LeetCode 19 删除倒数第n个节点

https://leetcode-cn.com/problems/remove-nth-node-from-end-of-list/

题目要求

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。

尝试使用一趟扫描实现

题目实现

思路

快慢指针法，快指针先移动n+1步，然后快慢指针一起移动，当快指针移动到NULL时，慢指针就指向了要删除节点的前置节点

代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
//快慢指针法：快指针先移动n+1步，然后快慢指针一起移动，这样当快指针移动到空时慢指针就指向要删除节点的前置节点

        ListNode* dummy = new ListNode(0);

        dummy -> next = head;
        ListNode* slow = dummy;
        ListNode* fast = dummy;

        //先移动fast指针
        while(n-- && fast) {
            fast = fast -> next;
        }
        //再多移动一步fast
        fast = fast -> next;

        //一起移动slow指针
        while(fast) {
            fast = fast -> next;
            slow = slow -> next;
```

```
        }

        //此时slow指针指向要删除节点的前置节点
        ListNode* temp = slow -> next;
        slow -> next = slow -> next -> next;
        delete temp;

        return dummy -> next;
    }
};
```

- 时间复杂度：O(n)
- 空间复杂度：O(1)

执行结果

LeetCode 24 两两交换链表节点

https://leetcode-cn.com/problems/swap-nodes-in-pairs/

题目要求

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。

题目实现

思路

穿针引线法实现相邻两个节点的交换，一次循环后节点指针步进两步

代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* swapPairs(ListNode* head) {
//设置虚拟头节点，避免讨论边界
        ListNode* dummy = new ListNode(0);
        dummy -> next = head;

        ListNode* cur = dummy;

        while(cur->next != nullptr && cur->next->next != nullptr){
            //设置临时节点，用于交换
            ListNode* temp = cur->next;
            ListNode* temp1 = cur->next->next->next;
            //交换节点顺序
            cur->next = cur->next->next;
            cur->next->next = temp;
            cur->next->next->next = temp1;
            //步进当前节点
            cur = cur->next->next;
        }

        return dummy->next;
    }
};
```

- 时间复杂度：O(n)
- 空间复杂度：O(1)

执行结果

LeetCode 83 删除排序链表中的重复元素

题目要求

存在一个按升序排列的链表，给你这个链表的头节点 `head` ，请你删除所有重复的元素，使每个元素 **只出现一次**

返回同样按升序排列的结果链表。

题目实现

思路

由于节点数量范围为[0, 300]，所以链表可能为空，即head = NULL，所以需要先判断head是否为空，当为空时直接返回head即可

head不为空时，定义指针cur指向head节点，当cur和cur的下一个节点不为空时循环判断cur节点的值和它下一个节点的值是否相等，如果相等就使cur -> next指向cur -> next -> next，bypass掉cur 的下一个节点

循环直到cur -> next为空终止，最后返回删除重复节点的链表头指针head

代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
```

```
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if (!head) return head;
        ListNode *cur = head;
        while (cur && cur -> next) {
            if (cur -> val == cur -> next -> val) {
                cur -> next = cur -> next -> next;
            }
            else {
                cur = cur -> next;
            }
        }
        return head;
    }
};
```

- 时间复杂度：O(n)
- 空间复杂度：O(1)

执行结果

LeetCode 141 环形链表

题目要求

给定一个链表，指向头节点的指针为head，

判断链表中是否有环

题目实现

思路

利用快慢指针判断是否有环，快指针一次走两步，慢指针一次走一步，如果链表有环，则快慢指针进入环以后就会变成追击问题，快指针一定会和慢指针相遇，否则不存在环

代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {
        if (!head) return false;
        ListNode *fast = head;
        ListNode *slow = head;
        while (fast && fast -> next) {
            fast = fast -> next -> next;
            slow = slow -> next;
            if (fast == slow) return true;
        }
        return false;
    }
};
```

- 时间复杂度：O(n)
- 空间复杂度：O(1)

