双指针

2022年8月11日 14:25

快慢指针

Floyd判圈算法

定义两个指针，一快一慢。

慢指针每次只移动一步，快指针每次移动两步。

初始时，慢指针在位置head，快指针在位置head.next

若在移动过程中，快指针反过来追上慢指针，就说明该链表为环形链表。否则快指针将到达链表底部，该链表不为环形链表

```
class Solution {
public:
    bool hasCycle(ListNode* head) {
        if (head == nullptr || head->next == nullptr) {
            return false;
        }
        ListNode* slow = head;
        ListNode* fast = head->next;
        while (slow != fast) {
            if (fast == nullptr || fast->next == nullptr) {
                return false;
            }
            slow = slow->next;
            fast = fast->next->next;
        }
        return true;
    }
};
```

指向首尾的双指针

653. 两数之和 IV - 输入 BST

难度 简单 422

给定一个二叉搜索树 `root` 和一个目标结果 `k`，如果 BST 中存在两个元素且它们的和等于给定的目标结果，则返回 `true`。

使用中序遍历得到严格递增的序列

使用双指针，分别指向头部和尾部，若头尾的和小于目标值，则头部指针右移，若大于目标值，则尾部指针左移。若等于目标值，返回true。当头尾指针相交时，返回false

```cpp
bool findTarget(TreeNode *root, int k) {
    inorderTraversal(root);
    int left = 0, right = vec.size() - 1;
    while (left < right) {
        if (vec[left] + vec[right] == k) {
            return true;
        }
        if (vec[left] + vec[right] < k) {
            left++;
        } else {
            right--;
        }
    }
    return false;
```

反转链表

206. 反转链表

难度 简单 2710

给你单链表的头节点 `head` ，请你反转链表，并返回反转后的链表。

示例 1：

```
输入：head = [1,2,3,4,5]
输出：[5,4,3,2,1]
```

理解双指针法的逻辑后，很容易写出递归。

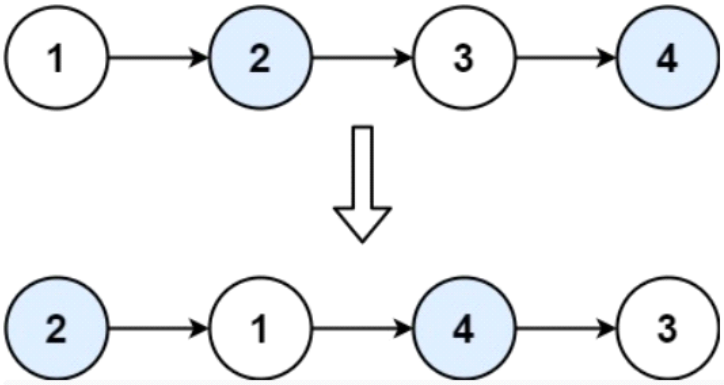
两两交换链表中的节点

24. 两两交换链表中的节点

难度 中等 1527 收藏 分享 切换为英文 接收动态 反馈

给你一个链表，两两交换其中相邻的节点，并返回交换后链表的头节点。你必须在不修改节点内部的值的情况下完成本题（即，只能进行节点交换）。

示例 1：



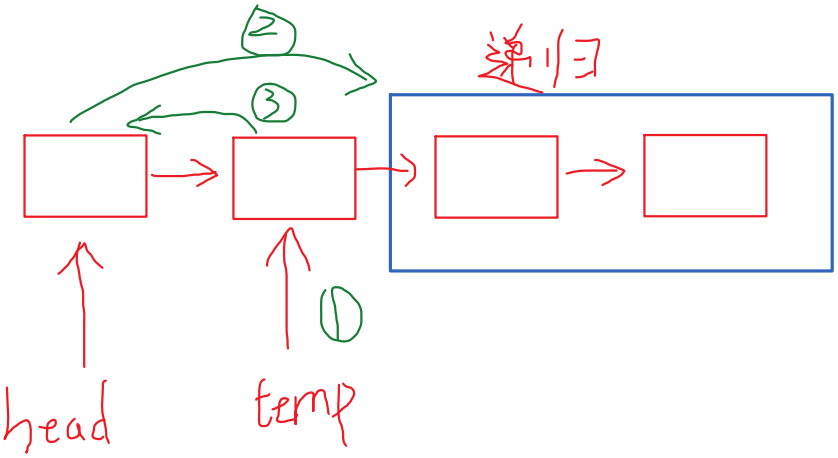
输入： head = [1,2,3,4]
输出： [2,1,4,3]

