两个链表的第一个公共节点\_LeetCode\_160\_剑指Offer\_56

# 两个链表的第一个公共节点\_LeetCode\_160\_剑指Offer\_56

## 题目介绍

输入两个链表，找出它们的第一个公共结点。

\* LeetCode160\_IntersectionOfTwoLinkedLists\_JianzhiOffer\_52\_FindFirstCommonNodeOfTwoLinkedList

\* 难度：Easy

\* DateTime : 2018-10-06 10:36

\* 题目描述:

\* 输入两个链表，找出它们的第一个公共结点。

\* Write a program to find the node

\* at which the intersection of two singly linked lists begins.

\* For example, the following two linked lists:

\* A: a1 → a2

\* ↘

\* c1 → c2 → c3

\* ↗

\* B: b1 → b2 → b3

\* begin to intersect at node c1.

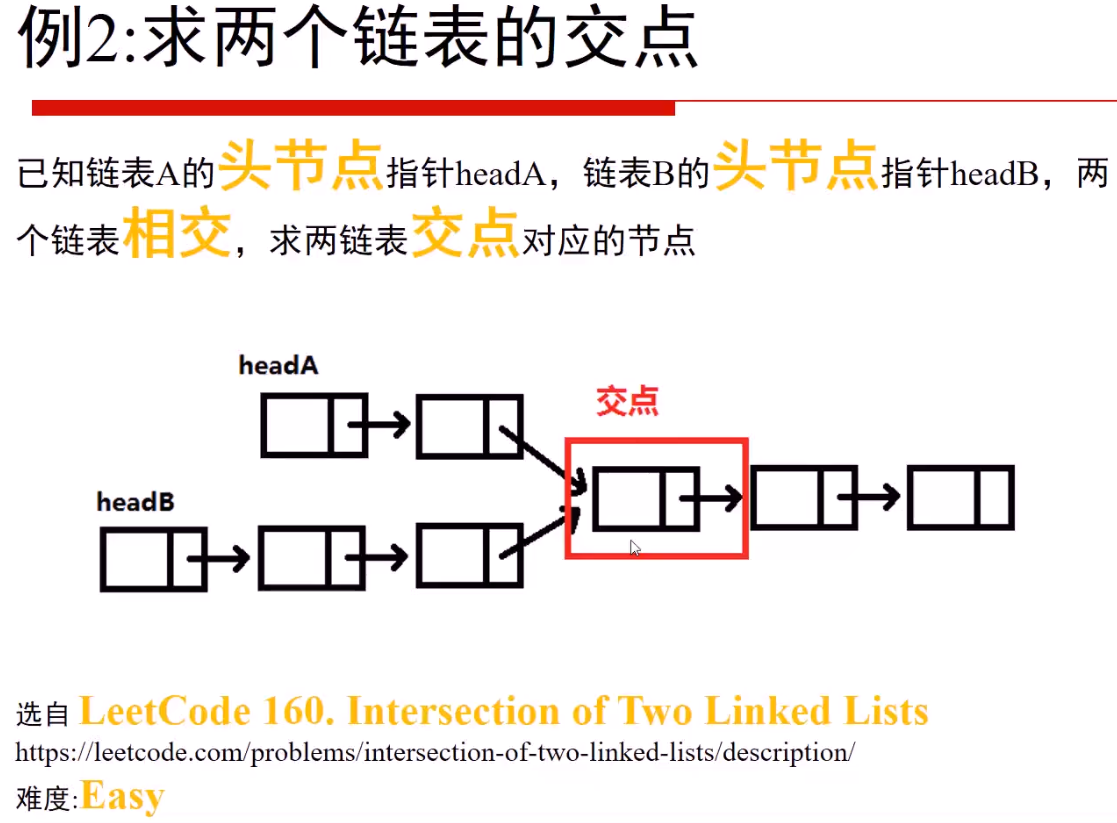
\* Notes:

\* If the two linked lists have no intersection at all, return null.

\* The linked lists must retain their original structure after the function returns.

\* You may assume there are no cycles anywhere in the entire linked structure.

\* Your code should preferably run in O(n) time and use only O(1) memory.