执行结果

提交结果	执行用时	内存消耗	语言	提交时间	备注
通过	12 ms	8 MB	C++	2021/08/04 10:21	

LeetCode 160 相交链表

题目要求

给定两个单向链表的头节点headA和headB，找出并返回两个单向链表相交的起始节点，如果两个链表没有焦点，则返回null

注意：两个链表相交并不是值相等而是指针相等

题目实现

思路

假设两条链表存在交点，并设A链表的长度为lenA，B链表的长度为lenB，则有lenA + lenB = lenB + lenA

利用双指针p1 p2分别指向A 和 B的头，然后同时向后遍历，

当p1遍历到A的末尾时（p1==NULL）就让它指向B的头继续遍历，同理p2遍历到B的末尾时（p2==NULL）就让它指向A的头继续遍历

那么p1走过的长度就是A+B，p2走过的长度就是B+A，如果两条链表相交则比有p1 == p2的时候

代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
```

```
 * };
 */
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
        if (!headA || !headB) return NULL;
        ListNode *p1 = headA;
        ListNode *p2 = headB;
        while (p1 != p2) {
            if (p1) p1 = p1 -> next;
            else p1 = headB;
            if (p2) p2 = p2 -> next;
            else p2 = headA;
        }
        return p1;
    }
};
```

- 时间复杂度：O(n)
- 空间复杂度：O(1)

执行结果

LeetCode 202 快乐数

题目要求

编写算法判断一个数n是否是快乐数

快乐数定义：

- 对一个正整数，每次将该数替换为它每个位置上数字的平方和
- 然后重复这个过程直到这个数变为1，以可能是**无限循环**，但始终变不到1
- 如果可以变为1，就是快乐数

如果n是快乐数返回true，否则返回false

题目实现

思路

观察题目描述，对输入的数n进行位平方和操作得到下一个数，可以设计一个函数实现位平方和运算，从一个数n1推出下一个数n2，由于可能存在无限循环，所以可以将这个过程想象成链表是否存在环，存在环就会陷入无限循环

类似链表判断环，使用快慢指针实现

代码

```cpp
class Solution {
public:
    int getSum(int n) {
        int sum = 0;
        while (n) {
            sum += (n % 10) * (n % 10);
            n /= 10;
        }
        return sum;
    }

    bool isHappy(int n) {
        int slow = n, fast = n;
        do {
            slow = getSum(slow);
            fast = getSum(fast);
            fast = getSum(fast);
        } while (slow != fast);
        return slow == 1;
    }
};
```

- 时间复杂度：O(n)
- 空间复杂度：O(1)

执行结果

提交结果	执行用时	内存消耗	语言	提交时间
通过	0 ms	5.6 MB	C++	2021/08/04 11:02

LeetCode 203 移除链表元素

题目要求

给定一个链表的头节点head，和一个整数val，删除链表中所有满足Node.val == val的节点，并返回新的头节点

题目实现

思路

建立虚拟头节点dummy，避免对头节点的特殊处理

从dummy开始遍历链表，逐个判断节点的值是否为val，如果相等就bypass掉该节点（删除节点操作）

最后返回dummy -> next

代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
```

```
    ListNode* removeElements(ListNode* head, int val) {
        if (!head) return head;
        ListNode *dummy = new ListNode(0);
        dummy -> next = head;
        ListNode *cur = dummy;
        while (cur -> next) {
            if (cur -> next -> val == val) {
                cur -> next = cur -> next -> next;
            }
            else {
                cur = cur -> next;
            }
        }
        return dummy -> next;
    }
};
```

- 时间复杂度：O(n)
- 空间复杂度：O(1)

