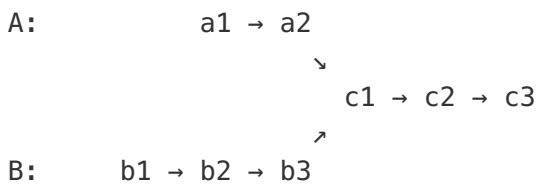
Intersection of Two Linked Lists

Write a program to find the node at which the intersection of two singly linked lists begins.

For example, the following two linked lists:



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.

Credits:
Special thanks to @stellari for adding this problem and creating all test cases.

Solution 1

```
ListNode *getIntersectionNode(ListNode *headA, ListNode *headB)
{
    ListNode *p1 = headA;
    ListNode *p2 = headB;

    if (p1 == NULL || p2 == NULL) return NULL;

    while (p1 != NULL && p2 != NULL && p1 != p2) {
        p1 = p1->next;
        p2 = p2->next;

        //
        // Any time they collide or reach end together without colliding
        // then return any one of the pointers.
        //
        if (p1 == p2) return p1;

        //
        // If one of them reaches the end earlier then reuse it
        // by moving it to the beginning of other list.
        // Once both of them go through reassigning,
        // they will be equidistant from the collision point.
        //
        if (p1 == NULL) p1 = headB;
        if (p2 == NULL) p2 = headA;
    }

    return p1;
}
```

written by satyakam original link here

Solution 2

I found most solutions here preprocess linkedlists to get the difference in len. Actually we don't care about the "value" of difference, we just want to make sure two pointers reach the intersection node at the same time.

We can use two iterations to do that. In the first iteration, we will reset the pointer of one linkedlist to the head of another linkedlist after it reaches the tail node. In the second iteration, we will move two pointers until they points to the same node. Our operations in first iteration will help us counteract the difference. So if two linkedlist intersects, the meeting point in second iteration must be the intersection point. If the two linked lists have no intersection at all, then the meeting pointer in second iteration must be the tail node of both lists, which is null

Below is my commented Java code:

```
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 iterat
ion
    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;
        b = b == null? headA : b.next;
    }

    return a;
}
```

written by hpplayer original link here

Solution 3

1, Get the length of the two lists.

2, Align them to the same start point.

3, Move them together until finding the intersection point, or the end null

```
public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
    int lenA = length(headA), lenB = length(headB);
    // move headA and headB to the same start point
    while (lenA > lenB) {
        headA = headA.next;
        lenA--;
    }
    while (lenA < lenB) {
        headB = headB.next;
        lenB--;
    }
    // find the intersection until end
    while (headA != headB) {
        headA = headA.next;
        headB = headB.next;
    }
    return headA;
}

private int length(ListNode node) {
    int length = 0;
    while (node != null) {
        node = node.next;
        length++;
    }
    return length;
}
```

written by zkfairytale original link here