LeetCode 第 25 号问题：K 个一组翻转链表

> *本文首发于公众号「图解面试算法」，是 图解 LeetCode 系列文章之一。*
>
> *同步博客：https://www.algomooc.com*

题目来源于 LeetCode 上第 25 号问题：K 个一组翻转链表。题目难度为 Hard

题目描述

给你一个链表，每 *k* 个节点一组进行翻转，请你返回翻转后的链表。

k 是一个正整数，它的值小于或等于链表的长度。

如果节点总数不是 *k* 的整数倍，那么请将最后剩余的节点保持原有顺序。

示例:

给你这个链表：`1->2->3->4->5`

当 *k* = 2 时，应当返回: `2->1->4->3->5`

当 *k* = 3 时，应当返回: `3->2->1->4->5`

说明:

- 你的算法只能使用常数的额外空间。
- **你不能只是单纯的改变节点内部的值**，而是需要实际进行节点交换。

题目解析

这道算法题可以说是 两两交换链表中的节点 的升级版, 区别就是反转的子链表节点个数变成了自定义.

总体思路还是一样的, 具体可以分为两个处理模块:

1. 根据 *k* 划分若干个需要反转的子链表, 连接反转后的子链表, 最后不足 *k* 的子链表保持不变
    - 设置哨兵 `dummy` 指向 `head` , 为了能找到反转后的链表头结点;
    - 循环 *k* 确定需要 反转子链表 的范围:
        - 循环完成, 确定子链表可以反转

          假设 *A* , *B* 子链表邻接且都可以反转
            - 指针 `start` 指向 *A* 的头结点, `end` 指向 *A* 的尾结点, `nxt` 指向 *B* 的头结点
            - `start -> end` 反转后, `start` 变成了 A 的尾结点, `start -> next = nxt` , 反转后的 *A* 链表指向了 *B*
            - 重置 `start` 为 *B* 的头节点, `end` 为 *B* 的尾结点, `nxt` 为下一个子链表头节点, 反转 *B* 链表
            - 重复上面动作, 知道 循环终止
        - 循环终止, 剩余节点不足 *k* , 终止反转, 返回链表

2. 反转子链表

假设子链表前三个节点为 *a*, *b*, *c* ,设置指针 `pre` , `cur` , `nxt` , 初始化 `pre` 值为 `null` , `cur` 值为 *a* , `nxt` 值为 *a* , 这三个指针位置不变且相邻

终止条件: `cur` 不为空

将当前节点的指针指向上一个节点

1. `cur` 指向 `nxt` (`nxt` 值为 *b*)
2. `cur` 指向 `pre` (`cur` 指向 `null`)
3. `cur` 赋值给 `pre` (`pre` 值为 *a*), `nxt` 赋值给 `cur` (`cur` 值为 *b*)
4. 在执行 步骤 `1` (`nxt` 值为 *c* , 到此相当于 `pre` , `cur` , `nxt` 指向依次向后移动 `1` 位)
5. 重复上面动作

动画描述

 Animation

参考代码

反转链表

```
/**
 * JavaScript 描述
 * 反转区间 [start, end) 的元素, 注意不包含 end
 */
function reverse(start, end) {
    let pre = null,
        cur = start,
        nxt = start;
    while (cur != end) {
        nxt = cur.next;
        // 逐个节点反转
        cur.next = pre;
        // 更新指针位置
        pre = cur;
        cur = nxt;
    }
    // 反转后的头结点, start 移到了最后, end 没有发生改变
    return pre;
};
```

递归解法

```
/**
 * JavaScript 描述
 * 递归
 */
var reverseKGroup = function(head, k) {
    if (head == null) {
        return null;
```

```
    }
    let start, end;
    start = end = head;
    for (let i = 0; i < k; i++) {
        // 不足 k 个，不需要反转
        if (end == null) {
            return head;
        }
        end = end.next;
    }
    // 反转前 k 个元素，不包含 end
    let reverseHead = reverse(start, end);
    // 递归反转后面k个元素 ，并前后连接起来
    start.next = reverseKGroup(end, k);
    return reverseHead;
};
```

迭代解法

```
/**
 * JavaScript 描述
 * 迭代
 */
var reverseKGroup = function(head, k) {
    let dummy = new ListNode(0);
    dummy.next = head;
    let pre, start ,end, nxt;
    pre = start = end = nxt = dummy;
    while (end.next != null) {
        for (let i = 0; i < k && end != null; i++) {
            end = end.next;
        }
        if (end == null) {
            // 不足 k 个，跳出循环
            break;
        }
        start = pre.next;
        nxt = end.next;
        // 反转前 k 个元素，不包含 nxt
        pre.next = reverse(start, nxt);
        // 链接后面的链表
        start.next = nxt;
        // pre , end 重置到 下一个 k 子链表
        pre = start;
        end = pre;
    }
    return dummy.next;
};
```

复杂度分析

- 时间复杂度: **O(nk)** , 最好情况 O(n), 最坏情况 O(n^2)
- 空间复杂度: **O(1)**