执行结果

LeetCode 206 反转链表

题目要求

给定链表头节点head，反转链表，并返回反转后的链表

题目实现

思路

使用双指针，穿针引线法遍历链表

建立虚拟头节点dummy，避免对头节点的特殊处理

代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        if (!head) return head;
        ListNode *pre = nullptr; //前驱节点
        ListNode *cur = head; //当前节点
        ListNode *suc = nullptr; //后继节点
        while (cur) {
            suc = cur -> next;
            cur -> next = pre;
            pre = cur;
            cur = suc;
        }
        return pre;
    }
};
```

- 时间复杂度：O(n)
- 空间复杂度：O(1)

执行结果

LeetCode 234 回文链表

题目要求

判断一个链表是否为回文链表

进阶：使用O(n)的时间复杂度和O(1)的空间复杂度实现

题目实现

思路

该题简单的思路遍历链表，并将链表节点中数值放入一个辅助数组中，然后使用双指针分别从数组头尾向中间遍历，判断回文性，但这样会使空间复杂度为O(n)

要想使空间复杂度下降为O(1)，就要在原链表本身操作，思路如下：

1. 由于单向链表只能从前往后遍历，为了判断回文性，就需要将链表后半部分反转，然后利用双指针分别指向链表的前半段头和后半段头，同时遍历，判断回文性
2. 可以利用快慢指针获取链表的中间节点，fast一次走两步，slow一次走一步，当fast走到链表末尾时，slow正好在中间节点
3. 反转链表也使用双指针实现
4. 判断完成后需要将后半部分再恢复原样

代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
```

```
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
private:
    //对半切分链表
    ListNode* splitList(ListNode* head){
        //设定快慢指针
        ListNode* slow = head;
        ListNode* fast = head;
        //快指针每次走两步，慢指针每次走一步，当快指针走到链表末尾时慢指针就是中点
        while(fast -> next && fast -> next -> next){
            fast = fast -> next -> next;
            slow = slow -> next;
        }
        return slow;
    }

    //反转链表
    ListNode* reverseList(ListNode* head){
        ListNode* pre = nullptr;
        ListNode* cur = head;
        while(cur){
            ListNode* temp = cur -> next;
            cur -> next = pre;
            pre = cur;
            cur = temp;
        }
        return pre;
    }

    //判断回文
    bool compareList(ListNode* head1, ListNode* head2){
        ListNode* cur1 = head1;
        ListNode* cur2 = head2;
        while(cur2){
            if(cur1 -> val != cur2 -> val) return false;
            cur1 = cur1 -> next;
            cur2 = cur2 -> next;
        }
        return true;
    }

public:
    bool isPalindrome(ListNode* head) {
        //1.切分列表
        ListNode* frontEnd = splitList(head);
        //2.反转后半部分
        ListNode* rearStart = reverseList(frontEnd -> next);
        //3.比较前后部分
        bool result = compareList(head, rearStart);
        //4.恢复后半部分
        reverseList(rearStart);
        //5.返回结果
        return result;
    }
```

```
};
```

- 时间复杂度：O(n)，链表整体被遍历两次
- 空间复杂度：O(1)，对链表原地操作

执行结果

LeetCode 237 删除链表中的节点

题目要求

编写函数使其可以删除某个链表中给定的（非末尾）节点

传入函数的唯一参数是要被删除的节点

- 链表至少包含两个节点
- 链表中所有节点的值都是唯一的
- 给定的节点为非末尾节点且一定是链表中的一个有效节点
- 不要从函数中返回任何结果

题目实现

思路

常用的删除链表节点的方法是遍历找到要删除节点的前一个节点，然后将该节点的next指针指向要删除节点的下一个节点（即bypass掉要删除节点），但该题中函数只传入了要删除节点node，由于链表只能从前往后遍历，所以无法获取node的前一个节点，这样一般方法就不适用了

考虑到通过仅有的节点node可以获取到它之后的所有节点，而题目要求删除node，可以抽象的理解为node被其后的节点覆盖，即将node后面一个节点的val赋值给node节点，再把node后面一个节点删掉，这样就从逻辑上实现了node节点删除操作

代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    void deleteNode(ListNode* node) {
        ListNode *suc = node -> next;
        node -> val = suc -> val;
        node -> next = suc -> next;
    }
};
```

- 时间复杂度：O(1)
- 空间复杂度：O(1)

执行结果

OJ 272 邻值查找

题目要求

给定一个长度为 *n*n 的序列 *A*A，*A*A 中的数各不相同。对于 *A*A 中的每一个数 *A****i*Ai，求：

min(1≤j<i) ∣Ai−Aj∣

以及令上式取到最小值的 *j*（记为 Pi）。若最小值点不唯一，则选择**使 Aj 较小的那个**。

题目实现

思路

实在不会。。。

网上资源：https://blog.csdn.net/weixin_43916947/article/details/86723767

慢慢研究ing。。。