## 思路分析

\* 思路分析：

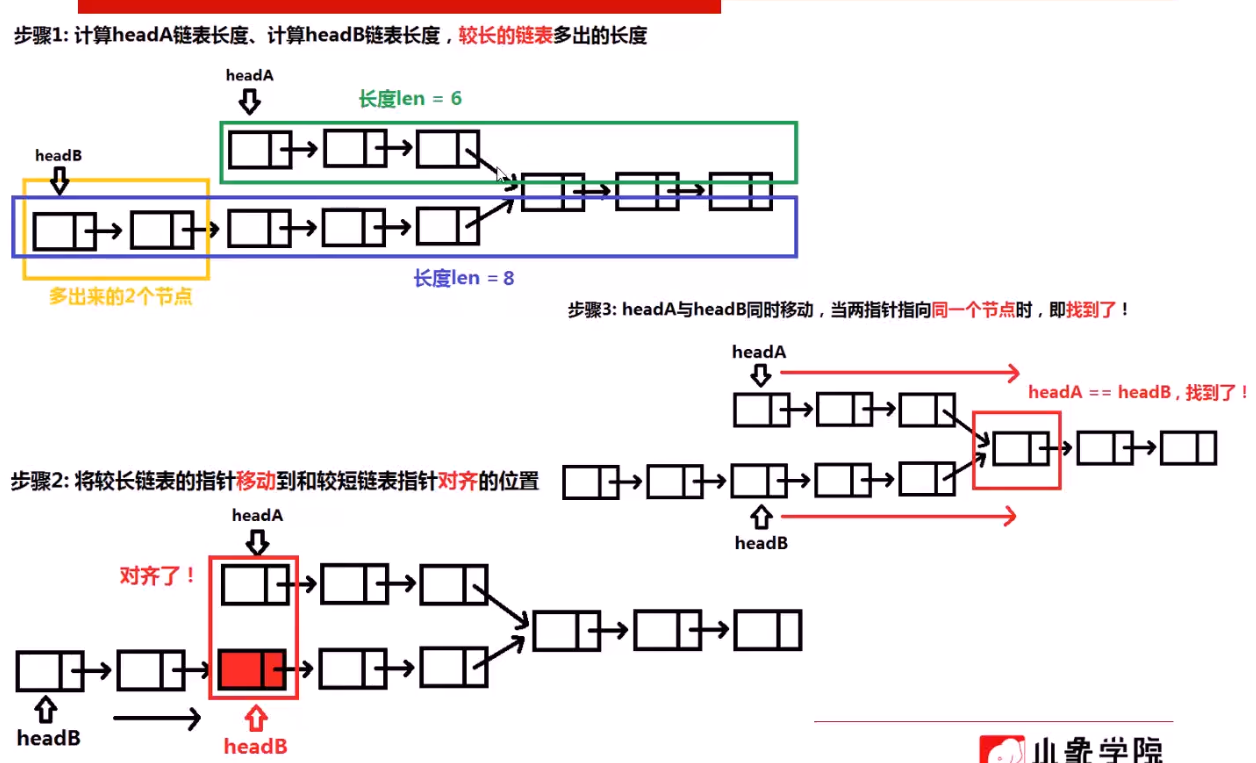
### \* 方法1：蛮力法，遍历第一个链表的每一个节点的时候，遍历第二个节点，时间复杂度为O(mn);

\* 由于两个链表有公共节点，那么应该成"Y"的结构，即从某个节点开始，以后都一样。那么如果从后往前查找

\* 就只需要O(n)的时间复杂度。

### \* 方法2：利用两个栈分别存储两个链表，从栈顶依次判断是否相等；

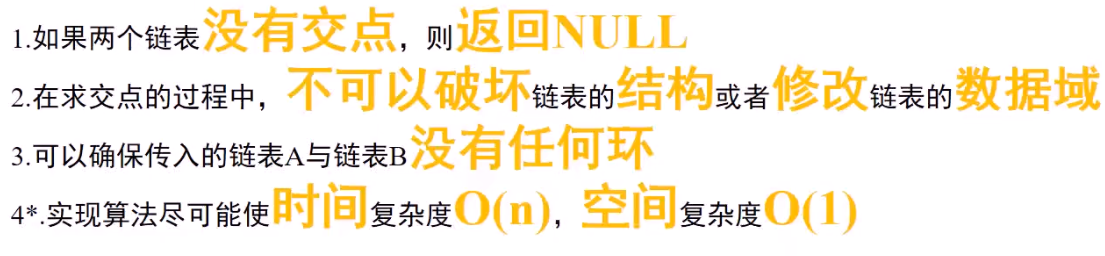
### \* 方法3：先计算两个链表的长度，然后先让长的链表指针走几步，然后两指针同时出发判断即可。

