**难度: 简单**

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：intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA = 2, skipB = 3

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。

示例 2：

输入：intersectVal = 2, listA = [0,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1

输出：Reference of the node with value = 2

输入解释：相交节点的值为 2 （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 [0,9,1,2,4]，链表 B 为 [3,2,4]。在 A 中，相交节点前有 3 个节点；在 B 中，相交节点前有 1 个节点。

示例 3：

输入：intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2

输出：null

输入解释：从各自的表头开始算起，链表 A 为 [2,6,4]，链表 B 为 [1,5]。由于这两个链表不相交，所以 intersectVal 必须为 0，而 skipA 和 skipB 可以是任意值。

解释：这两个链表不相交，因此返回 null。

注意：

如果两个链表没有交点，返回 null 。

在返回结果后，两个链表仍须保持原有的结构。

可假定整个链表结构中没有循环。

程序尽量满足 O(n) 时间复杂度，且仅用 O(1) 内存。

来源：力扣（LeetCode）

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

著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。

**思路1(双循环比较暴力):**

1). 每次让第一个链表的引用比较第二个链表中所有的引用

如果相同,则返回

2). 不相同,就让第一个链表下移,然后再次比较第二个链表中所以有的引用

3). 一直持续只要找到,或者循环结束,返回null

|  |
| --- |
| public ListNode getIntersectionNode(ListNode headA, ListNode headB) {  ListNode tempA = headA;  while (tempA != null) {  ListNode tempB = headB;  while(tempB != null) {  if (tempA == tempB) {  return tempA;  }  tempB = tempB.next;  }  tempA = tempA.next;  }  return null;  } |

**思路2(双指针):**

|  |
| --- |
| public ListNode getIntersectionNode2(ListNode headA, ListNode headB) |

1). 循环用第一个链表与第二个链表进行相互比较

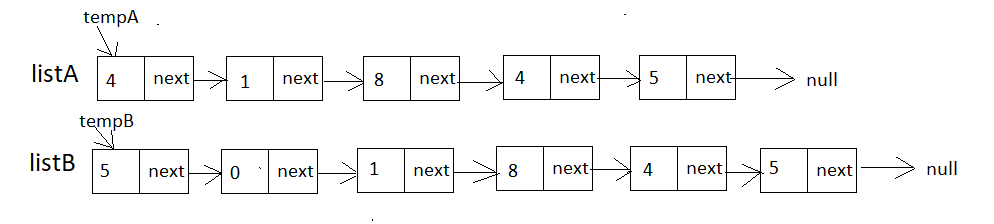
2). 当第一个链表为空就把第二个链表给到第一个链表进行比较

3). 当第二个链表为空就把第一个链表给到第二个链表进行比较

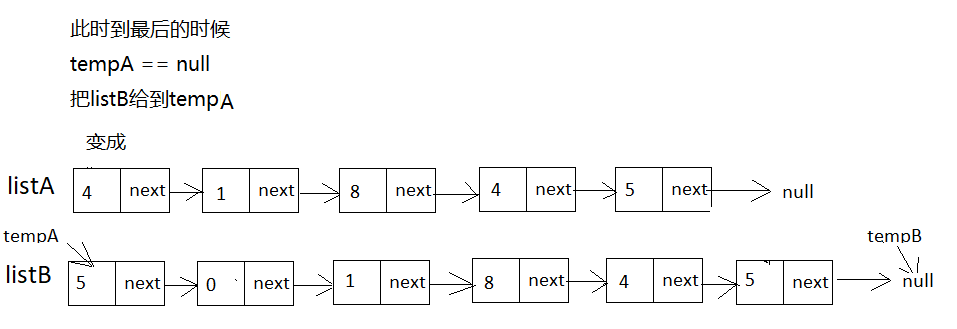
例如: intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5]

图解:

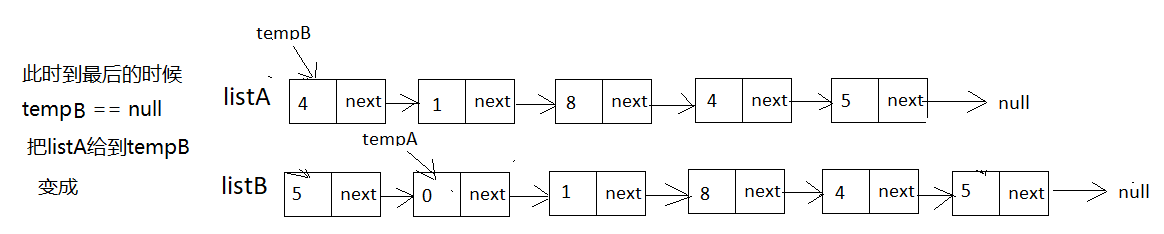
循环用第一个链表与第二个链表进行相互比较



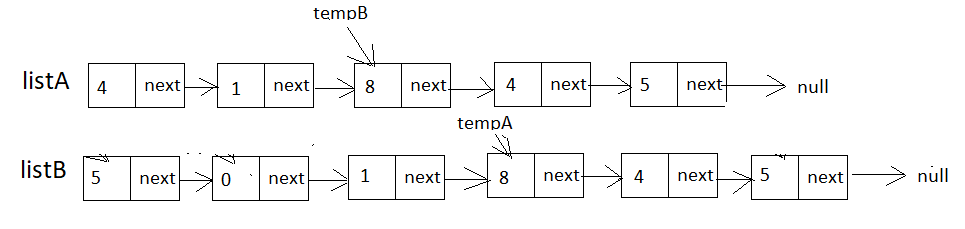
当第一个链表为空就把第二个链表给到第一个链表进行比较



此时在往下比较



此时相同返回



如果不是这个8而是一个9,不可能相同, 但是一直到最后null还是会相同,如果就返回null

|  |
| --- |
| public ListNode getIntersectionNode2(ListNode headA, ListNode headB) {  if (headA == null || headB == null) {  return null;  }  ListNode tempA = headA;  ListNode tempB = headB;  while(tempA != tempB) {  if (tempA != null) {  tempA = tempA.next;  }else {  tempA = headB;  }  if (tempB != null) {  tempB = tempB.next;  }else {  tempB = headA;  }  }  return tempA;  } |