Remove N-th node from the end of a Linked List

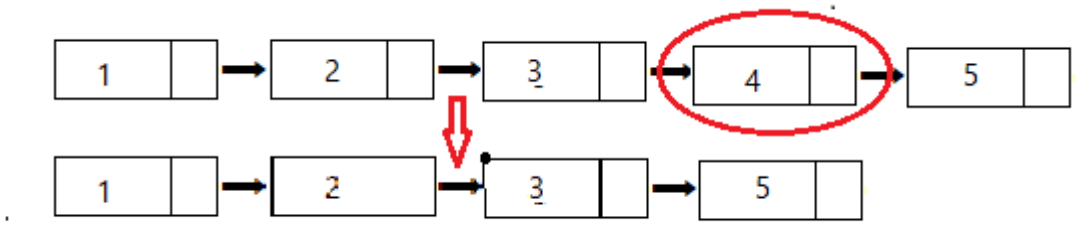
Problem Statement: Given a linked list, and a number N. Find the Nth node from the end of this linkedlist, and delete it. Return the head of the new modified linked list.

Example 1 :

Input: `head = [1,2,3,4,5], n = 2`

Output: `[1,2,3,5]`

Explanation: Refer Below Image

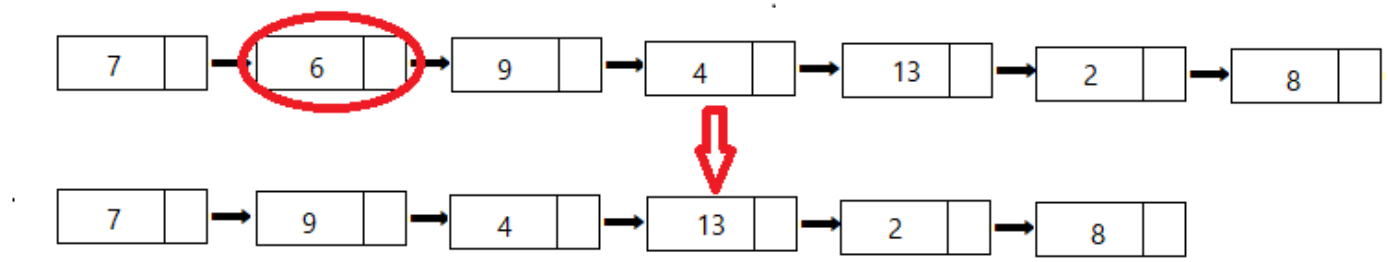


Example 2:

Input: `head = [7,6,9,4,13,2,8], n = 6`

Output: `[7,9,4,13,2,8]`

Explanation: Refer Below Image



Example 3:

```
Input: head = [1,2,3], n = 3
```

```
Output: [2,3]
```

Solution :

Disclaimer: *Don't jump directly to the solution, try it out yourself first.*

Solution 1: Naive Approach [Traverse 2 times]

Intuition: We can traverse through the Linked List while maintaining a count of nodes, let's say in variable ***count*** , and then traversing for the 2nd time for (n – ***count***) nodes to get to the nth node of the list.

```
public ListNode removeNthFromEnd(ListNode head, int n) {
    if(head==null||head.next==null)
        return null;
    ListNode start=new ListNode();
    start.next=head;
    int length=0;
    ListNode curr=head;
    while(curr!=null)
    {
        length++;
        curr=curr.next;
    }
    curr=start;
    for(int i=0;i<length-n;i++)
    {
        curr=curr.next;
    }
```

```
        curr.next=curr.next.next;
        return start.next;
    }
```

Solution 2: [Efficient] Two pointer method

Unlike the above approach, we don't have to maintain the count value, we can find the nth node just by one traversal by using two pointer approach.

Intuition :

What if we had to modify the same above approach to work in just one traversal? Let's see what all information we have here:

1. We have the flexibility of using two-pointers now.
2. We know, the n-th node from the end is the same as (total nodes – n)th node from start.
3. But, we are not allowed to calculate total nodes, as we can do only one traversal.

What if, one out of the two-pointers is at the nth node from start instead of end? Will it make anything easier for us?

Yes, with two pointers in hand out of which one at n-th node from start, we can just advance both of them till end simultaneously, once the faster reaches the end, the slower will stand at the n-th node from the end.

Approach :

1. Take two dummy nodes, who's next will be pointing to the head.
2. Take another node to store the head, initially it,s a dummy node(start), and the next of the node will be pointing to the head.The reason why we are using this extra dummy node, is because there is an edge case. If the node is equal to the length of the linkedlist, then this slow's will point to slow's next→ next. And we can say our dummy start node will be broken, and will be connected to the slow's next→ next.
3. Start traversing until the fast pointer reaches the nth node.

```
for(int i = 1; i <= n; ++i)
    fast = fast->next;
```

4. Now start traversing by one step both of the pointers until the fast pointers reach the end.

```
while(fast->next != NULL)
{
    fast = fast->next;
    slow = slow->next;
}
```

5. When the traversal is done, just do the deleting part. Make slow pointer's next to the next of next of the slow pointer to ignore/disconnect the given node.

```
slow->next = slow->next->next;
```

6. Last, return the next of start.

Dry Run: We will be taking the first example for the dry run, so, the linkedlist is [1,2,3,4,5] and the node which has to be deleted is 2 from the last. For the first time fast ptr starts traversing from node 1 and reaches to 2, as it traverses for node number 2, then the slow ptr starts increasing one, and as well as the fast ptr, until it reaches the end.

- 1st traversal : fast=3, slow=1
- 2nd traversal : fast=4, slow=2
- 3rd traversal : fast=5, slow=3

Now, the slow->next->next will be pointed to the slow->next

So , the new linked list will be [1,2,3,5]

Source Code:

```
public ListNode removeNthFromEnd(ListNode head, int n) {
    ListNode start = new ListNode();
```

```
        start.next = head;
        ListNode fast = start;
        ListNode slow = start;

        for(int i = 1; i <= n; ++i)
            fast = fast.next;

        while(fast.next != null)
        {
            fast = fast.next;
            slow = slow.next;
        }

        slow.next = slow.next.next;

        return start.next;
    }
```

Time Complexity: O(N)

Space Complexity: O(1)