LeetCode 25

https://leetcode.com/problems/reverse-nodes-in-k-group/description/

Yifeng Zeng

Description

25. Reverse Nodes in k-Group

Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.

k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.

You may not alter the values in the nodes, only nodes itself may be changed.

Only constant memory is allowed.

For example,

Given this linked list: 1->2->3->4->5

For k = 2, you should return: 2->1->4->3->5

For k = 3, you should return: 3->2->1->4->5

Idea Report

We can reduce the problem into a series of reverse linked list sub problem. First we get k nodes, reverse them and get the next k nodes, reverse them, and get the next k nodes and so on. So to reverse k nodes, we can have a seperate function (reverseK()) to implement it. But how can we find where it starts for the next k nodes? We can simply return it using the reverseK() funciton. So the input the of the reverseK() function is the previous node of the current k nodes to reverse (reversing k nodes need the previos node). And we store the node that should be the previous node of next k nodes, we return it after reverse the current k nodes. If there is less than k nodes, we simply don't need reverse what's left, and simply return a null to stop the outer loop and stop reversing.

Code

```java
class Solution {
    public ListNode reverseKGroup(ListNode head, int k) {
        if (head == null || head.next == null || k <= 1) {
            return head;
        }

        ListNode dummy = new ListNode(0);
        dummy.next = head;
        head = dummy;

        while (head != null) {
            head = reverseK(head, k);
        }

        return dummy.next;
    }

    private ListNode reverseK(ListNode head, int k) {
        ListNode n1 = head.next; // node 1
        ListNode nk = head; // node k
        for (int i = 0; i < k; i++) {
            nk = nk.next;
            if (nk == null) {
                return null;
            }
        }
        ListNode nkn = nk.next; // node k's next

        ListNode prev = nkn; // previous for reverse
        ListNode cur = n1; // current for reverse
        while (cur != nkn) {
            ListNode temp = cur.next;
            cur.next = prev;
            prev = cur;
            cur = temp;
        }

        head.next = nk;

        return n1;
    }
}
```

Summary

- Reduce a big problem into a sequence of sub problems.