# 题目

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：



在节点 c1 开始相交。

**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。

**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，如果两个链表相交则不能为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。

**类似题目：**面试题52

# 分析

## 方法一：暴力法

## 方法二：哈希表

### unordered\_set

**思路：**

遍历链表A并将每个结点的地址/引用存储在哈希表中。然后检查链表B中的每一个结点bi是否在哈希表中。若在，则bi为相交结点。

说明：唯一的相交点，可以考虑使用数据结构set。  
 **代码：**

class Solution {

public:

ListNode \*getIntersectionNode(ListNode \*headA, ListNode \*headB) {

std::unordered\_set<ListNode\*> set;

ListNode\* cur\_a = headA;

while (cur\_a)

{

set.insert(cur\_a);

cur\_a = cur\_a->next;

}

ListNode\* cur\_b = headB;

while (cur\_b)

{

if(set.find(cur\_b) != set.end()) //找到了

{

return cur\_b;

}

cur\_b = cur\_b->next;

}

return nullptr;

}

};

**复杂度分析：**

时间复杂度 : O(m+n)。

空间复杂度 : O(m)或O(n)。

### unordered\_mp

也可以采用unordered\_mp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) {

        unordered\_map<ListNode\*,int> mp;

        while(headA)

        {

            mp[headA]++;

            headA = headA->next;

        }

        while(headB)

        {

            mp[headB]++;

            if(mp[headB]>1)

                return headB;

            headB = headB->next;

        }

        return nullptr;

    }

};

## 方法三：双指针

**思路：**

**代码：**

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

private:

int getListLength(ListNode \*node)

{

int len = 0;

while(node)

{

len++;

node = node->next;

}

return len;

}

//两链表尾节点对齐

ListNode \*forwardLongList(int long\_len,int short\_len,ListNode \*head)

{

int len = long\_len - short\_len;

while(head && len)

{

head = head->next;

len--;

}

return head;

}

public:

ListNode \*getIntersectionNode(ListNode \*headA, ListNode \*headB) {

int listALen = getListLength(headA);

int listBLen = getListLength(headB);

if(listALen > listBLen)

{

headA = forwardLongList(listALen,listBLen,headA);

}

else

{

headB = forwardLongList(listBLen,listALen,headB);

}

while(headA && headB)

{

if(headA == headB)

return headA;

headA = headA->next;

headB = headB->next;

}

return NULL;

}

};