实际上有递归就有迭代法！尽量每次都把递归和迭代的解法写全

递归法：

在每次递归中需要做的事情是：

1. 用临时变量temp储存head->next的信息
2. 将head->next指向递归返回的值
3. 将temp->next指向head



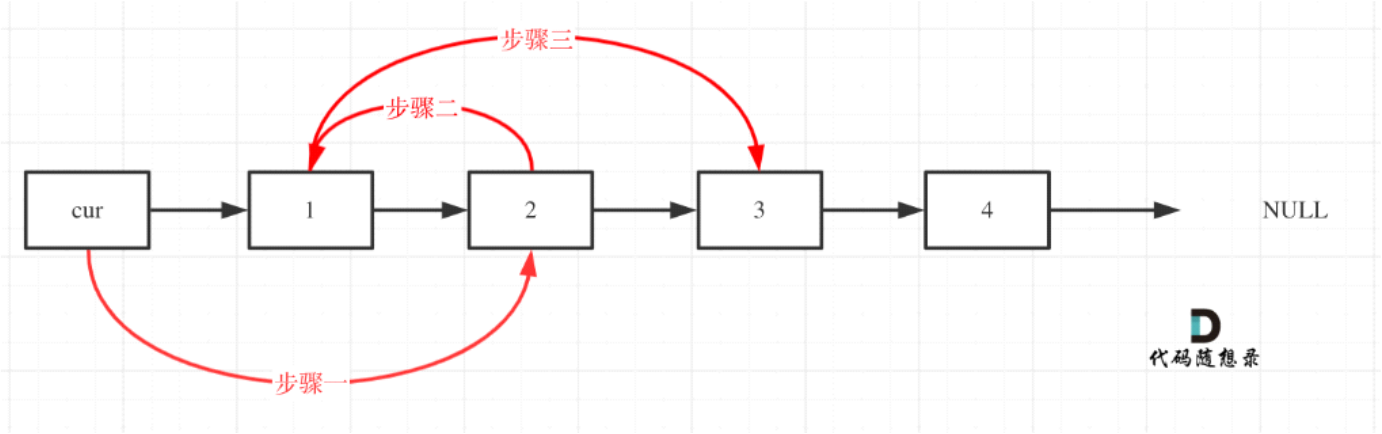
```cpp
class Solution {
public:
    ListNode* swapPairs(ListNode* head) {
        if(head==nullptr||head->next==nullptr){
            return head;
        }
        ListNode* temp=head->next;
        head->next=swapPairs(temp->next);
        temp->next=head;
        return temp;
    }
};
```

迭代法：

我们将需要两两交换的节点称为节点1、节点2

在每次迭代中需要做的是：

1. 将虚拟头节点p指向节点2
2. 将节点2的next指向节点1
3. 将节点1的next指向节点2原先的next
4. 将虚拟头节点向下移动两位（也就是指向节点1）



注意，不能返回head的原因是，原来的头节点head已经被交换了，应在开始交换前，使用q存储虚拟头节点，并返回q的next

```cpp
class Solution {
public:
    ListNode* swapPairs(ListNode* head) {
        ListNode* p=new ListNode(0,head);
        ListNode* q=p;
        while(p->next&&p->next->next){
            ListNode* temp1=p->next;
            ListNode* temp2=p->next->next->next;
            p->next=p->next->next;
            p->next->next=temp1;
            p->next->next->next=temp2;

            p=p->next->next;

        }
        return q->next;
    }
};
```

