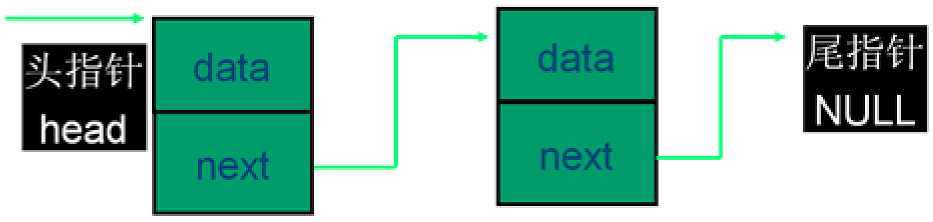
第一课 链表及经典问题

1.链表的访问问题

1.1链表结构如下图所示：



1.1.1 上次课后讨论问题：

1.链表理解

2.入环口问题求证 a=c+(n-1)(b+c)

3.指针与结点问题

4.leetcode和IDE区别

1.2相关题目

1.2.1环形链表

相关链接:https://leetcode-cn.com/problems/linked-list-cycle/

解题重点：

1.快慢指针，问题转化为追及问题

课堂代码：

In [ ]:
```
class Solution:
    def hasCycle(self, head: ListNode) -> bool:
        if not head:
            return False
        slow,fast = head,head
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next
            if slow and slow == fast:return True
        return False
```

1.2.2环形链表II

相关链接:https://leetcode-cn.com/problems/linked-list-cycle-ii

解题重点：

1.根据快慢指针2倍关系特点，推算出链表入环点距离

2.转换等式：2(a+b)=a+n(b+c)+b ==>a=c+(n-1)(b+c)

3.证明a=c+(n-1)(b+c)? ==>n-1?

4.没有办法直接判断a=c距离，需要找关系!

5.定义指针时，如果没办法两指针定义在一个结点上，要注意相遇的位置!

课堂代码:

```
In [ ]: class Solution:
            def detectCycle(self, head: ListNode) -> ListNode:
                if not head:
                    return None
                slow,fast,start = head,head,head
                while fast and fast.next:
                    slow = slow.next
                    fast = fast.next.next
                    if slow and slow == fast:
                        while slow != start:
                            slow = slow.next
                            start = start.next
                        return slow
                return None
```

1.2.3快乐数

相关链接:https://leetcode-cn.com/problems/happy-number/

解题重点:

1.将快乐数问题映射成链表判断是否为环问题

2.然后我们只需要封装一个指定算法函数，就能够当作next方法来使用，就更加形象映射问题

课堂代码:

```
In [ ]: class Solution:
            def get_next(self,number):
                num_sum = 0
                while number > 0:
                    number , a = divmod(number,10)
                    num_sum += a ** 2
                return num_sum
            def isHappy(self, n: int) -> bool:
                slow = n
                fast = self.get_next(n)
                while fast != 1 and fast != slow:
                    slow = self.get_next(slow)
                    fast = self.get_next(self.get_next(fast))
                return fast == 1
```

2.链表的反转问题

2.1相关题目

2.1.1反转链表

相关链接:https://leetcode-cn.com/problems/reverse-linked-list/

解题重点:

1.一定要有三个指针，一个放反转前，一个放翻转后，一个放反转时

课堂代码:

In [ ]:
```
class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        pre = None
        cur = head
        while (cur):
            tem = cur.next
            cur.next = pre
            pre = cur
            cur = tem
        return pre
```

2.1.2反转链表Ⅱ

相关链接:https://leetcode-cn.com/problems/reverse-linked-list-ii

解题重点:

1.与上一题反转链表函数无大差异，但是需要改变反转停止条件，此时为反转K个链表

2.主体函数找到起始反转位置即可，剩下的交给反转函数

3.找到起始反转位置的前一个结点，方便反转后链接

课堂代码:

In [ ]:
```
class Solution:
    def reverse(self,head,k):#反转k个链表
        pre = None
        cur = head
        for _ in range(k):
            tem = cur.next
            cur.next = pre
            pre = cur
            cur = tem
        head.next = cur
        return pre
    def reverseBetween(self, head: ListNode, left: int, right: int) -> ListNode:
        empty = ListNode()
        empty.next = head
        p = empty
        for _ in range(left - 1):
            p = p.next
        p.next = self.reverse(p.next,right - left + 1)
        return empty.next
```

2.1.3 K个一组反转链表

相关链接:https://leetcode-cn.com/problems/reverse-nodes-in-k-group/

解题重点:

