youngyangyang04 Update History

5 contributors

274 lines (233 sloc) 8.01 KB

面试题 02.07. 链表相交

题目链接：https://leetcode-cn.com/problems/intersection-of-two-linked-lists-lcci/

给定两个（单向）链表，判定它们是否相交并返回交点。请注意相交的定义基于节点的引用，而不是基于节点的值。换句话说，如果一个链表的第k个节点与另一个链表的第j个节点是同一节点（引用完全相同），则这两个链表相交。

示例 1：

输入：listA = [4,1,8,4,5], listB = [5,0,1,8,4,5]

输出：Reference of the node with value = 8

输入解释：相交节点的值为 8 （注意，如果两个列表相交则不能为 0）。从各自的表头开始算起，链表 A 为 [4,1,8,4,5]，链表 B 为 [5,0,1,8,4,5]。在 A 中，相交节点前有 2 个节点；在 B 中，相交节点前有 3 个节点。

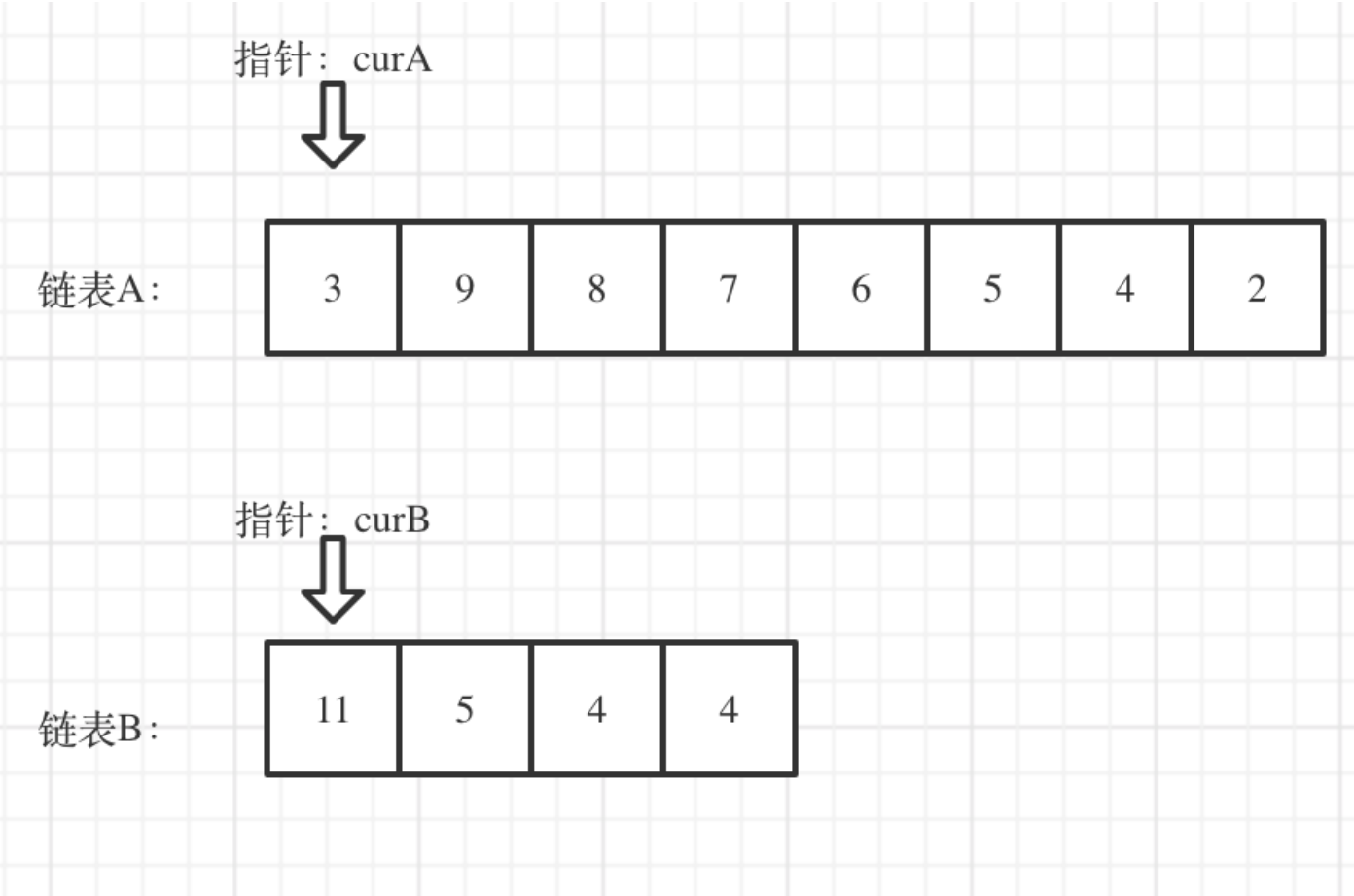
思路

本来很简洁明了的一道题，让题目描述搞的云里雾里的。

简单来说，就是求两个链表交点节点的**指针**。 这里同学们要注意，交点不是数值相等，而是指针相等。

为了方便举例，假设节点元素数值相等，则节点指针相等。 ???

