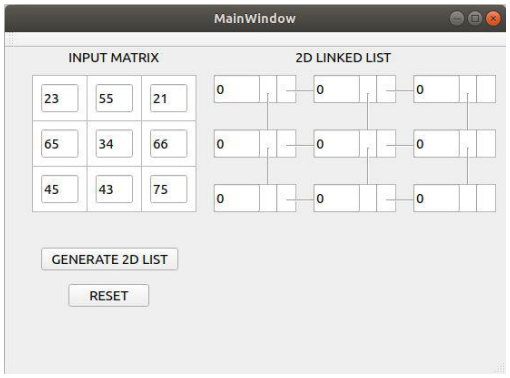

Figure 1: Initialized

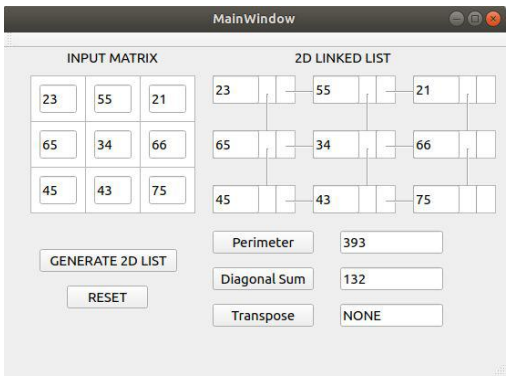

Figure 2: After Creation

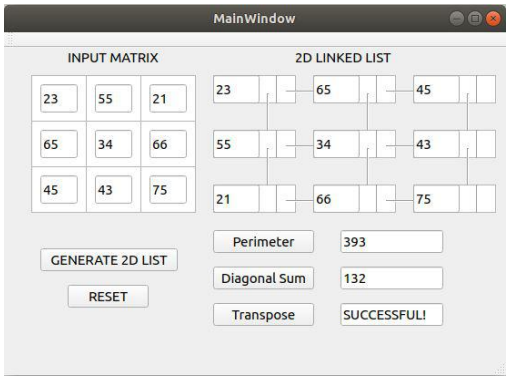

Figure 3: After Transpose