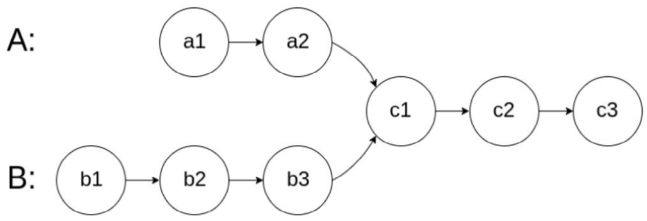
# 160. Intersection of Two Linked Lists

09 March 2022 10:06 AM

Given the heads of two singly linked-lists `headA` and `headB`, return *the node at which the two lists intersect*. If the two linked lists have no intersection at all, return `null`.

For example, the following two linked lists begin to intersect at node `c1`:



The test cases are generated such that there are no cycles anywhere in the entire linked structure.

**Note** that the linked lists must **retain their original structure** after the function returns.

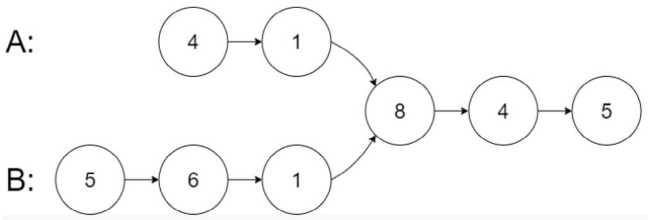
**Custom Judge:**

The inputs to the **judge** are given as follows (your program is **not** given these inputs):

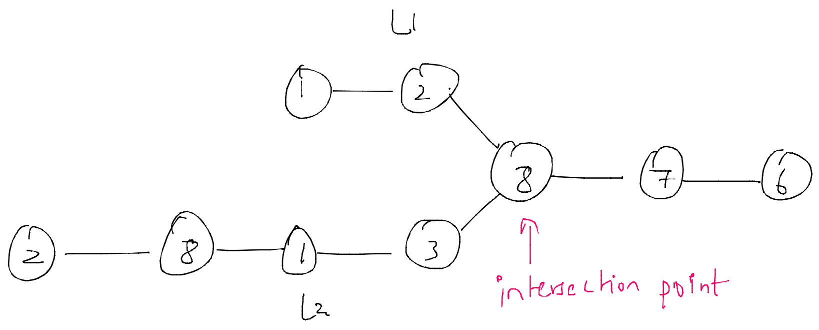
- `intersectVal` - The value of the node where the intersection occurs. This is `0` if there is no intersected node.
- `listA` - The first linked list.
- `listB` - The second linked list.
- `skipA` - The number of nodes to skip ahead in `listA` (starting from the head) to get to the intersected node.
- `skipB` - The number of nodes to skip ahead in `listB` (starting from the head) to get to the intersected node.

The judge will then create the linked structure based on these inputs and pass the two heads, `headA` and `headB` to your program. If you correctly return the intersected node, then your solution will be **accepted**.

**Example 1:**



```
Input: intersectVal = 8, listA = [4,1,8,4,5], listB = [5,6,1,8,4,5], skipA
= 2, skipB = 3
Output: Intersected at '8'
Explanation: The intersected node's value is 8 (note that this must not be
0 if the two lists intersect).
From the head of A, it reads as [4,1,8,4,5]. From the head of B, it reads
as [5,6,1,8,4,5]. There are 2 nodes before the intersected node in A;
There are 3 nodes before the intersected node in B.
```

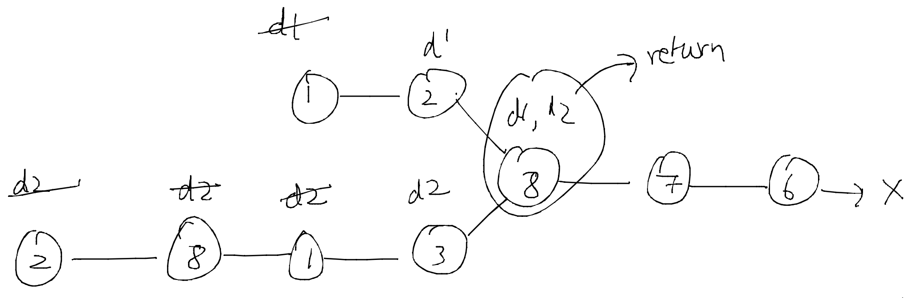


Brute force:

To compare ever node pointer in the 1st list with the every other node pointer in the second list by which the matching node pointers will lead us to to the intersecting node. But the TC ⇒ O(mn) SC ⇒ O(1)

Better Approach: 1

* Find the length (L1 and L2) of both list → O(n) + O(M) = O(max (m, n))
* Take the difference d of the length -- O(1)
* Make d steps in longer list -- O(d)
* Steps in both list in parallel until links to next node match — O(min (m, n))
* Space Complexity → O(1)
* If no intersection return null.



