10月学习笔记leetcode

# Day 1

[02.01.01 链表基础知识（第 01 ~ 02 天）](https://datawhalechina.github.io/leetcode-notes/#/ch02/02.01/02.01.01-Linked-List-Basic)

**「求线性链表长度」** 的代码如下：

# 获取链表长度

def length(self):

count = 0

cur = self.head

while cur:

count += 1

cur = cur.next

return count

**「链表尾部插入元素」** 的代码如下：

# 链表尾部插入元素

def insertRear(self, val):

node = ListNode(val)

cur = self.head

while cur.next:

cur = cur.next

cur.next = node

while的地方有不同之处，上面是要遍历到最后一个节点后停止。

下面要遍历到最后一个节点时停止

**「链表尾部删除元素」** 的代码如下：

# 链表尾部删除元素

def removeRear(self):

# 如果链表只有一个元素或没有元素，则返回错误

if not self.head.next:

return 'Error'

cur = self.head

while cur.next.next:

cur = cur.next

cur.next = None

此处while要遍历到最后一个节点前的一个节点停止

**「链表中间插入元素」** 的代码（自已用for写了一遍）如下：

def insertInside(self, index, val):

cur = self.head

count = 0

for count=0 in index-1

count += 1

cur = cur.next

# 这一步是检查指针变量是否还在链表，如果不在链表，返回空，not 返回true，执行return

if not cur:

return 'Error'

node = ListNode(val)

node.next = cur.next

cur.next = node

**「链表中间删除元素」** 的代码如下：

# 链表中间删除元素

def removeInside(self, index):

count = 0

cur = self.head

while cur.next and count < index - 1:

count += 1

cur = cur.next

if not cur:

return 'Error'

# 这里为什么不能直接cur.next = cur.next.next？

del\_node = cur.next

cur.next = del\_node.next

# 尝试用for改写

def removeInside(self, index):

count = 0

cur = self.head

for count in index-1

count += 1

cur = cur.next

if not cur

return ‘Error’

cur.next = cur.next.next

# Day2

## [1.](https://datawhalechina.github.io/leetcode-notes/#/ch02/02.01/02.01.02-Exercises?id=_1-0707-%e8%ae%be%e8%ae%a1%e9%93%be%e8%a1%a8)[0707. 设计链表](https://leetcode.cn/problems/design-linked-list/)

class MyLinkedList:

    def \_\_init\_\_(self) -> None:

        self.Head = None

    class MyListNode:

        def \_\_init\_\_(self, val = 0, next = None):

            self.val = val

            self.next = next

    # 获取链表长度

    def length(self):

        count = 0

        cur = self.head

        while cur:

            count += 1

            cur = cur.next

        return count

    def get(self,index: int) -> None: #这里加self的作用？让链表本身有定义  -> None的作用？

        count = 0

        cur = self.head

        while cur and count < index - 1:

            count += 1

            cur = cur.next

        if not cur:

            return '-1'

        return cur.next #错误应为：return current.val

    def addAtHead(self, val) -> None:

        node = self(val) #错误应为：node = self.MyListNode(val)

        # 错误答案 MyLinkedList.Head = node.next

        node.next = self.Head

    def addAtTail(self, val) -> None:

        node = self.MyListNode(val) #缺失判断：如果链表里没有元素，if not self.head: （下一行）self.head = new\_node

        while cur.next:

            cur = cur.next

        cur.next = node

    def addAtIndex(self, index: int, val):

        count = 0 #缺失判断：如果index不在链表里，小于零则排在表头，大于等于链表长度-1则排在表尾

        if index < 0:

            self.addAtHead(val)

        elif index == self.length(self):

            self.addAtTail(val)

        elif index < self.length(self) and index > 0:

            cur = self.Head

            while cur and count < index - 1:

                count += 1

                self.cur = self.cur.next

            node = self.MyListNode(val)

            # 错误答案MyLinkedList.cur = node

            # 错误答案node.next = MyLinkedList.cur.next

            node.next = cur.next

            cur.next = node

    def deleteAtIndex(self, index: int):

        count = 0

        cur = self.Head

        #错误答案while MyLinkedList.cur and count < index - 2:

        while cur and count < index - 1:

            count += 1

            cur = cur.next

        if not cur: #缺失：如果索引index无效？例如index小于0 应改为 if index < 0: \\return \\elif index > self.length(self) return \\elif index == 0 self.Head = self.Head.next \\else \\while count < index - 1 cur = cur.next

            return

        cur = cur.next.next

linkedList = MyLinkedList()

linkedList.addAtHead(1)

# Day3

## [2.](https://datawhalechina.github.io/leetcode-notes/#/ch02/02.01/02.01.02-Exercises?id=_2-0206-%e5%8f%8d%e8%bd%ac%e9%93%be%e8%a1%a8)[0206. 反转链表](https://leetcode.cn/problems/reverse-linked-list/)