1.保证反转函数正确，反转函数需要判断传入的链表是否满足k个结点

2.主体函数通过找到起始反转位置，同时判断下一个位置够不够k个（题目要求剩余的结点不足k个则保留

3.一定需要将复杂问题拆分解决，先解决反转函数，然后反转函数升级改造为要判断K个结点

课堂代码:

In [ ]:
```
class Solution:
    def reverse(self,head,k):#反转K个结点（这个函数的链表输入不一定是有k个链表的）
        pre = head#pre = None
        cur = head
        cnt = 0
        while (pre and cnt < k - 1):#传入的需要被反转的链表是否够k个
            pre = pre.next
            cnt += 1
        if pre == None:return head
        pre = None#出bug的时候:这儿是没有的
        for _ in range(k):
            tem = cur.next
            cur.next = pre
            pre = cur
            cur = tem
        head.next = cur
        return pre
    def reverseKGroup(self, head: ListNode, k: int) -> ListNode:
        empty = ListNode()
        empty.next = head
        pre = empty
        while (1):
            pre.next = self.reverse(pre.next,k)
            cnt = 0
            while pre and cnt < k:#判断接下来的链表结点够不够k个
                cnt += 1
                pre = pre.next
            if pre == None:break
        return empty.next
```

2.1.4 旋转链表

相关链接:https://leetcode-cn.com/problems/rotate-list

解题重点:

1.将问题转化为环形链表重新剪开环（贪吃蛇现象）

2.如何将链表成环，成环操作

3.将右边第K个结点旋转，转化为从头部到第len-k个位置，但是剪开环的时候需要他的前一个位置才能操作

4.需要得到新的链表头的地址，然后再操作剪环

课堂代码:

In [ ]:
```
class Solution:
    def rotateRight(self, head: ListNode, k: int) -> ListNode:
        if not head or not head.next: return head
        pre = head
        length = 1
        while pre.next:
            pre = pre.next
            length += 1
        k = k % length
        pre.next = head#链表成环了
        for _ in range(length - k - 1):#保证拿到第len-k的结点的地址，
            head = head.next
        new_head = head.next#保证newhead地址，我们先拿到手
        head.next = None#断开结点
        return new_head
```

2.1.5 两两交换链表中的节点

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

解题重点:

1.K个链表反转的特例，将K=2即可

2.直接两两反转也是很简单的，保证反转时有三个指针标记地址即可，切记画图!!!

课堂代码：

```
In [ ]: class Solution:
            def swapPairs(self, head: ListNode) -> ListNode:
                if not head:return None
                empty = ListNode()
                empty.next = head
                T = empty
                while T.next and T.next.next:
                    node1 = T.next
                    node2 = T.next.next
                    T.next = node2
                    node1.next = node2.next
                    node2.next = node1
                    T = node1
                return empty.next
```

3.链表的节点删除问题

3.1相关题目

3.1.1 删除链表的倒数第N个结点

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

解题重点：

1.删除结点需要定位到被删除结点的前一个结点

2.倒数第N个结点可转化为顺数第Length-N个结点

课堂代码：

```
In [ ]: class Solution:
            def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:
                empty = ListNode()
                empty.next = head
                p,q = empty,head
                while n:
                    n -= 1
                    q = q.next
                while q:
                    q = q.next
                    p = p.next
                p.next = p.next.next
                return empty.next
```

3.1.2 删除排序链表中的重复元素

相关链接:https://leetcode-cn.com/problems/remove-duplicates-from-sorted-list/

解题重点：

1.注意是排序链表，根据排序链表可知重复元素一定近邻

2.头节点肯定是动不了，不用虚拟头节点

课堂代码：

```
In [ ]: class Solution:
            def deleteDuplicates(self, head: ListNode) -> ListNode:
```

```
        if not head:return None
        fast,slow = head,head
        while fast:
            if fast.val != slow.val:
                slow.next = fast
                slow = slow.next
            fast = fast.next
        slow.next = None
        return head
```

3.1.3 删除排序链表中的重复元素Ⅱ

相关链接:https://leetcode-cn.com/problems/remove-duplicates-from-sorted-list-ii

解题重点:

1.删除有重复的元素，则head结点有可能也会被删除，需要加虚拟头节点

2.指针临近的时候不用跳越(不需要slow.next = fast.next)

课堂代码:

In [ ]:

```
class Solution:
    def deleteDuplicates(self, head: ListNode) -> ListNode:
        empty_head = ListNode()
        empty_head.next = head
        pre,cur = empty_head,head
        while cur:
            while cur.next and cur.val == cur.next.val:
                cur = cur.next
            if pre.next == cur:
                pre = pre.next
            else:
                pre.next = cur.next
            cur = cur.next
        return empty_head.next
```