Sheryians Coding School

Live Cohort

Day 102

Class 1 - Linked List Advanced (Part 1)

Topic Name: Reverse Nodes in K-Group

Definition:

The problem is to reverse nodes in a linked list in groups of k. If the number of nodes is not a multiple of k, the remaining nodes at the end remain as-is.

Code Example:

```
function reverseKGroup(head, k) {
    // Count nodes to check if k nodes available
    let count = 0;
    let temp = head;
    while (temp && count < k) {
        temp = temp.next;
        count++;
    }

    if (count < k) return head;
```

```
}

// Separate reverse function
function reverseList(head, k) {
    let prev = null;
    let curr = head;
    while (k > 0) {
        let next = curr.next;
        curr.next = prev;
        prev = curr;
        curr = next;
        k--;
```

Use Case

Used in applications where data packets or chunks need to be processed in groups and order needs to be reversed for each group, such as scheduling systems, memory batch updates, or distributed processing tasks.

Interview Q&A

Q: What's the base condition for recursion in this problem?

A: If the remaining nodes are less than k, we return the head as-is.

Q: Why do we need a separate reverse function?

A: It makes the code cleaner and allows reusability of the reverse logic for k nodes.

Q: What is the time complexity?

A: O(N), where N is the number of nodes.

Q: What is the space complexity?

A: O(N/k) due to recursive calls.

Class 2 - Linked List Advanced (Part 2)

Topic Name: Add Two Numbers (Linked List)

Definition:

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

Code Example:

```
function addTwoNumbers(l1, l2) {
    let dummy = new ListNode(0);
    let temp = dummy;
    let carry = 0;

    while (l1 !== null || l2 !== null) {
        let val1 = (l1 !== null) ? l1.val : 0;
        let val2 = (l2 !== null) ? l2.val : 0;

        let data = val1 + val2 + carry;
        carry = Math.floor(data / 10);
```

```
        temp.next = new ListNode(data % 10);
        temp = temp.next;

        if (l1 !== null) l1 = l1.next;
        if (l2 !== null) l2 = l2.next;
    }
```

```
    if (carry > 0) {
        temp.next = new ListNode(carry);
    }

    return dummy.next;
}
```

Use Case:

Removes consecutive duplicate nodes from a sorted linked list.

Interview Q&A

Q: Why is a dummy node used?

A: It simplifies handling the head of the result list and avoids null checks during iteration.

Q: What happens when the sum of digits exceeds 9?

A: A carry is computed and added to the next sum.

Q: How do we handle lists of different lengths?

A: We treat missing nodes as 0 using a ternary operation: (l1 !== null) ? l1.val : 0.

Q: What if there's a leftover carry after the loop?

A: We add a new node with the carry value.