19. Remove Nth Node From End of List

May 4, 2016 | 289.9K views

Average Rating: 4.66 (154 votes)

Given a linked list, remove the *n*-th node from the end of list and return its head.

Example:

```
Given linked list: 1->2->3->4->5, and n = 2.

After removing the second node from the end, the linked list becomes 1->2->3->5.
```

Note:

Given *n* will always be valid.

Follow up:

Could you do this in one pass?

Summary

This article is for beginners. It introduces the following idea: Linked List traversal and removal of nth element from the end.

Solution

Approach 1: Two pass algorithm

Intuition

We notice that the problem could be simply reduced to another one : Remove the $(L - n + 1)$ th node from the beginning in the list , where L is the list length. This problem is easy to solve once we found list length L.

Algorithm

First we will add an auxiliary "dummy" node, which points to the list head. The "dummy" node is used to simplify some corner cases such as a list with only one node, or removing the head of the list. On the first pass, we find the list length L. Then we set a pointer to the dummy node and start to move it through the list till it comes to the $(L - n)$ th node. We relink `next` pointer of the $(L - n)$ th node to the $(L - n + 2)$ th node and we are done.

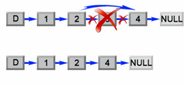

Figure 1. Remove the L - n + 1 th element from a list.

Java

```
public ListNode removeNthFromEnd(ListNode head, int n) {
    ListNode dummy = new ListNode(0);
    dummy.next = head;
    int length  = 0;
    ListNode first = head;
    while (first != null) {
        length++;
        first = first.next;
    }
    length -= n;
    first = dummy;
    while (length > 0) {
        length--;
        first = first.next;
    }
    first.next = first.next.next;
    return dummy.next;
}
```

Complexity Analysis

- Time complexity : $O(L)$.

  The algorithm makes two traversal of the list, first to calculate list length L and second to find the $(L - n)$ th node. There are $2L - n$ operations and time complexity is $O(L)$.

- Space complexity : $O(1)$.

  We only used constant extra space.

Approach 2: One pass algorithm

Algorithm

The above algorithm could be optimized to one pass. Instead of one pointer, we could use two pointers. The first pointer advances the list by $n + 1$ steps from the beginning, while the second pointer starts from the beginning of the list. Now, both pointers are exactly separated by n nodes apart. We maintain this constant gap by advancing both pointers together until the first pointer arrives past the last node. The second pointer will be pointing at the nth node counting from the last. We relink the next pointer of the node referenced by the second pointer to point to the node's next next node.

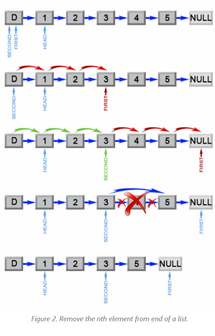

Figure 2. Remove the nth element from end of a list.

Java

```
public ListNode removeNthFromEnd(ListNode head, int n) {
    ListNode dummy = new ListNode(0);
    dummy.next = head;
    ListNode first = dummy;
    ListNode second = dummy;
    // Advances first pointer so that the gap between first and second is n nodes apart
    for (int i = 1; i <= n + 1; i++) {
        first = first.next;
    }
    // Move first to the end, maintaining the gap
    while (first != null) {
        first = first.next;
        second = second.next;
    }
    second.next = second.next.next;
    return dummy.next;
}
```

Complexity Analysis

- Time complexity : $O(L)$.

  The algorithm makes one traversal of the list of L nodes. Therefore time complexity is $O(L)$.

- Space complexity : $O(1)$.

  We only used constant extra space.

Rate this article:

Previous Next

Comments: 150

Sort By

Type comment here... (Markdown is supported)

Preview Post

sharyuliang ★ 216 March 17, 2019 2:47 PM

The `one pass` is NOT a real one pass.

```
while (first != null) {
    first = first.next;
    second = second.next;
}
```

Read More

215 Share Reply

SHOW 17 REPLIES

wickman ★ 53 September 26, 2018 3:34 AM Report

My solution only using one pass and one loop:

```
def removeNthFromEnd(self, head, n):
    size = 1
    node = head
```

Read More

51 Share Reply

SHOW 13 REPLIES

Roobecks ★ 54 January 6, 2019 11:09 PM

Returning `dummy.next` and not `head` is very crucial.
Thanks for such a clear cut solution.

42 Share Reply

SHOW 5 REPLIES

XiaoChow ★ 120 July 17, 2018 10:28 AM Report

I don't think the second method is one pass. Using two pointers and keeping them n+1 steps apart is the same as using one pointer to pass once and then passing again to the -(n+1)th position.

47 Share Reply

SHOW 5 REPLIES

henrykira ★ 221 February 10, 2019 7:25 AM

So two pointers are actually consider 'one pass' huh?
I tried 10 minutes to think out a one pointer solution...

19 Share Reply

SHOW 5 REPLIES

lindexwindy ★ 21 September 14, 2018 9:50 AM Report

this two solution almost do the same thing with same effort -- I wonder why they are treated as two different algorithms.

12 Share Reply

mdtanvir1591 ★ 27 September 27, 2018 8:26 PM

But is not this solution too visiting every node twice which is the same as the normal solution.

10 Share Reply

SHOW 1 REPLY

neo41 ★ 17 June 8, 2018 11:16 PM

Stupid question. One pass or two passes, the time complexity is the same. why bother using one pass

18 Share Reply

SHOW 5 REPLIES

conbein ★ 8 August 14, 2018 9:00 AM Report

Are not the two solutions both "two pass"? Actually, they visited the same number of nodes, length + (length -n).

8 Share Reply

SHOW 4 REPLIES

Jongtackjung ★ 8 March 27, 2018 11:44 AM

Is the second one pass?

I was constrained with the condition 'try to solve this in one pass', and the second solution doesn't really feel like a truly one pass algorithm.

Read More

6 Share Reply

SHOW 4 REPLIES