educative

Intersection Point of Two Lists

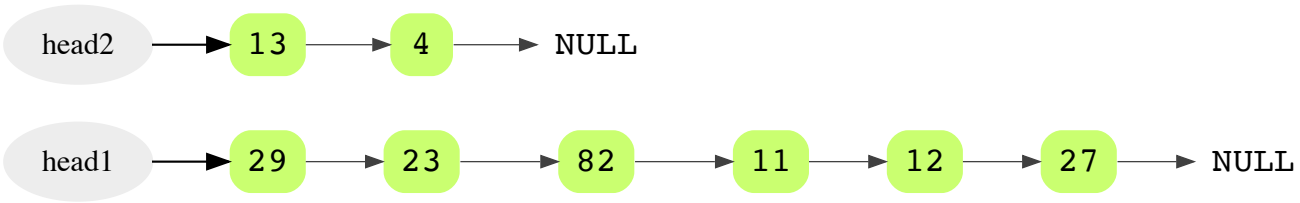
Given the head nodes of two linked lists that may or may not intersect, find out if they intersect and return the point of intersection. Return null otherwise.

We'll cover the following

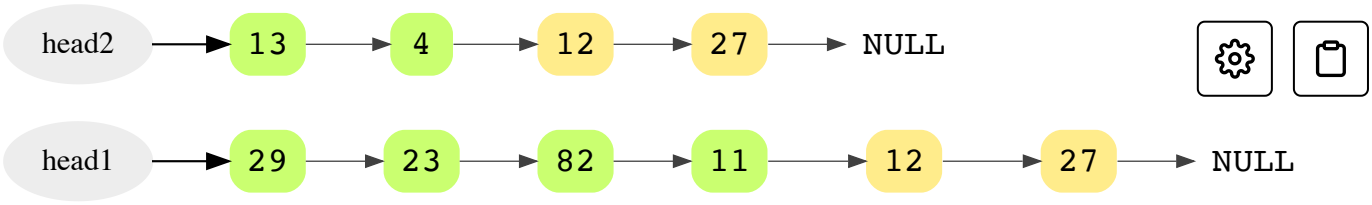
Description

Given the head nodes of two linked lists that may or may not intersect, find out if they do in fact intersect and return the point of intersection. Return null otherwise.

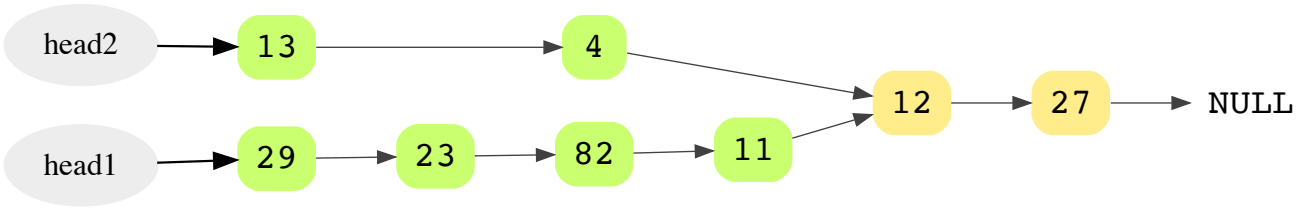
In the below example, neither lists intersect. `intersect()` should return `NULL`.



After adding nodes 12 and 27 in linked list `head2`, the list now have two same nodes as the linked list `head1`.



However, in the below example, both lists intersect at the node with data 12, so the node 4 in linked list `head2` points to node 12 and the node 11 in linked list `head1` points to node 12 (have same address).



Hints

- Find the length of both linked lists.
- The linked lists have to physically intersect. This means that their addresses need to be the same. If two nodes have the same data but their addresses are not the same, the lists won't intersect and the function should return NULL.

Try it yourself

C++ Java Python JS Ruby

```
1  def intersect(head1, head2):
2    #TODO: Write - Your - Code
3    return head1
```

Solution

Runtime complexity

The runtime complexity of this solution is *linear*, $O(m + n)$.

Where **m** is the length of the first linked list and **n** is the length of the second linked list.

Memory complexity

The memory complexity of this solution is *constant*, $O(1)$.

