CTCI 2.7: Intersection

October 4, 2017

Question

Given two (singly) linked lists, determine if the two lists intersect. Return the intersecting node. Note that the intersection is defined based on reference, not value. That is, if the kth node of the first linked list is the exact same node (by reference) as the jth node of the second linked list, then they are intersecting.

Explanation and Algorithm

How can we tell if there's an intersection? If two linked lists have the same last node, this means that they intersect at some point. The next part is finding out how they intersect. The idea is to set two pointers for the start of each linked list. If one list is longer than the other, advance the pointer for the longer linked list by difference in length. Traverse if/when linked list pointers are the same.

Hints

1. What are some automatic base cases?
2. If there is no matching last node, what does this mean? How should we execute this in the problem?
3. There's a lot going on here. Think of different helper methods you can do to ensure that the code is organized.
4. If we do find a match, make sure to keep track of the pointer to a value, not necessarily the value itself.

Code

```
/*Answer*/
```

```
LinkedListNode findIntersection(LinkedListNode list1, LinkedListNode
    list2) { if if(list1 == null || list2 == null) return null;
/* Get tail and sizes. */
Result result1 getTailAndSize(list1);
Result result2 = getTailAndSize(list2);

/* If different tail nodes, then there's no intersection. */
if (result1.tail != result2.tail) {
   return null;
}

/* Set pointers to the start of each linked list. */
LinkedListNode shorter = result1.size < result2.size ? list1 : list2;
    LinkedListNode longer = result1.size < result2.size ? list2 : list1;
/* Advance the pointer for the longer linked list by difference in
    lengths. */ longer = getKthNode(longer, Math.abs(result1.size -
    result2.size));

/* Move both pointers until you have a collision. */
while (shorter != longer) {
   shorter = shorter. next;
   longer = longer. next;
}

/* Return either one. */
return longer;
}

 class Result {
public LinkedListNode tail;
public int size;
public Result(LinkedListNode tail, int size) {
   this. tail = tail;
   this.size = size;
   }
}
Result getTailAndSize(LinkedListNode list) {
if (list == nUll) return null;
   int size = 1;
   LinkedListNode current = list;
 while (current. next != null) {
   size++;
   current =current.next;
}
return new Result(current, size);
}

LinkedListNode getKthNode(LinkedListNode head, int k) {
   LinkedListNode current =head;
   while (k > 0 && current != nUll) {
```

```
   current =current.next;
   k--;
    }
return current
}
```

Run time analysis

Time complexity is $O(n + m)$ where n is the size of the first list and m is the size of the second list.

Sources

.