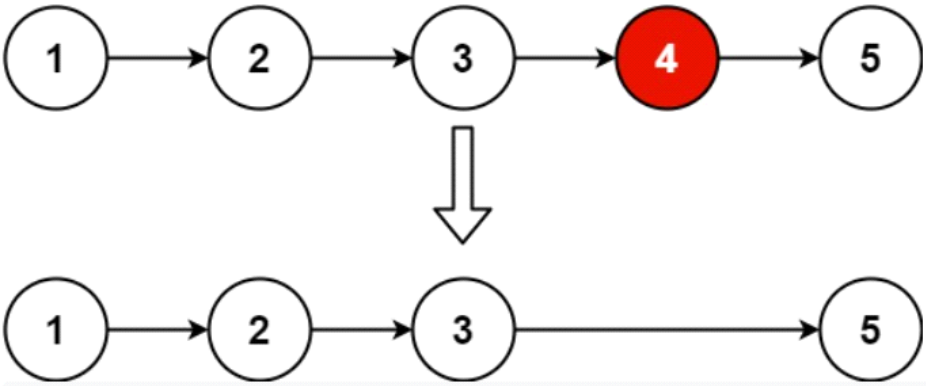
快慢指针

19. 删除链表的倒数第 N 个结点

难度 中等 2177 收藏 分享 切换为英文 接收动态 反馈

给你一个链表，删除链表的倒数第 `n` 个结点，并且返回链表的头结点。

示例 1：



输入： head = [1,2,3,4,5], n = 2
输出： [1,2,3,5]

关键之处在于如何使用一次遍历，完成删除?

快慢指针的运用很灵活，**关键在于如何制定指针的规则**

例如快指针移动两次，慢指针移动一次，那么当快指针到达末尾时，慢指针即指向中间点

那么在删除倒数第n个节点中，若快指针比慢指针先走n次，再一起移动，当快指针到达末尾时，慢指针指向倒数第n个节点！

```
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode* dummyHead=new ListNode(0,head);
        ListNode* slow=dummyHead;
        ListNode* fast=dummyHead;
        while(n--){
            fast=fast->next;
        }
        ListNode* pre;
        while(fast){
            fast=fast->next;
            pre=slow;
            slow=slow->next;
        }
        pre->next=slow->next;
        delete slow;
        return dummyHead->next;
    }
};
```

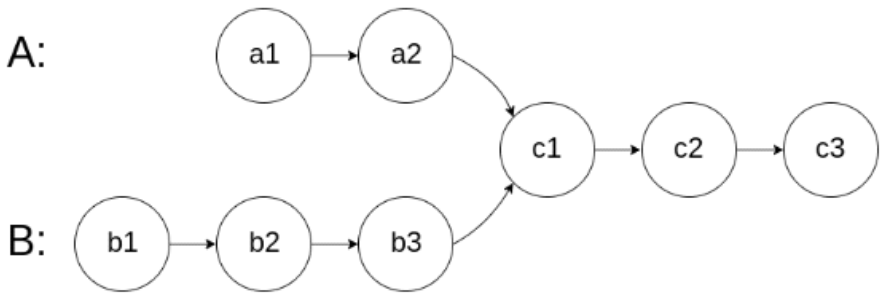
链表相交——双指针

面试题 02.07. 链表相交

难度 简单　253　收藏　分享　切换为英文　接收动态　反馈

给你两个单链表的头节点 `headA` 和 `headB` ，请你找出并返回两个单链表相交的起始节点。如果两个链表没有交点，返回 `null` 。

图示两个链表在节点 `c1` 开始相交：



题目数据 **保证** 整个链式结构中不存在环。

注意，函数返回结果后，链表必须 **保持其原始结构** 。

此题最最最关键之处在于，判断条件是**链表结点相等**而不是节点储存值相等！

将两条链拼在一起，若一条长度为m，另一条长度为n。
一个指针pA指向A链，另一个指针pB指向B链，若当pA遍历完A链后，指向B链头节点，pB遍历完B链后，指向A链头节点。那么若两条链表有交点，则pA一定等于pB

```
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
        if(headA==nullptr||headB==nullptr){
            return nullptr;
        }
        ListNode *pA=headA,*pB=headB;
        while(pA!=pB){
            pA=pA==nullptr?headB:pA->next;
            pB=pB==nullptr?headA:pB->next;
            //当遍历完链A和链B时,pA与pB同时为nullptr
        }
        return pA;
    }
};
```

