Linked List Matrix

solved by	Senan
Platform	GeeksForGeeks
difficulty	Easy
tags	Array Linked List Matrix
language	C++
solved on	@09/10/2024
link	https://www.geeksforgeeks.org/problems/linked-list-matrix/1
Completion	☑

Intuition

We want to represent a 2D matrix as a linked list structure where each node points to the right (for the next element in the same row) and down (for the element in the next row of the same column). This structure allows us to traverse both rows and columns in constant time, making it a powerful data structure for certain operations.

Approach

1. **genLinkedList Function**:
   - This function generates a single linked list from a given row of the matrix. It takes the row index `i` and the matrix as inputs, then creates a linked list with `right` pointers connecting the nodes in that row.
2. **merge Function**:
   - This function merges two rows by connecting the `down` pointers of nodes in the top row to the corresponding nodes in the bottom row. This creates a vertical link between the rows.
3. **constructLinkedMatrix Function**:
   - This function constructs the entire linked list matrix. It first generates the linked list for the first row. Then, for each subsequent row, it generates a new linked list and merges it with the previous one using the `merge` function.

Complexity

Time Complexity:

- **genLinkedList Function**:
  - This function iterates through each element in a row (size `n`), so it runs in **O(n)** for each row.
- **merge Function**:
  - This function also iterates through each element in a row to connect the rows vertically, so it runs in **O(n)** per row.
- **constructLinkedMatrix Function**:
  - It calls `genLinkedList` and `merge` for each of the `n` rows, so the total time complexity is **O(n^2)**.

Space Complexity:

- **genLinkedList Function**:
  - For each row, we allocate `n` nodes, which contributes to **O(n)** space per row.
- **constructLinkedMatrix Function**:
  - The entire matrix will have `n * n` nodes, so the total space complexity is **O(n^2)**.

Code

```
class Solution {
    // Function to merge two rows of linked lists by setting the 'down' pointers
    void merge(Node* top, Node* bottom) {
        Node* first = top;
        Node* second = bottom;
        while (first && second) {
            first->down = second;
            first = first->right;
            second = second->right;
        }
    }

    // Function to generate a linked list for a given row in the matrix
    Node* genLinkedList(int i, vector<vector<int>>& mat) {
        Node* dummy = new Node(-1);
        Node* marker = dummy;

        int n = mat.size();
        for (int j = 0; j < n; j++) {
            Node* temp = new Node(mat[i][j]);
            dummy->right = temp;
            dummy = temp;
        }
        dummy = marker->right;
        delete marker;
        return dummy;
    }

public:
    // Main function to construct the 2D linked list from a matrix
    Node* constructLinkedMatrix(vector<vector<int>>& mat) {
        int n = mat.size();
        Node* curr = genLinkedList(0, mat);
        Node* head = curr;
        for (int i = 1; i < n; i++) {
            Node* next = genLinkedList(i, mat);
            merge(curr, next);
            curr = next;
        }
        return head;
    }
};
```