迭代法

class Solution:

    def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:

        prev = None

        current = head

        while current is not None:

            next\_node = current.next

            current.next = prev

            prev = current

            current = next\_node

        return prev

递归法

class Solution:

    def reverseList(self, head):

        if head is None or head.next is None:

            return head

        new\_head = self.reverseList(head.next)

        head.next.next = head

        head.next = None

        return new\_head

# Day 4

[**3.**](https://datawhalechina.github.io/leetcode-notes/#/ch02/02.01/02.01.02-Exercises?id=_3-0203-%e7%a7%bb%e9%99%a4%e9%93%be%e8%a1%a8%e5%85%83%e7%b4%a0)[**0203. 移除链表元素**](https://leetcode.cn/problems/remove-linked-list-elements/)

**AI答案**

class Solution:

    def removeElements(self, head: Optional[ListNode], val: int) -> Optional[ListNode]:

       # 创建一个虚拟头节点，简化边界情况的处理

        dummy = ListNode(-1)

        dummy.next = head

        current = dummy

        while current.next:

            if current.next.val == val:

                # 如果下一个节点的值等于 val，将下一个节点跳过

                current.next = current.next.next

            else:

                # 否则继续遍历

                current = current.next

        return dummy.next

## [1.](https://datawhalechina.github.io/leetcode-notes/#/ch02/02.01/02.01.03-Exercises?id=_1-0328-%e5%a5%87%e5%81%b6%e9%93%be%e8%a1%a8)[0328. 奇偶链表](https://leetcode.cn/problems/odd-even-linked-list/)

## 自己的答案

#自己的答案

class Solution:

    def oddEvenList(self, head: Optional[ListNode]) -> Optional[ListNode]:

        if head is None:

            return head

        odd\_list = ListNode(0)

        even\_list = ListNode(0)

        odd = odd\_list

        even = even\_list

        count = true #语法错误

        while head:

            if count

             odd.next = head

             head = head.next #思路错误，没让odd指针移动

            else:

                even.next = head

                head = head.next #思路错误，没让odd指针移动

            count = not cocunt

        even.next = None

        odd.next = even\_list.next

        return odd\_list #如果没有next，则会多一个0

#修改后

class Solution:

    def oddEvenList(self, head: Optional[ListNode]) -> Optional[ListNode]:

        if head is None:

            return head

        odd\_list = ListNode(0)

        even\_list = ListNode(0)

        odd = odd\_list

        even = even\_list

        count = True

        while head:

            if count:

             odd.next = head

             odd = odd.next

            else:

                even.next = head

                even = even.next

            head = head.next

            count = not count

        even.next = None

        odd.next = even\_list.next

        return odd\_list.next

#AI答案

class Solution:

    def oddEvenList(self, head: Optional[ListNode]) -> Optional[ListNode]:

        if not head:

            return head

        # 初始化奇数链表和偶数链表的头节点

        odd\_head = ListNode(0)

        even\_head = ListNode(0)

        # 初始化奇数链表和偶数链表的指针

        odd = odd\_head

        even = even\_head

        # 遍历原始链表并按位置将节点添加到奇数链表或偶数链表

        is\_odd = True

        while head:

            if is\_odd:

                odd.next = head

                odd = odd.next

            else:

                even.next = head

                even = even.next

            head = head.next

            is\_odd = not is\_odd

        # 将奇数链表的末尾连接到偶数链表的头部

        odd.next = even\_head.next

        even.next = None

        return odd\_head.next

## [2.](https://datawhalechina.github.io/leetcode-notes/#/ch02/02.01/02.01.03-Exercises?id=_2-0234-%e5%9b%9e%e6%96%87%e9%93%be%e8%a1%a8)[0234. 回文链表](https://leetcode.cn/problems/palindrome-linked-list/)

#自己的答案

class Solution:

    def isPalindrome(self, head: Optional[ListNode]) -> bool:

        new\_list = ListNode(0) #它的值始终为0，而且没有为其创建节点

        new = new\_list

        head2 = head

        new2 = new\_list

        while new\_list: #循环条件 while new\_list 是不正确的，应该使用 while head 来遍历原始链表。

            new.val = head.val #语法，无法赋值

            new = new.next

            head = head.next

        pre = None

        cur = new #不能写成cur = new.next  一定要写成cur = new\_list.next

        while cur:

            temp = cur.next

            cur.next = pre

            pre = cur

            cur = temp

        while head: #没有定义new2和head2，无法使用val

            if new2.val != head2.val:

                return False

            new2 = new2.next

            head2 = head2.next

        return True

#第一次修改

class Solution:

    def isPalindrome(self, head: Optional[ListNode]) -> bool:

        new\_list = ListNode(0)

        new = new\_list

        cur = head

        while cur:

            new.next = ListNode(cur.val)

            new = new.next

            cur = cur.next

        pre = None

        cur = new\_list.next

        while cur:

            temp = cur.next

            cur.next = pre

            pre = cur

            cur = temp

        while head and pre:

            if head.val != pre.val:

                return False

            head = head.next

            pre = pre.next

        return not head and not pre

#AI的答案

class Solution:

    def isPalindrome(self, head: Optional[ListNode]) -> bool:

