Daily Coding Problem #26

Problem

This problem was asked by Google.

Given a singly linked list and an integer k, remove the kth last element from the list. k is guaranteed to be smaller than the length of the list.

The list is very long, so making more than one pass is prohibitively expensive.

Do this in constant space and in one pass.

Solution

If we didn't have the constraint of needing only to make one pass, this problem would be trivial to implement. We could simply iterate over the whole list to find out the total length N of the list, and then restart from the beginning and iterate N - k steps and remove the node there. That would take constant space as well.

However, given that we have the constraint of needing to make only one pass, we have to find some way of getting the N - kth node in the list in one shot.

What we can do, then, is this:

- Set up two pointers at the head of the list (let's call them `fast` and `slow`)
- Move `fast` up by k
- Move both `fast` and `slow` together until `fast` reaches the end of the list
- Now `slow` is at the N - kth node, remove it

That only makes one pass and is constant time. The code should look something like this:

```
class Node:
    def __init__(self, val, next=None):

        self.val = val
```

```python
        self.next = next

    def __str__(self):
        current_node = self
        result = []
        while current_node:
            result.append(current_node.val)
            current_node = current_node.next
        return str(result)

def remove_kth_from_linked_list(head, k):
    slow, fast = head, head
    for i in range(k):
        fast = fast.next

    prev = None
    while fast:
        prev = slow
        slow = slow.next
        fast = fast.next

    prev.next = slow.next

head = Node(1, Node(2, Node(3, Node(4, Node(5)))))
print(head)
remove_kth_from_linked_list(head, 3)
print(head)
```