链表review

2017-04-11 | 数据结构和算法 | | | 95.5k | 20 | 19

> 链表（Linked list）是一种常见的基础数据结构，是一种线性表，但是并不会按线性的顺序存储数据，而是在每一个节点里存到下一个节点的指针(Pointer)。——维基百科

链表和数组都是一种线性结构

- 数组是一段连续的存储空间
- 链表空间不一定保证连续，为临时分配的。

链表的分类

按连接方向分类

- 单链表
- 双链表

按照有环无环分类

- 普通链表
- 循环链表

链表问题代码实现的关键点

1.链表调整函数的返回值类型，根据要求往往是节点类型

2.处理链表过程中，先采用画图的方式理清逻辑

3.链表问题对于边界条件讨论要求严格

链表插入和删除的注意事项：

1.特殊处理链表为空，或者链表长度为1 的情况

2.注意插入时的操作过程

3.注意删除操作的调整过程

注意点：头尾节点及空节点需特殊考虑

双链表的插入与删除和单链表类似，但是需要额外考虑previous指针的指向。

单链表分翻转操作

当链表为空或者长度为1时，特殊处理

注意

1.大量链表问题可以使用额外数据结构来简化调整过程

2.但链表问题最优解往往是不使用额外数据结构的方法

环形链表插值

有一个整数val，如何在节点值有序的环形链表中插入一个节点值为val的节点，并且保证这个环形单链表依然有序。

给定链表的信息，及元素的值A及对应的nxt指向的元素编号同时给定val，请构造出这个环形链表，并插入该值。

测试样例：

[1,3,4,5,7],[1,2,3,4,0],2

返回：{1,2,3,4,5,7}

```
# -*- coding:utf-8 -*-

class ListNode:
     def __init__(self, x):
         self.val = x
         self.next = None
class InsertValue:
    def insert(self, A, nxt, val):
        #先构造出要插入的节点
        nodeT=ListNode(val)
        if not A:
            nodeT.next = NodeT
            return nodeT
```

```
        #接下来要构造环形单链表
        head=ListNode(A[0])
        r=head
        for i in range(len(A)-1):
            r.next=ListNode(A[nxt[i]])
            r=r.next
        #先边界处理
        if head.val>val:
            #r.next=nodeT
            nodeT.next=head
            head=nodeT
            return head
        if r.val<val:
            r.next=nodeT
            nodeT.next=None
            return head

        #中间插值
        pre,now=head,head.next
        while True:
            if pre.val<=val and now.val>=val:
                break
            else:
                pre=pre.next
                now=now.next
        pre.next=nodeT
        nodeT.next=now
        return head
        # write code here
```

访问单个节点的删除

实现一个算法，删除单向链表中间的某个结点，假定你只能访问该结点。

给定带删除的节点，请执行删除操作，若该节点为尾节点，返回false，否则返回true

```
# -*- coding:utf-8 -*-
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

class Remove:
    def removeNode(self, pNode):
        # write code here
```

```
        self.pNode = pNode
        if self.pNode.next == None:
            return False
        else:
            self.pNode = self.pNode.next
            return True
```

注意：

该删除方式并不是删除了该删除的节点，而是进行了值的拷贝。

1.结构复制且拷贝操作受限时，不可行

2.在工程上影响外部依赖

链表的分化

对于一个链表，我们需要用一个特定阈值完成对它的分化，使得小于等于这个值的结点移到前面，大于该值的结点在后面，同时保证两类结点内部的位置关系不变。

给定一个链表的头结点head，同时给定阈值val，请返回一个链表，使小于等于它的结点在前，大于等于它的在后，保证结点值不重复。

测试样例：

{1,4,2,5},3

{1,2,4,5}

解决思路：

简单做法：

1.将链表的所有节点放入数组中，然后将数组进行快排划分的调整过程。

2.然后将数组中的节点依次重新串连。

最优解：

遍历链表过程中，分成3个小链表，分别为大于、等于、小于，最后连接起来。

```
# -*- coding:utf-8 -*-
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

