




master ▾

...

leetcode-master / problems / 0024.两两交换链表中的节点.md

 z80160280 Update 0024.两两交换链表中的节点.md History

6 contributors



206 lines (159 sloc) | 7.34 KB

...

PDF下载

代码随想录

刷题

微信群

B站

代码随想录

知识星球

代码随想录

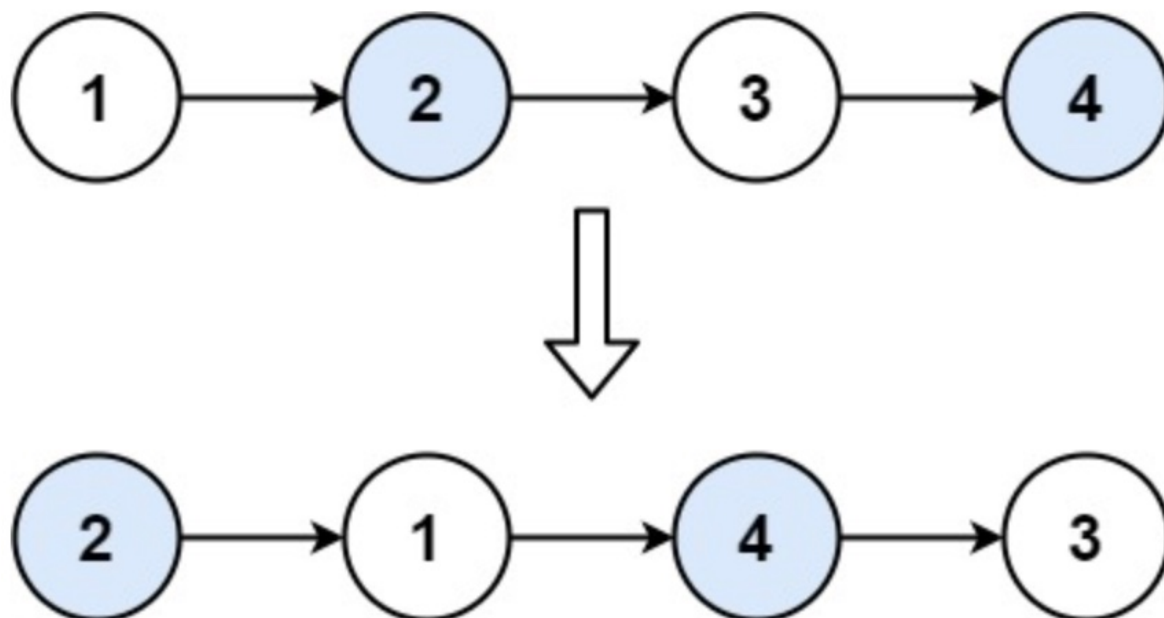
欢迎大家[参与本项目](#)，贡献其他语言版本的代码，拥抱开源，让更多学习算法的小伙伴们收益！

## 24. 两两交换链表中的节点

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 1:



输入: head = [1,2,3,4]

输出: [2,1,4,3]

示例 2:

输入: head = []

输出: []

示例 3:

输入: head = [1]

输出: [1]