看如下两个链表，目前curA指向链表A的头结点，curB指向链表B的头结点：



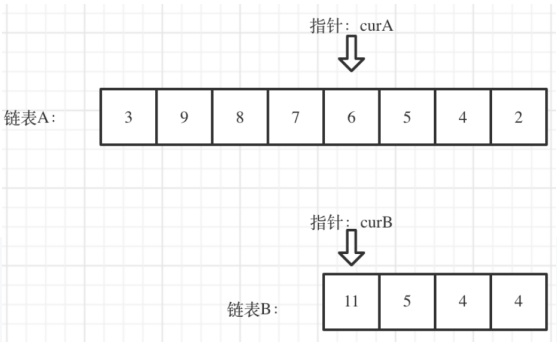
我们求出两个链表的长度，并求出两个链表长度的差值，然后让curA移动到，和curB 末尾对齐的位置，如图：

面试题02.07.链表相交_2

此时我们就可以比较curA和curB是否相同，如果不相同，同时向后移动curA和curB，如果遇到curA == curB，则找到焦点。

否则循环退出返回空指针。

C++代码如下：




```
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
        ListNode* curA = headA;
        ListNode* curB = headB;
        int lenA = 0, lenB = 0;
        while (curA != NULL) { // 求链表A的长度
            lenA++;
            curA = curA->next;
        }
        while (curB != NULL) { // 求链表B的长度
            lenB++;
            curB = curB->next;
        }
        curA = headA;
        curB = headB;
        // 让curA为最长链表的头，lenA为其长度
```

```
        if (lenB > lenA) {
            swap (lenA, lenB);
            swap (curA, curB);
        }
        // 求长度差
        int gap = lenA - lenB;
        // 让curA和curB在同一起点上（末尾位置对齐）
        while (gap--) {
            curA = curA->next;
        }
        // 遍历curA 和 curB，遇到相同则直接返回
        while (curA != NULL) {
            if (curA == curB) {
                return curA;
            }
            curA = curA->next;
            curB = curB->next;
        }
        return NULL;
    }
};
```

理解思路，但不明白？

要求是节点指针相同

- 时间复杂度：$O(n + m)$
- 空间复杂度：$O(1)$

其他语言版本

Java：

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        ListNode curA = headA;
        ListNode curB = headB;
        int lenA = 0, lenB = 0;
        while (curA != null) { // 求链表A的长度
            lenA++;
            curA = curA.next;
        }
        while (curB != null) { // 求链表B的长度
            lenB++;
            curB = curB.next;
        }
        curA = headA;
        curB = headB;
```

```
        // 让curA为最长链表的头，lenA为其长度
        if (lenB > lenA) {
            //1. swap (lenA, lenB);
            int tmpLen = lenA;
            lenA = lenB;
            lenB = tmpLen;
            //2. swap (curA, curB);
            ListNode tmpNode = curA;
            curA = curB;
            curB = tmpNode;
        }
        // 求长度差
        int gap = lenA - lenB;
        // 让curA和curB在同一起点上（末尾位置对齐）
        while (gap-- > 0) {
            curA = curA.next;
        }
        // 遍历curA 和 curB，遇到相同则直接返回
        while (curA != null) {
            if (curA == curB) {
                return curA;
            }
            curA = curA.next;
            curB = curB.next;
        }
        return null;
    }

}
```

Python：

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNode:
        lengthA,lengthB = 0,0
        curA,curB = headA,headB
        while(curA!=None): #求链表A的长度
            curA = curA.next
            lengthA +=1

        while(curB!=None): #求链表B的长度
            curB = curB.next
            lengthB +=1

        curA, curB = headA, headB

        if lengthB>lengthA: #让curA为最长链表的头，lenA为其长度
            lengthA, lengthB = lengthB, lengthA
            curA, curB = curB, curA

        gap = lengthA - lengthB #求长度差
```

```
        while(gap!=0):
            curA = curA.next #让curA和curB在同一起点上
            gap -= 1

        while(curA!=None):
            if curA == curB:
                return curA
            else:
                curA = curA.next
                curB = curB.next
        return None
```

Go：

```
func getIntersectionNode(headA, headB *ListNode) *ListNode {
    curA := headA
    curB := headB
    lenA, lenB := 0, 0
    // 求A，B的长度
    for curA != nil {
        curA = curA.Next
        lenA++
    }
    for curB != nil {
        curB = curB.Next
        lenB++
    }
    var step int
    var fast, slow *ListNode
    // 请求长度差，并且让更长的链表先走相差的长度
    if lenA > lenB {
        step = lenA - lenB
        fast, slow = headA, headB
    } else {
        step = lenB - lenA
        fast, slow = headB, headA
    }
    for i:=0; i < step; i++ {
        fast = fast.Next
    }
    // 遍历两个链表遇到相同则跳出遍历
    for fast != slow {
        fast = fast.Next
        slow = slow.Next
    }
    return fast
}
```

javaScript:

```
/**
 * @param {ListNode} headA
 * @param {ListNode} headB
 * @return {ListNode}
 */
```

```
var getListLen = function(head) {
    let len = 0, cur = head;
    while(cur) {
       len++;
       cur = cur.next;
    }
    return len;
}
var getIntersectionNode = function(headA, headB) {
    let curA = headA,curB = headB,
        lenA = getListLen(headA),
        lenB = getListLen(headB);
    if(lenA < lenB) {
        [curA, curB] = [curB, curA];
        [lenA, lenB] = [lenB, lenA];
    }
    let i = lenA - lenB;
    while(i-- > 0) {
        curA = curA.next
    }
    while(curA && curA !== curB) {
        curA = curA.next;
        curB = curB.next;
    }
    return curA;
};
```

- 作者微信：程序员Carl
- B站视频：代码随想录
- 知识星球：代码随想录