```

```
class Divide:
    def listDivide(self, head, val):
        llst, rlst = [], []
        cur = head
        if head == None: return None
        count = 0
        while cur != None:
            count += 1
            tmp = ListNode(cur.val)
            if tmp.val <= val:
                llst.append(tmp)
            else:
                rlst.append(tmp)
            cur = cur.next
        llst += rlst
        for s in xrange(count - 1):
            llst[s].next = llst[s + 1]
        return llst[0]
```

打印两个链表的公共值

现有两个升序链表，且链表中均无重复元素。请设计一个高效的算法，打印两个链表的公共值部分。

给定两个链表的头指针headA和headB，请返回一个vector，元素为两个链表的公共部分。请保证返回数组的升序。两个链表的元素个数均小于等于500。保证一定有公共值

测试样例：
{1,2,3,4,5,6,7},{2,4,6,8,10}
返回：[2.4.6]

解决思路：
如果两个链表有任何一个为空，直接返回即可。
如果都不为空，则从两个链表的头节点开始遍历。接下来：
如果list1 当前节点的值小于list2当前节点的值，则继续遍历list1的下一个节点。
如果list2 当前节点的值小于list1当前节点的值，则继续遍历list2的下一个节点。
如果list1 和 lsit2，当前节点值相等，则打印当前节点值，然后都向下移动。
list 1 或者list2 当前节点值中有一个为空时，结束整个过程。

```
# -*- coding:utf-8 -*-
class ListNode:
```

```
    def __init__(self, x):
        self.val = x
        self.next = None

class Common:
    def findCommonParts(self, headA, headB):
        # write code here
        #如果两个链表有任何一个为空，直接返回即可。
        if headA == None or headB == None:
            return None
        listc = []
        #如果两个链表都不为空，则从头开始遍历
        while headA != None and headB != None:
            #如果listA 当前节点的值小于listB当前节点的值，则继续遍历listB的下一个节点。
            if headA.val < headB.val:
                headA = headA.next
            #如果listA 当前节点的值小于listB当前节点的值，则继续遍历listB的下一个节点。
            elif headA.val > headB.val:
                headB = headB.next
            #如果listA 和 lsitB，当前节点值相等，则打印当前节点值，然后都向下移动。
            else:
                listc.append(headA.val)
                headA = headA.next
                headB = headB.next
        return listc
```

链表的k逆序

有一个单链表，请设计一个算法，使得每K个节点之间逆序，如果最后不够K个节点一组，则不调整最后几个节点。例如链表1->2->3->4->5->6->7->8->null，K=3这个例子。调整后为，3->2->1->6->5->4->7->8->null。因为K==3，所以每三个节点之间逆序，但其中的7，8不调整，因为只有两个节点不够一组。

给定一个单链表的头指针head,同时给定K值，返回逆序后的链表的头指针。

解决思路：

如果链表为空，或长度为1，或k < 2，链表不用进行调整

方法1： 时间复杂度为O(n),额外空间复杂度为O(k).

利用栈来处理，将k个元素依次进栈，然后依次从栈中弹出。

下一组元素继续按上面过程处理。

如果最后一组元素不足k个，则不调整。

方法2： 时间复杂度为O(n),额外空间复杂度为O(1).

1.基本过程与方法一类似

2.依然是每收集k个元素就做逆序调整

3.需要更多的边界讨论及代码实现技巧

```python
# -*- coding:utf-8 -*-
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

class KInverse:
    def reverse(self, head, count):
        pre, nxt = None, None
        while count:
            nxt = head.next
            head.next = pre
            pre = head
            head = nxt
            count -= 1
        return pre

    def inverse(self, head, k):
        # write code here
        count = 0
        cur = head
        tmp = head
        root, tail = None, head
        res = head
        while cur:
            count += 1
            if count == k:
                cur = cur.next
                if not root:
                    res = self.reverse(tmp, k)
                    root = res
                else:
                    root = self.reverse(tmp, k)
                    tail.next = root
                    tail = tmp

                tmp.next = cur
                tmp = tmp.next
                count = 0
                continue
            else:
                cur = cur.next
        return res
```