## 思路

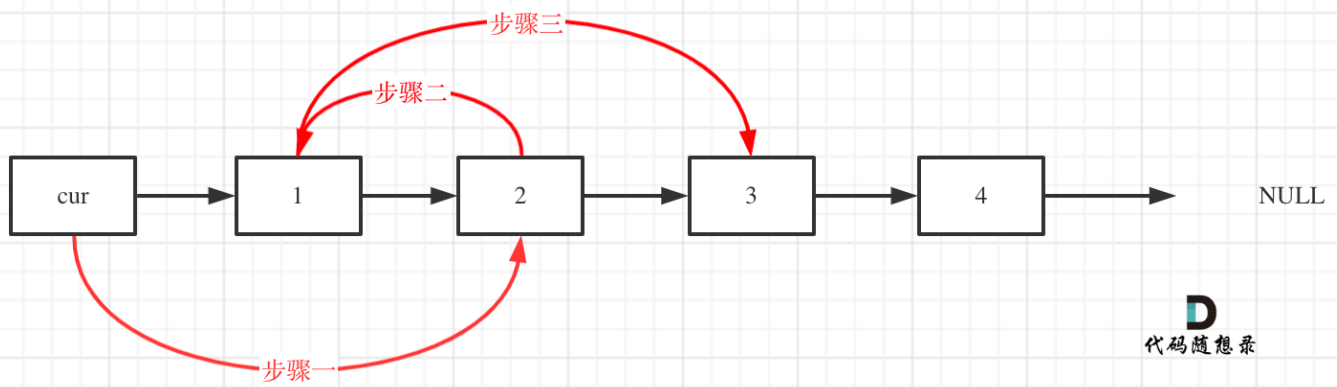
这道题目正常模拟就可以了。

建议使用虚拟头结点，这样会方便很多，要不然每次针对头结点（没有前一个指针指向头结点），还要单独处理。

对虚拟头结点的操作，还不熟悉的话，可以看这篇[链表：听说用虚拟头节点会方便很多？](#)。

接下来就是交换相邻两个元素了，**此时一定要画图，不画图，操作多个指针很容易乱，而且要操作的先后顺序**

初始时，cur指向虚拟头结点，然后进行如下三步：



**D**  
代码随想录

操作之后，链表如下：

## 24.两两交换链表中的节点2

看这个可能就更直观一些了：



**D**  
代码随想录

对应的C++代码实现如下：（注释中详细和如上图中的三步做对应）

```
class Solution {
public:
    ListNode* swapPairs(ListNode* head) {
        ListNode* dummyHead = new ListNode(0); // 设置一个虚拟头结点
        dummyHead->next = head; // 将虚拟头结点指向head，这样方便后面做删除操作
        ListNode* cur = dummyHead;
        while(cur->next != nullptr && cur->next->next != nullptr) {
            ListNode* tmp = cur->next; // 记录临时节点 1
            ListNode* tmp1 = cur->next->next->next; // 记录临时节点 3

            cur->next = cur->next->next; // 步骤一 虚拟头结点指向节点2
            cur->next->next = tmp; // 步骤二 节点2指向节点1
            cur->next->next->next = tmp1; // 步骤三 节点1指向节点3

            cur = cur->next->next; // cur移动两位，准备下一轮交换
        }
        return dummyHead->next;
    }
};
```

本质上就是节点指针来回赋值

- 时间复杂度：  $O(n)$
- 空间复杂度：  $O(1)$

## 拓展

这里还是说一下，大家不必太在意力扣上执行用时，打败多少多少用户，这个统计不准确的。

做题的时候自己能分析出来时间复杂度就可以了，至于力扣上执行用时，大概看一下就行。

上面的代码我第一次提交执行用时8ms，打败6.5%的用户，差点吓到我了。

心想应该没有更好的方法了吧，也就O(n)的时间复杂度，重复提交几次，这样了：

执行结果：**通过** [显示详情](#) >

执行用时：**0 ms**，在所有 C++ 提交中击败了 **100.00%** 的用户

内存消耗：**7.6 MB**，在所有 C++ 提交中击败了 **5.30%** 的用户

炫耀一下：



力扣上的统计如果两份代码是 100ms 和 300ms的耗时，其实是需要注意的。

如果一个是 4ms 一个是 12ms，看上去好像是一个打败了80%，一个打败了20%，其实是没有差别的。只不过是力扣上统计的误差而已。

## 其他语言版本

Java：

```
// 递归版本
class Solution {
    public ListNode swapPairs(ListNode head) {
        // base case 退出提交
        if(head == null || head.next == null) return head;
        // 获取当前节点的下一个节点
        ListNode next = head.next;
        // 进行递归
        ListNode newNode = swapPairs(next.next);
        // 这里进行交换
        next.next = head;
        head.next = newNode;

        return next;
    }
}
```

```
// 虚拟头结点
class Solution {
    public ListNode swapPairs(ListNode head) {

        ListNode dummyNode = new ListNode(0);
```

```

dummyNode.next = head;
ListNode prev = dummyNode;

while (prev.next != null && prev.next.next != null) {
    ListNode temp = head.next.next; // 缓存 next
    prev.next = head.next;          // 将 prev 的 next 改为 head 的 next
    head.next.next = head;          // 将 head.next(prev.next) 的 next, 指向 head
    head.next = temp;               // 将 head 的 next 接上缓存的 temp
    prev = head;                    // 步进1位
    head = head.next;               // 步进1位
}
return dummyNode.next;
}
}

```

Python:

```

class Solution:
    def swapPairs(self, head: ListNode) -> ListNode:
        dummy = ListNode(0) #设置一个虚拟头结点
        dummy.next = head
        cur = dummy
        while cur.next and cur.next.next:
            tmp = cur.next #记录临时节点
            tmp1 = cur.next.next.next #记录临时节点

            cur.next = cur.next.next          #步骤一
            cur.next.next = tmp                #步骤二
            cur.next.next.next = tmp1         #步骤三

            cur = cur.next.next #cur移动两位，准备下一轮交换
        return dummy.next

```

Go:

```

func swapPairs(head *ListNode) *ListNode {
    dummy := &ListNode{
        Next: head,
    }
    //head=list[i]
    //pre=list[i-1]
    pre := dummy
    for head != nil && head.Next != nil {
        pre.Next = head.Next
        next := head.Next.Next
        head.Next.Next = head
        head.Next = next
        //pre=list[(i+2)-1]
        pre = head
        //head=list[(i+2)]
        head = next
    }
    return dummy.Next
}

```

// 递归版本

```
func swapPairs(head *ListNode) *ListNode {  
    if head == nil || head.Next == nil {  
        return head  
    }  
    next := head.Next  
    head.Next = swapPairs(next.Next)  
    next.Next = head  
    return next  
}
```

Javascript:

```
var swapPairs = function (head) {  
    let ret = new ListNode(0, head), temp = ret;  
    while (temp.next && temp.next.next) {  
        let cur = temp.next.next, pre = temp.next;  
        pre.next = cur.next;  
        cur.next = pre;  
        temp.next = cur;  
        temp = pre;  
    }  
    return ret.next;  
};
```

- 
- 作者微信: [程序员Carl](#)
  - B站视频: [代码随想录](#)
  - 知识星球: [代码随想录](#)