环形链表并找到环入口——双指针

142. 环形链表 II

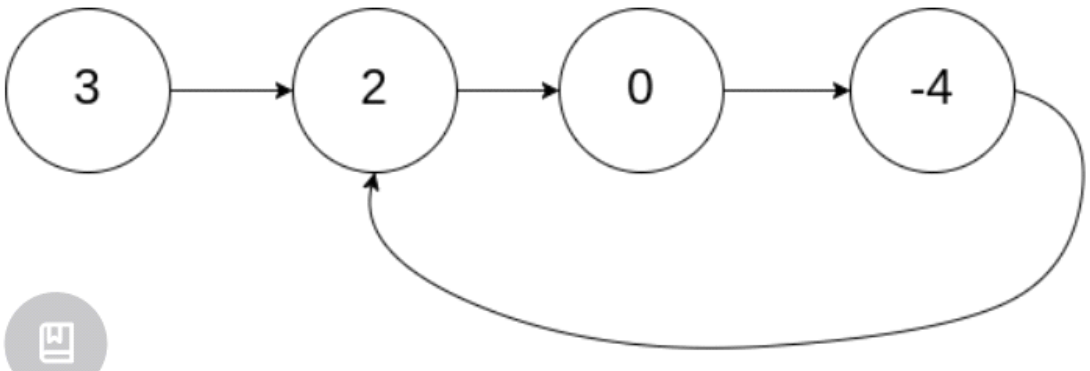
难度 中等 1727

给定一个链表的头节点 `head` ，返回链表开始入环的第一个节点。 *如果链表无环，则返回 `null`*。

如果链表中有某个节点，可以通过连续跟踪 `next` 指针再次到达，则链表中存在环。 为了表示给定链表中的环，评测系统内部使用整数 `pos` 来表示链表尾连接到链表中的位置（**索引从 0 开始**）。如果 `pos` 是 `-1`，则在该链表中没有环。**注意：** `pos` **不作为参数进行传递**，仅仅是为了标识链表的实际情况。

不允许修改 链表。

示例 1：



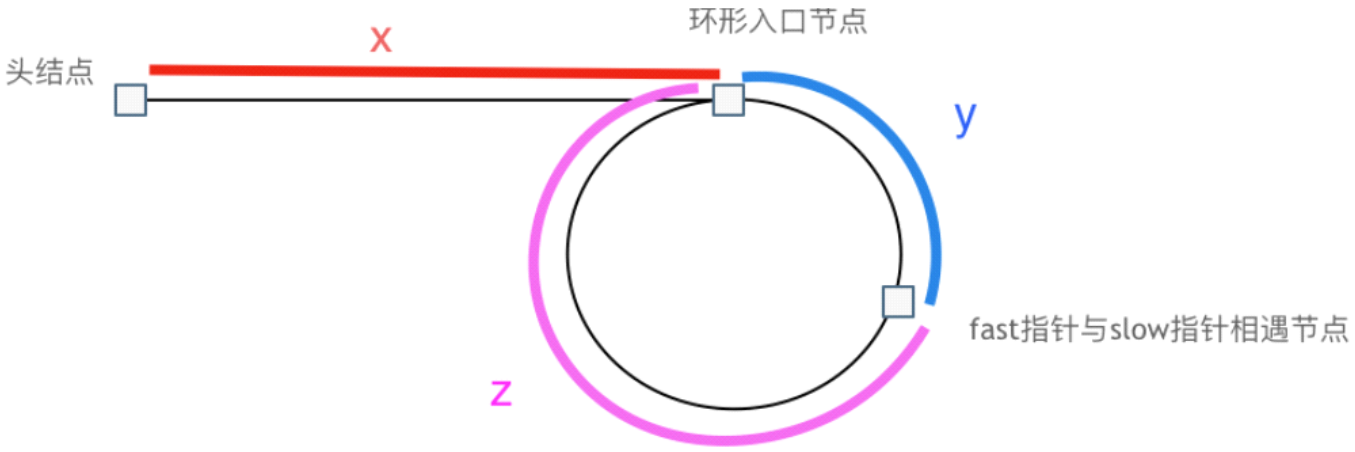
此题是一个比较巧妙的数学问题。在开篇第一题，我们就介绍了如何使用快慢指针来判断这个链表是