链表指定值清除

现在有一个单链表。链表中每个节点保存一个整数，再给定一个值val，把所有等于val的节点删掉。

给定一个单链表的头结点head，同时给定一个值val，请返回清除后的链表的头结点，保证链表中有不等于该值的其它值。请保证其他元素的相对顺序。

测试样例：
{1,2,3,4,3,2,1},2
{1,3,4,3,1}

```
-*- coding:utf-8 -*-
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

class ClearValue:
    def clear(self, head, val):
        #遍历节点，当节点值与val值相等时，修改指针
        #如果节点为空，直接返回
        if head == None:
            return head
        pre ,curl = ListNode(-1) ,head
        nhead = pre
        pre.next = curl
        while curl != None:
            if curl.val == val:
                pre.next = curl.next
            else:
                pre = curl
            curl = curl.next
        return nhead.next
```

链表的回文结构

请编写一个函数，检查链表是否为回文。

给定一个链表ListNode* pHead，请返回一个bool，代表链表是否为回文。

测试样例：

{1,2,3,2,1}

返回：true

{1,2,3,2,3}

返回：false

```
# -*- coding:utf-8 -*-
class ListNode:
    def __init__(self, x):
         self.val = x
         self.next = None

class Palindrome:
    def isPalindrome(self, pHead):
        #如果pHead为空，直接返回false
        if pHead == None:
            return False
        #用两个指针分别指向头和尾，从两边往中间扫，如果从始自终，所指字符都相等，则返回true 否则，
        node = pHead
        temp = []
        while node != None:
            temp.append(node.val)
            node = node.next
        front = 0
        back = len(temp) - 1
        while front < back:
            if temp[front] == temp[back]:
                front += 1
                back -= 1
            else:
                return False
        return True
```

方法二：利用栈，先入栈，然后依次pop,与链表值进行比较

```
# -*- coding:utf-8 -*-
class ListNode:
    def __init__(self, x):
         self.val = x
         self.next = None
class Palindrome:
    def isPalindrome(self, pHead):
        p = pHead
```

```
        stack = []
        while p != None:
            stack.append(p.val)
            p = p.next
        while stack:
            if stack.pop() == pHead.val:
                pHead = pHead.next
            else:
                return False
        return True
```

链表判环

如何判断一个单链表是否有环？有环的话返回进入环的第一个节点的值，无环的话返回-1。如果链表的长度为N，请做到时间复杂度O(N)，额外空间复杂度O(1)。

给定一个单链表的头结点head（注意另一个参数adjust为加密后的数据调整参数，方便数据设置，与本题求解无关)，请返回所求值。

```
# -*- coding:utf-8 -*-
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

class ChkLoop:
    def chkLoop(self, head, adjust):
        if not head:  #链表为空
            return -1
        slowp,fastp=head,head  #p慢指针 q快指针
        while fastp.next and fastp.next.next:
            slowp=slowp.next
            fastp=fastp.next.next
            if fastp==slowp:
                break
        if not fastp.next or not fastp.next.next:
            return -1

        fastp=head
        while fastp!=slowp:
            fastp=fastp.next
            slowp=slowp.next
        return fastp.val
```

无环单链表判相交

现在有两个无环单链表，若两个链表的长度分别为m和n，请设计一个时间复杂度为O(n + m)，额外空间复杂度为O(1)的算法，判断这两个链表是否相交。

给定两个链表的头结点headA和headB，请返回一个bool值，代表这两个链表是否相交。保证两个链表长度小于等于500。
方法一：哈希表实现，一个链表加入Hash表中，遍历另一个链表，一旦遇到某个节点在hash表中有记录，则说明有交点，且为第一个交点

方法二：
假设链表1的长度为m, 链表2的长度为n,且 m > n, 则 先遍历链表1走m -n步，接着俩个链表同时遍历，遇到相等的节点极为相交点。

```
# -*- coding:utf-8 -*-
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

class CheckIntersect:
    def chkIntersect(self, headA, headB):
        # 差值处理
        if not headA or not headB:
            return False
        lenA, lenB = 0, 0
        pA, pB = headA, headB
        while pA :
            lenA += 1
            pA = pA.next
        while pB:
            lenB += 1
            pB = pB.next
        pA, pB = headA, headB
        if lenA > lenB:
            distance = lenA - lenB
            for i in range(distance): pA = pA.next
        else:
            distance = lenB - lenA
            for i in range(distance): pB = pB.next

        while pA:
            if pA == pB: return True
```

```
            pA , pB = pA.next, pB.next
        return False
```