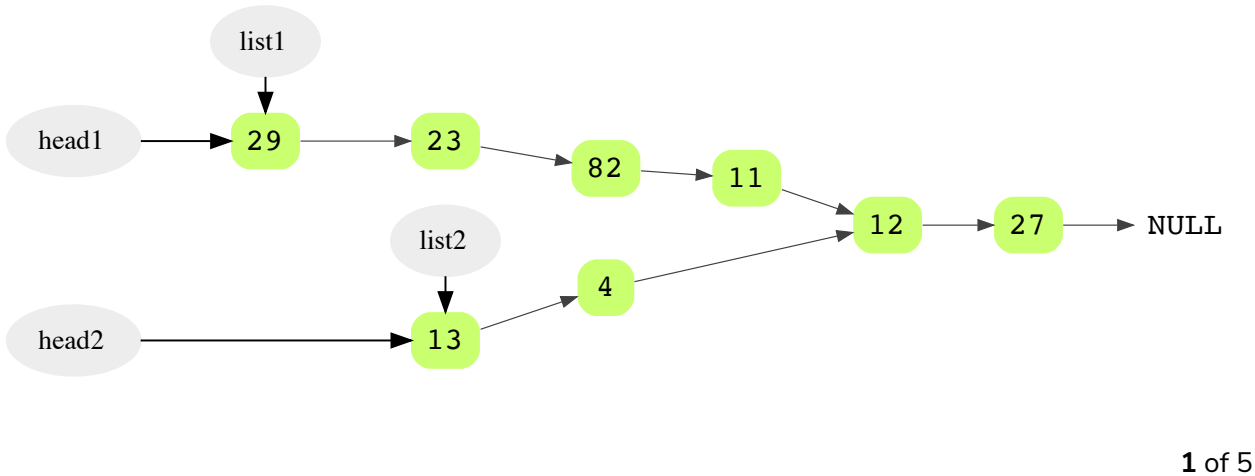
The first solution that comes to mind is one with quadratic time complexity, i.e., for each node in the first linked list, a linear scan must be done in the second linked list. If any node from the first linked list is found in the second linked list (comparing addresses of nodes, not their data), that is the intersection point. However, if none of the nodes from the first list is found in the second list, that means there is no intersection point.

Although this works, it is not efficient. A more efficient solution would be to store the nodes of the first linked list in a HashSet and then go through the second linked list nodes to check whether any of the nodes exist in the HashSet. This approach has a linear runtime complexity and linear space complexity.

We can use a much better, i.e., $O(m + n)$, linear time complexity algorithm that doesn't require extra memory. To simplify our problem, let's say both the linked lists are of the same size. In this case, you can start from the heads of both lists and compare their addresses. If these addresses match, it means it is an intersection point. If they don't match, move both pointers forward one step and compare their addresses. Repeat this process until an intersection point is found, or both of the lists are exhausted. How do we solve this problem if the lists are not of the same length? We can extend the linear time solution with one extra scan on the linked lists to find their lengths. Below is the complete algorithm:

```
Find lengths of both linked lists: L1 and L2
Calculate the difference in length of both linked lists: d = |L1 -
 L2|
Move head pointer of longer list 'd' steps forward
Now traverse both lists, comparing nodes until we find a match o
r reach the end of lists
```

Let's consider the above example of two lists that intersect at the node with data 12. The length of the first list is 6, whereas the length of the second list is 4. The difference between their lengths is 2. We'll initialize two pointers, `list1` and `list2`, at the heads of both linked lists. We need to move `list1` forward (pointing to the larger list) 2 steps. `list1` will be pointing to the third node of the first list, whereas `list2` will be pointing to the first node of the second list.



1 of 5

C++ Java Python JS Ruby

```
1  def intersect(head1, head2):
2    list1node = None
3    list1length = get_length(head1)
4    list2node = None
5    list2length = get_length(head2)
6
7    length difference = 0
```

```
 8    if list1length >= list2length :
 9      length_difference = list1length - list2lengt
10      list1node = head1
11      list2node = head2
12    else:
13      length_difference = list2length - list1lengt
14      list1node = head2
15      list2node = head1
16
17    while length_difference > 0:
18      list1node = list1node.next
19      length_difference-=1
20
21    while list1node:
22      if list1node is list2node:
23        return list1node
24
25      list1node = list1node.next
26      list2node = list2node.next
27    return None
28
```




Back

Next

Solution Review: Return the Nth Node...

Happy Number (medium)

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/intersection-point-of-two-lists__linked-list__coderust-hacking-the-coding-interview)