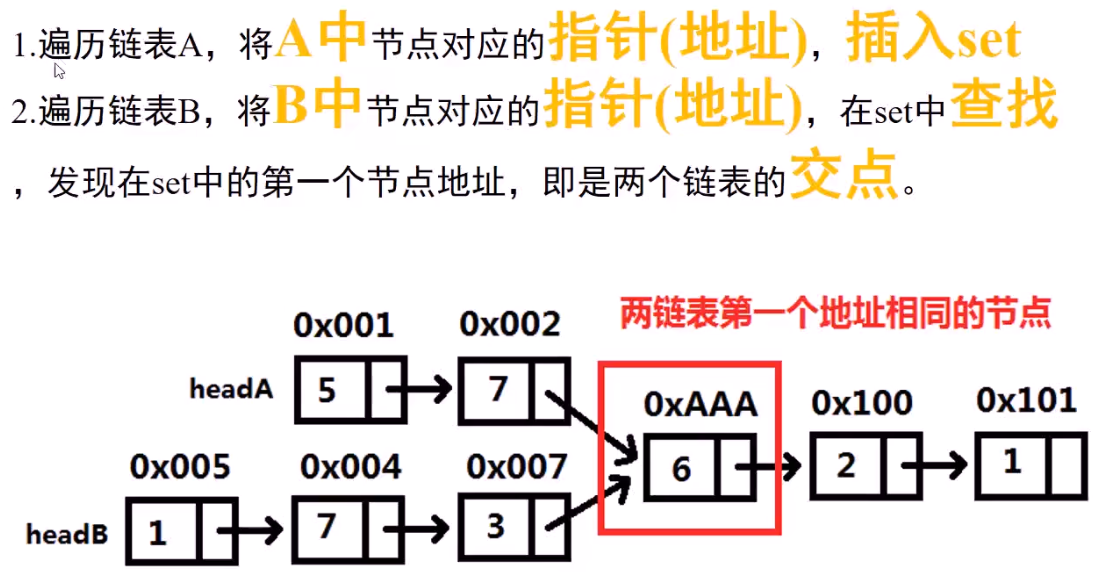


### \* 方法4：利用Set集合完成。首先将链表head1的节点都put到set集合中，然后遍历链表head2；逐个判断set中是否含有head2中的节点，若有，则直接返回，即为第一个公共节点。空间复杂度为O(N).

Set集合：







## Java代码

### 方法2：利用两个栈实现

public ListNode getIntersectionNode(ListNode pHead1, ListNode pHead2) {

if(pHead1==null||pHead2==null) return null;

ListNode head1 = pHead1,head2 = pHead2;//不改变原来的引用

Deque<ListNode> stack1 = new ArrayDeque<ListNode>();

Deque<ListNode> stack2 = new ArrayDeque<ListNode>();

while(head1 != null){

stack1.push(head1);

head1 = head1.next;

}

while(head2 != null){

stack2.push(head2);

head2 = head2.next;

}

ListNode preCommonNode = null;

while(!stack1.isEmpty() && !stack2.isEmpty()){

ListNode node1 = stack1.pop();

ListNode node2 = stack2.pop();

**if(node1 == node2) preCommonNode = node1;//倒着看应该获取最后一个公共节点**

}

return preCommonNode;

}

### 方法3：先求链表的长度，然后利用两指针遍历链表

注意：需要判断哪一个链表长及相应的处理。

public ListNode **getIntersectionNode\_2** (ListNode pHead1, ListNode pHead2) {

if(pHead1==null||pHead2==null) return null;

ListNode head1 = pHead1,head2 = pHead2;//不改变原来的引用

//求链表长度

int len1 = getLengthOfLinkedList(head1);

int len2 = getLengthOfLinkedList(head2);

int diffLen = len1 - len2;//链表长度之差

if(len1 < len2){//默认是链表1长于链表2

ListNode temp = head1;

head1 = head2;

head2 = temp;

diffLen = -diffLen;//取相反数

}

while(diffLen-- != 0) head1 = head1.next;//先走几步

while(head1 != head2){//同时前进

head1 = head1.next;

head2 = head2.next;

