



题目来源于LeetCode上第160号问题：相交链表。题目难度为Easy，目前通过率54.4%。##题目描述 编写一个程序，找到两个单链表相交的起始节点。如下面的两个链表： LeetCode图解|160.相交链表 在节点 c1 开始相交。示例

1:  LeetCode图解|160.相交链表 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，如果两个链表相交则不能为 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。
```

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。

##题目解析 为满足题目时间复杂度和空间复杂度的要求，我们可以使用双指针法。

- 创建两个指针pA和pB分别指向链表的头结点headA和headB。
- 当pA到达链表的尾部时，将它重新定位到链表B的头结点headB，同理，当pB到达链表的尾部时，将它重新定位到链表A的头结点headA。
- 当pA与pB相等时便是两个链表第一个相交的结点。这里其实就是相当于把两个链表拼在一起了。pA指针是按B链表拼在A链表后面组成的新链表遍历，而pB指针是按A链表拼在B链表后面组成的新链表遍历。举个简单的例子：A链表：{1,2,3,4} B链表：{6,3,4} pA按新拼接的链表{1,2,3,4,6,3,4}遍历 pB按新拼接的链表{6,3,4,1,2,3,4}遍历

##动画理解

##代码实现

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
        ListNode *pA = headA;
        ListNode *pB = headB;
        while (pA != pB) {
            if (pA != NULL) {
                pA = pA->next;
            } else {
                pA = headB;
            }
            if (pB != NULL) {
                pB = pB->next;
            } else {
                pB = headA;
            }
        }
        return pA;
    }
};
```

```
        pA = headB;
    }
    if (curB != NULL) {
        pB = pB->next;
    } else {
        pB = headA;
    }
}
return pA;
}
};
```

##复杂度分析

- 时间复杂度： $O(m+n)$ 。
- 空间复杂度： $O(1)$