LeetCode 第 24 号问题：两两交换链表中的节点

> *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 24 号问题：两两交换链表中的节点。

题目描述

给定一个链表，两两交换其中相邻的节点，并返回交换后的链表。

你不能只是单纯的改变节点内部的值，而是需要实际的进行节点交换。

示例:

```
给定 1->2->3->4, 你应该返回 2->1->4->3.
```

题目解析 - 迭代法

由题目描述可知需要两两交换, 那么以两个为一组子链表交换指针即可, 在设置一个 **哨兵** 指向交换后的子链表 (或者哨兵提前指向子链表的第二个节点,因为第二个节点交换后就成了第一个节点); 然后让哨兵指向下一组子链表,继续交换,直至最后.

设 **哨兵** 为 节点 `prev` , 子链表第一个节点为 `A` , 第二个节点为 `B` , 第三个节点为 `C` , 那么操作流程如下:

- 终止条件 `head == null && head -> next == null`
    1. prev -> B (A -> B -> C)
    2. A - > C
    3. B -> A (prev -> B -> A -> C)
    4. prev -> A
    5. head -> C
    6. 循环以上步骤

动画描述

 Animation1

代码实现

```
/**
 * JavaScript描述
 * 迭代法
 */
var swapPairs = function(head) {
    let dummy = new ListNode(0);
    dummy.next = head;

    let prevNode = dummy;

    while (head !== null && head.next !== null) {
```

```
        // Nodes to be swapped
        let firstNode = head,
            secondNode = head.next;
        // Swapping
        prevNode.next = secondNode;  // 放到交换前后都可以
        firstNode.next = secondNode.next;
        secondNode.next = firstNode;
        // Reinitializing the head and prevNode for next swap
        prevNode = firstNode;
        head = firstNode.next;
    }
    return dummy.next;
};
```

复杂度分析

- 时间复杂度：**O(N)**，其中 *N* 指的是链表的节点数量
- 空间复杂度：**O(1)**

题目解析 - 递归

递归的思路和迭代类似, 都是分组交换. 具体来说这里的递归不是针对一个问题深入进去,而是不断向后推进.

- 每次递归只交换一对节点
- 下一次递归则是交换下一对节点
- 交换完成后返回第二个节点, 因为它是交换后的子链表新头
- 递归完成后返回第一次递归的第二个节点, 这就是新链表的头结点

注意: 不要人肉递归, 更多关注整体逻辑

示例执行大致流程为:

- 终止条件: `(head == null) || (head.next == null)`
    1. 1 -> 2 -> 3 -> 4 (原始链表)
    2. 1 -> 3 -> 4
    3. (2 -> 1) -> 3 -> 4 (第一次递归完成后返回原来的第二个节点, 也就是值为 2 的节点)
    4. 2 -> 1 -> 3 -> null
    5. 2 -> 1 -> (4 -> 3) (第二次递归完成后返回原来的第二个节点, 也就是值为 4 的节点)

动画描述

 Animation2

代码实现

```
/**
 * JavaScript描述
 * 递归法
 */
var swapPairs = function(head) {
    if (head == null || head.next == null) {
        return head;
```

```
    }
     // Nodes to be swapped
    let firstNode = head,
        secondNode = head.next;
     // Swapping
    firstNode.next = swapPairs(secondNode.next);
    secondNode.next = firstNode;

    return secondNode;

};
```

复杂度分析

- 时间复杂度：**O(N)**，其中 *N* 指的是链表的节点数量
- 空间复杂度：**O(N)**, 递归过程使用的堆栈空间