}

return head1;

}

/\*\*

\* 获取链表的长度

\*/

public int getLengthOfLinkedList(ListNode head){

ListNode head1 = head;//不改变head引用

int len = 0;

**while(head1 != null){**

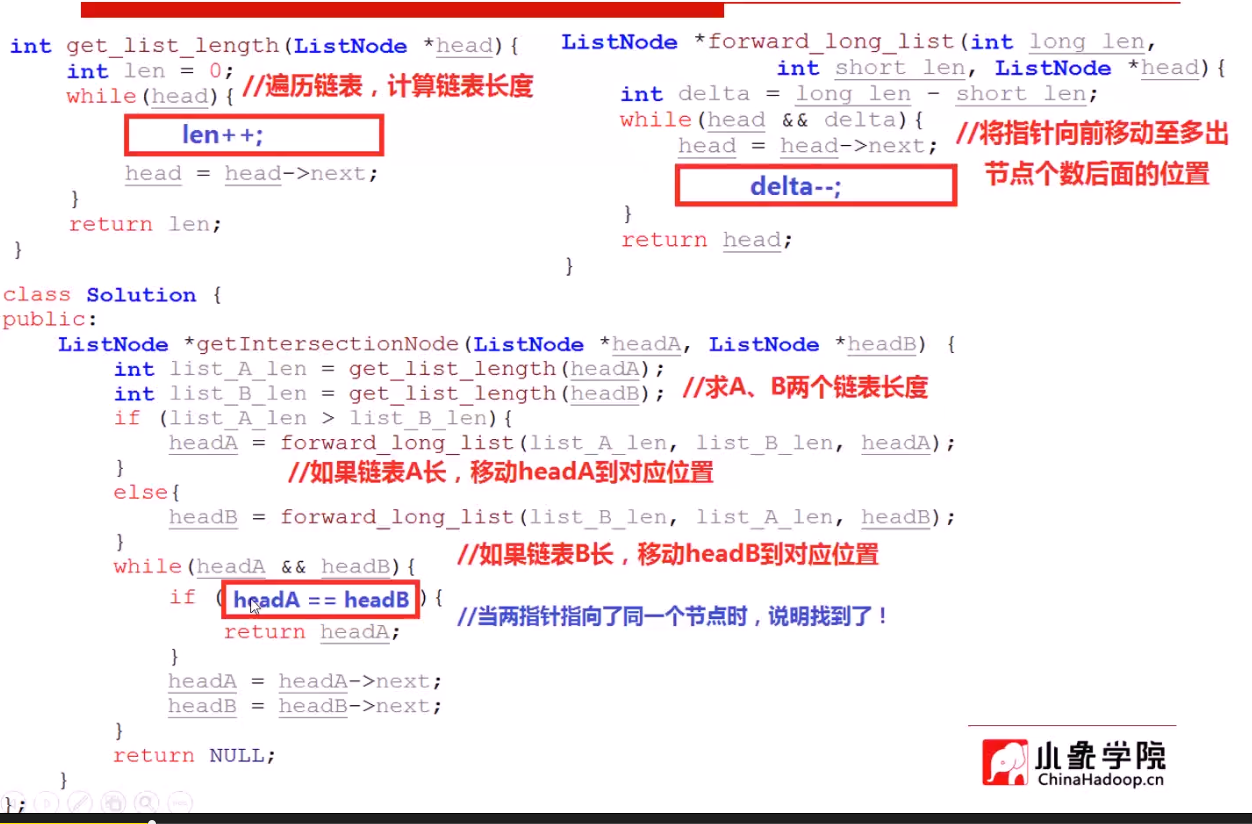
**len++;**

**head1 = head1.next;**

**}**

return len;

}



### 方法4：利用Set集合完成。

/\*\*

\* 方法4:利用Set集合完成

\*/

public ListNode getIntersectionNode\_3(ListNode pHead1, ListNode pHead2) {

if(pHead1 == null||pHead2 == null) return null;

Set<ListNode> set = new HashSet<ListNode>();

while(pHead1 != null){

set.add(pHead1);

pHead1 = pHead1.next;

}

while(pHead2 != null){

if(set.contains(pHead2))

return pHead2;

pHead2 = pHead2.next;

}

return null;

}