有环单链表相交判断

如何判断两个有环单链表是否相交？相交的话返回第一个相交的节点，不想交的话返回空。如果两个链表长度分别为N和M，请做到时间复杂度O(N+M)，额外空间复杂度O(1)。

给定两个链表的头结点head1和head2(注意，另外两个参数adjust0和adjust1用于调整数据,与本题求解无关)。请返回一个bool值代表它们是否相交。

```
# -*- coding:utf-8 -*-
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

class ChkIntersection:
    def chkInter(self, head1, head2,adjust0, adjust1):
        loop1 = self.chkLoop(head1)
        loop2 = self.chkLoop(head2)
        if loop1 == loop2:
            return True
        p = loop1.next
        while p != loop1:
            if p == loop2:
                return True
            p = p.next
        return False

    def chkLoop(self, head):
        p1, p2 = head, head
        while True:
            p1 = p1.next
            p2 = p2.next.next
            if p1 == p2:
                return p1
```

单链表相交判断

给定两个单链表的头节点head1和head2，如何判断两个链表是否相交？相交的话返回true，不想交的话返回false。

给定两个链表的头结点head1和head2(注意，另外两个参数adjust0和adjust1用于调整数据,与本题求解无关)。请返回一个bool值代表它们是否相交。

解决思路：

1.利用案例九的方法找到两个链表各自的入环节点。
假设：head1链表的入环节点为node1
head2链表的入环节点为node2
2.如果node1 和 node2, 一个为空，另一个不为空，则两个链表不可能相交
3.如果node1 和 node2 都等于空， 则说明两个链表都无环，则用上面“无环单链表判相交”解决方法解决
4.如果node1 和 node2 都不为空，说明两个链表都有环，则用上面“有环单链表相交判断”的方法进行判断。

代码实现：

```
# -*- coding:utf-8 -*-
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class ChkIntersection:
    def chkInter(self, head1, head2, adjust0, adjust1):
        # write code here
        if head1 == None or head2 == None:
            return False
        #找到两个链表各自的入环节点
        p1,p2 = self.chkLooper(head1),self.chkLooper(head2)
        #如果node1 和 node2 都等于空， 则说明两个链表都无环，则用"无环单链表判相交"解决方法
        if p1 == None and p2 == None:
            n, m = 0,0
            node1 = head1
            node2 = head2
            while node1 != None:
                node1 = node1.next
                n += 1
            while node2 != None:
                node2 = node2.next
                m += 1
            if m >= n:
                long = head2
                short = head1
                k = m - n
            else:
```

```
                long = head1
                short = head2
                k = n - m
            for i in xrange(0,k):
                long = long.next
            while long != short and long != None:
                long = long.next
                short = short.next
            if long == None:
                return False
            else:
                return True
        #如果node1 和 node2, 一个为空，另一个不为空，则两个链表不可能相交
        elif p1 == None or p2 == None:
            return False
        #如果node1 和 node2 都不为空，说明两个链表都有环，则用“有环单链表相交判断”的方法进行判断。
        else:
            if p1 == p2:
                return True
            p = p1.next
            while p != p1:
                if p == p2:
                    return True
                p = p.next
            return False

    #找到入环节点
    def chkLooper(self,head):
        p1,p2 = head,head
        while 1:
            p1 = p1.next
            if p2.next == None:
                return None
            p2 = p2.next.next
            if p1 == None or p2 == None:
                return None
            if p1 == p2:
                return p1
```

思考题

复杂链表的复制

输入一个复杂链表（每个节点中有节点值，以及两个指针，一个指向下一个节点，另一个特殊指针指向任意一个节点）。

扫一扫，关注我的微信公众号！

链表

第十二周：时间黑洞

二叉树 review