L1 = 5

L2 = 7

diff = L2 − L1
= 2

→ The longer list is L2 so move L2 by 2 times

→ Now move both parallel

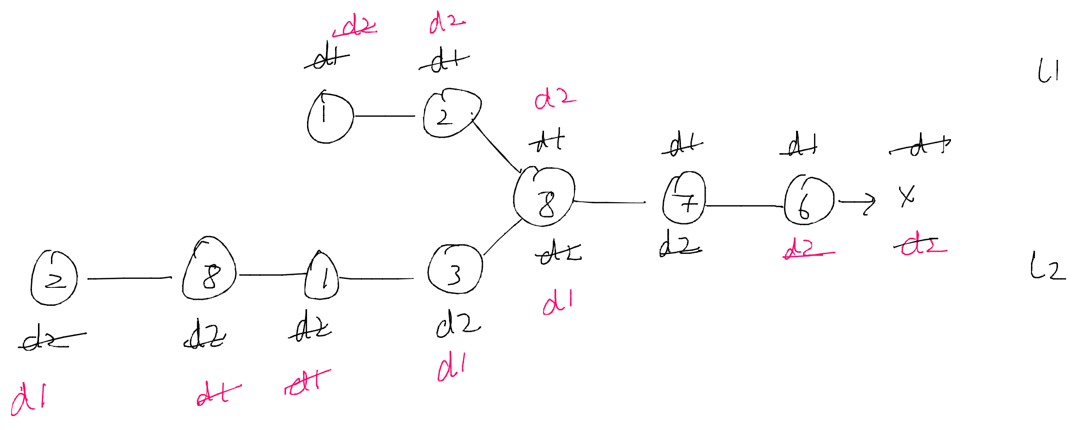
```java
public class Solution {
    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        int l1 = 0, l2 = 0, diff = 0;
        ListNode head1 = headA;
        ListNode head2 = headB;

        while(head1 != null){
            l1++;
            head1 = head1.next;
        }
        while(head2 != null){
            l2++;
            head2 = head2.next;
        }

        if(l1 < l2){
            head1 = headB;
            head2 = headA;
            diff = l2 - l1;
        }

        for(int i = 0; i < diff; i++)
            head1 = head1.next;
        while(head1 != null && head2 != null){
            if(head1 == head2)
                return head1;    // the point it get intersected
            head1 = head1.next;
            head2 = head2.next;
        }
        return null;
    }
}
```

Approach 3: (without the iteration)



* Take two dummy nodes (d1, d2)

* Start moving both parallel

* The moment any of your dummy node reaches the end point of linked list you take that dummy node assing it to opposite side of linked list.
  (here d1 reach the end point so assing it to L2.)

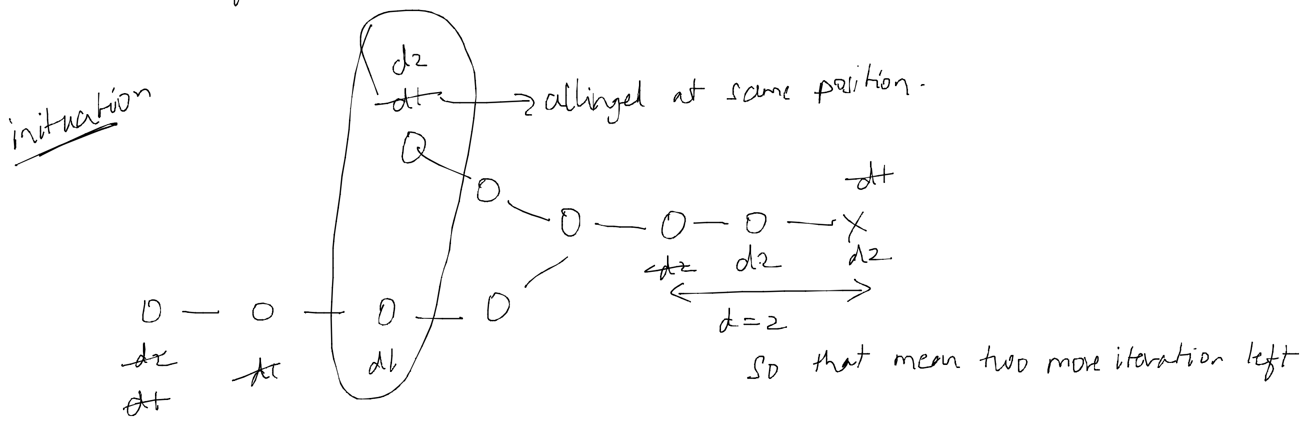
  and agin keep moving d1 and d2.

* Now agin the dummy node reachs the end point (d2) so now point d2 to the L1.

* Now both the dummy node stand at the same node.

* Now, when at same iteration d1 and d2 collide that the point of intersection.

* if there is no intersection then it's null.



initution
d2
~~d1~~
allinged at same position.
~~d1~~
~~d2~~
d2
d2
d=2
~~d2~~
~~d1~~
~~d1~~
d1
So that mean two more iteration left

T.C => O(2M)

C++

```
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
        // boundary check
        if(headA == NULL && headB == NULL) return NULL;

        ListNode *a = headA;
        ListNode *b = headB;

        // if a & b have different length, then we will stop the loop after second iteration.
        while(a != b){
            //for the end of first iteration, we just reset the pointer to the head of another
linkedlist
            a = a == NULL ? headB : a->next;    // if a is null then shit it to other linked list
                                                // if it not null simply move it to next node
            b = b == NULL ? headA : b->next;    // same for b
        }
        return a;   // the moment a == b we return it.
    }
};
```

JAVA

```java
public class Solution {
    public ListNode getIntersectionNode(ListNode headA, ListNode headB)
    {
        //boundary check
        if(headA == null || headB == null) return null;

        ListNode a = headA;
        ListNode b = headB;

        //if a & b have different len, then we will stop the loop after second iteration
        while( a != b){
            //for the end of first iteration, we just reset the pointer to the head of another
linkedlist
            a = a == null? headB : a.next; // if a is null then shit it to other linked list
                                            // if it not null simply move it to next node
            b = b == null? headA : b.next;      // same for b
        }

        return a;   // the moment a == b we return it.
    }
}
```