否为环形链表，也就是说是否存在“圈”。

并且我们将指针的规则制定为：**慢指针移动一次，快指针移动两次。**

那么在一个圈中，这个规则等价于，快指针以每次循环，移动一次的速度，向着慢指针去追赶（速度是两倍，那么路程也是两倍）

我们画出如下示意图，不难发现数学等式



那么相遇时： slow指针走过的节点数为: `x + y` ， fast指针走过的节点数： `x + y + n (y + z)` ，n为fast指针在环内走了n圈才遇到slow指针， （y+z）为 一圈内节点的个数A。

`(x + y) * 2 = x + y + n (y + z)`

两边消掉一个（x+y）： `x + y = n (y + z)`

因为要找环形的入口，那么要求的是x，因为x表示 头结点到 环形入口节点的的距离。

所以要求x ， 将x单独放在左面： `x = n (y + z) - y` ，

再从n(y+z)中提出一个 （y+z）来，整理公式之后为如下公式： `x = (n - 1) (y + z) + z` 注意这里n一定是大于等于1的，因为 fast指针至少要多走一圈才能相遇slow指针。

由公式 x=(n - 1)(y + z) + z

可知，若我们将slow指针复位到头节点，fast指针每次与slow指针一样，移动一次

那么fast指s内走n-1圈和z的距离，最后和slow指针从头节点开始走x的距离，在环入口中相遇

```cpp
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        ListNode* fast=head;
        ListNode* slow=head;
        while(fast&&fast->next){
            slow=slow->next;
            fast=fast->next->next;
            if(slow==fast){
                slow=head;
                while(fast!=slow){
                    fast=fast->next;
                    slow=slow->next;
                }
                return fast;
            }
        }
        return nullptr;
    }
};
```

替换空格——双指针

剑指 Offer 05. 替换空格

难度 简单 342

请实现一个函数，把字符串 `s` 中的每个空格替换成"%20"。

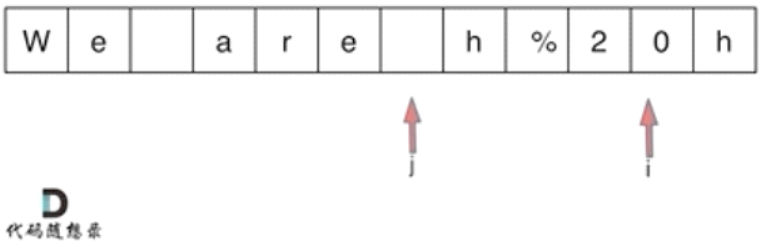
示例 1：

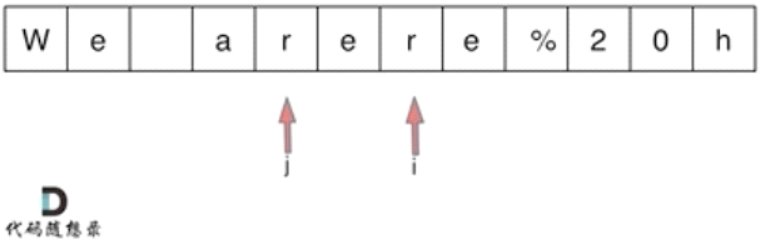
```
输入：s = "We are happy."
输出："We%20are%20happy."
```

限制：

`0 <= s 的长度 <= 10000`

若想要不申请额外空间，并将时间复杂度控制在o(n)
那么我们可以使用双指针
先通过一次遍历，统计空格' '出现的次数sum
将s的size重置为size()+2*sum
再通过一个指针oldSize指向原来s的尾部，另一个指针newSize指向现在s的尾部





上述所需的判断条件即

```
if(s[i]!=' '){
    s[j]=s[i];
}
else{
    s[j]='0';
    s[j-1]='2';
```

```
    if(s[i]!=' '){
        s[j]=s[i];
    }
    else{
        s[j]='0';
        s[j-1]='2';
        s[j-2]='%';
        j-=2;
    }
    string replaceSpace(string s) {
        int sum=0;
        for(auto c:s){
            if(c==' '){
                sum++;
            }
        }
        int oldSize=s.size();
        s.resize(s.size()+2*sum);
        int newSize=s.size();
        for(int i=oldSize-1,j=newSize-1;i<j;i--,j--){
            if(s[i]!=' '){
                s[j]=s[i];
            }
            else{
                s[j]='0';
                s[j-1]='2';
                s[j-2]='%';
                j-=2;
            }
        }
        return s;
    }
};
```