        # 创建一个空链表用于存储反转后的链表

        new\_list = ListNode(0)

        new = new\_list

        # 复制原链表到新链表

        current = head

        while current:

            new.next = ListNode(current.val)

            new = new.next

            current = current.next

        # 反转新链表

        reversed\_head = None

        current = new\_list.next

        while current:

            temp = current.next

            current.next = reversed\_head

            reversed\_head = current

            current = temp

        # 比较原链表和反转后的链表

        while head and reversed\_head:

            if head.val != reversed\_head.val:

                return False

            head = head.next

            reversed\_head = reversed\_head.next

        # 如果链表长度不同，说明不是回文链表

        return not head and not reversed\_head

#更优解

class Solution:

    def isPalindrome(self, head: Optional[ListNode]) -> bool:

        # 使用快慢指针的方法移动到链表中点

        cur = slow = fast = head

        prev = None

        while fast is not None and fast.next is not None:

            slow = slow.next

            fast = fast.next.next

            # 反转前半部分链表

            head = cur

            cur = head.next

            head.next = prev

            prev = head

        if fast is not None:

            slow = slow.next

        # 比较前半部分链表（已反转）和后半部分链表

        while slow is not None:

            if slow.val == head.val:

                slow = slow.next

                head = head.next

            else:

                return False

        return True

# Day 5

## [3.](https://datawhalechina.github.io/leetcode-notes/#/ch02/02.01/02.01.03-Exercises?id=_3-0138-%e5%a4%8d%e5%88%b6%e5%b8%a6%e9%9a%8f%e6%9c%ba%e6%8c%87%e9%92%88%e7%9a%84%e9%93%be%e8%a1%a8)[0138. 复制带随机指针的链表](https://leetcode.cn/problems/copy-list-with-random-pointer/)

#自己的答案

class Solution:

    def copyRandomList(self, head: 'Optional[Node]') -> 'Optional[Node]':

        cur = head

        while cur:

            new = Node(cur.val,None)

            rcur = cur.random

            new.random = Node(rcur.val) #仅仅赋值了，没有建立指针

            new = new.next

            cur =cur.next

        return  new.next

#AI的答案

class Solution:

    def copyRandomList(self, head: 'Optional[Node]') -> 'Optional[Node]':

        if not head:

            return None

        # 第一遍遍历：创建新节点并将其插入原链表的每个节点后面

        current = head

        while current:

            new\_node = Node(current.val, None)

            new\_node.next = current.next

            current.next = new\_node

            current = new\_node.next

        # 第二遍遍历：设置新节点的random指针

        current = head

        while current:

            if current.random:

                current.next.random = current.random.next

            current = current.next.next

        # 第三遍遍历：恢复原链表，同时构建深拷贝链表

        current = head

        new\_head = head.next

        new\_current = new\_head

        while current:

            current.next = new\_current.next

            current = current.next

            if current:

                new\_current.next = current.next

            new\_current = new\_current.next

        return new\_head

## [61. 旋转链表](https://leetcode.cn/problems/rotate-list/)

class Solution:

    def rotateRight(self, head: Optional[ListNode], k: int) -> Optional[ListNode]:

        if not head or not head.next or k == 0:

            return head

        # 计算链表的长度

        size = 1

        cur = head

        while cur.next: #注意这里终止的条件是是cur.next 而非cur，即到最后一个停止，否则下面的cur.next将无法执行

            cur = cur.next

            size += 1

        # 将链表的尾部与头部相连，形成环状链表

        cur.next = head

        # 找到新的头节点位置

        new\_head = head

        for \_ in range(size - k % size - 1):

            new\_head = new\_head.next

        # 新的尾节点

        new\_tail = new\_head

        # 断开环状链表

        new\_head = new\_head.next

        new\_tail.next = None

        return new\_head

## [430. 扁平化多级双向链表](https://leetcode.cn/problems/flatten-a-multilevel-doubly-linked-list/)

class Solution:

    def flatten(self, head: 'Optional[Node]') -> 'Optional[Node]':

        curr = head

        while curr:

            if curr.child:

                # 保存下一个节点

                next\_node = curr.next

                # 扁平化子链表

                flattened\_child = self.flatten(curr.child)

                # 连接当前节点和子链表

                curr.next = flattened\_child

                flattened\_child.prev = curr

                curr.child = None

                # 找到子链表的最后一个节点

                while curr.next:

                    curr = curr.next

                # 连接子链表和下一个节点

                curr.next = next\_node

                if next\_node:

                    next\_node.prev = curr

            curr = curr.next

        return head

# Day6

# [02.01.05 链表排序](https://datawhalechina.github.io/leetcode-notes/#/ch02/02.01/02.01.05-Linked-List-Sort?id=_020105-%e9%93%be%e8%a1%a8%e6%8e%92%e5%ba%8f%ef%bc%88%e7%ac%ac-03-%e5%a4%a9%ef%bc%89)