22 删除链表的倒数第 n 个节点

更新时间：2019-09-04 09:26:58

> 知识是一种快乐，而好奇则是知识的萌芽。
>
> ——培根

刷题内容

> 难度: Medium
>
> 原题链接：https://leetcode-cn.com/problems/remove-nth-node-from-end-of-list/

内容描述

给定一个链表，删除链表的倒数第 n 个节点，并且返回链表的头结点。

实例:

给定一个链表: 1->2->3->4->5, 和 n = 2.

当删除了倒数第二个节点后，链表变为 1->2->3->5.

注意：给定的 n 保证是有效的

进阶：你能尝试一下用一趟扫描实现吗？

解题方案

思路 1：时间复杂度: O(N) 空间复杂度: O(1)

根据题意，我们去移除从后数第n个元素，相当于我们要移除从前数第len(list)-n个元素。

我们利用一个指针fast，先走n个元素，这时候这个指针的位置离链表尾端就还有len(list)-n个元素了。

我们再用一个指针flow，从头开始走，指针A走一步，指针B就走一步，完美走到第len(list)-n个元素。

Python beats 98.91%

```python
class Solution(object):
    def removeNthFromEnd(self, head, n):
        """
        :type head: ListNode
        :type n: int
        :rtype: ListNode
        """
        slow = fast = dummy = ListNode(-1)
        dummy.next = head
        for i in range(n):
            fast = fast.next
        while fast.next:
            fast = fast.next
            slow = slow.next
        slow.next = slow.next.next
        return dummy.next
```

Java beats 100%

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {
        ListNode dummy = new ListNode(-1);
        dummy.next = head;
        ListNode slow = dummy;
        ListNode fast = dummy;
        for (int i = 0; i < n; i++) {
            fast = fast.next;
        }
        while (fast.next != null) {
            fast = fast.next;
            slow = slow.next;
        }
        slow.next = slow.next.next;
        return dummy.next;
    }
}
```


c++ beats 91.93 %

```cpp
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode* dummy = new ListNode(-1);
        dummy->next = head;
        ListNode *A = dummy, *B = dummy;
        for (int i = 0;i < n;i++) {
            A = A->next;
        }
        while (A->next != NULL) {
            A = A->next;
            B = B->next;
        }
        B->next = B->next->next;
        return dummy->next;
    }
};
```

go beats 100 %

```go
func removeNthFromEnd(head *ListNode, n int) *ListNode {
    dummy := &ListNode{-1, head}
    A := dummy
    B := dummy
    for i := 0;i < n;i++ {
        A = A.Next
    }
    for A.Next != nil {
        A = A.Next
        B = B.Next
    }
    B.Next = B.Next.Next
    return dummy.Next
}
```


思路 2：时间复杂度: O(N) 空间复杂度: O(1)

链表是单链表，只能从前往后遍历，题目要求移除从后数第n个元素，相当于我们要移除从前数第len(list)-n个元素，如果我们知道链表的长度，自然可以从前数遍历到第len(list)-n个元素。

Python beats 50.38%

```python
class Solution:
    def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:
        cnt = 0 # cnt 是链表的长度
        tmp = head
        while tmp:
            cnt += 1
            tmp = tmp.next
        cnt -= n # 算出从后数 n 个从前数第几个
        if cnt == 0:
            return head.next
        else:
            tmp = head
            cnt -= 1 # head 已经是第一个了，所以 cnt 减1
            while cnt != 0:
                tmp = tmp.next
                cnt -= 1
            tmp.next = tmp.next.next
            return head
```

Java beats 100%

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {
        // cnt 是链表的长度
        int cnt = 0;
        ListNode temp = head;
        while (temp != null) {
            cnt++;
            temp = temp.next;
        }
        // 算出从后数 n 个从前数第几个
        cnt = cnt - n;
        if (cnt == 0) {
            return head.next;
        }else {
            temp = head;
            // head 已经是第一个了，所以 cnt 减一
            cnt--;
            while (cnt-- != 0) {
                temp = temp.next;
            }
            temp.next = temp.next.next;
            return head;
        }
    }
}
```


c++ beats 92.96%

```cpp
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        //cnt是链表的长度
        int cnt = 0;
        ListNode* temp = head;
        while (temp != NULL) {
            cnt++;
            temp = temp->next;
        }
        //算出从后数n个从前数第几个
        cnt = cnt - n;
        if (cnt == 0) {
            return head->next;
        } else {
            temp = head;
            //head已经是第一个了，所以cnt减一
            cnt--;
            while (cnt--) {
                temp = temp->next;
            }
            temp->next = temp->next->next;
            return head;
        }
    }
};
```

go beats 100%

```go
func removeNthFromEnd(head *ListNode, n int) *ListNode {
    //cnt是链表的长度
    cnt := 0
    tmp := head
    for tmp != nil {
        cnt++
        tmp = tmp.Next
    }
    //算出从后数n个从前数第几个
    cnt = cnt - n
    if cnt == 0 {
        return head.Next
    } else {
        tmp = head
        //head已经是第一个了，所以cnt减一
        cnt--
        for cnt > 0 {
            cnt--
            tmp = tmp.Next;
        }
        tmp.Next = tmp.Next.Next;
        return head;
    }
}
```

进阶: 你能尝试一下用一趟扫描实现吗？

思路 3：时间复杂度: O(N) 空间复杂度: O(1)

一趟扫描，时间复杂度限制在o(n)。

注意到我们前面的思路2，我们先将指针fast移动n个元素，再将指针fast和指针slow一起移动len(list)-n个元素，这两个步骤加起来，fast指针从链表开头移动到链表末端。

我们可以将两个步骤合起来，fast移动n个元素前，slow不动；移动了n个元素后，slow跟着一起移动，于是我们实现了用一趟扫描就解决问题。

Python beats 98.91%

```python
class Solution(object):
    def removeNthFromEnd(self, head, n):
        """
        :type head: ListNode
        :type n: int
        :rtype: ListNode
        """
        slow = fast = dummy = ListNode(-1)
        dummy.next = head
        count = 0
        while fast.next:
            if count < n:
                count += 1
                fast= fast.next
            else:
                fast = fast.next
                slow = slow.next
        slow.next = slow.next.next
        return dummy.next
```

Java beats 100%

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {
        ListNode dummy = new ListNode(-1);
        dummy.next = head;
        ListNode slow = dummy;
        ListNode fast = dummy;
        int count = 0;
        while (fast.next != null) {
            if (count < n) {
                count++;
                fast = fast.next;
            } else {
                fast = fast.next;
                slow = slow.next;
            }
        }
        slow.next = slow.next.next;
        return dummy.next;
    }
}
```

go beats 100 %


```go
func removeNthFromEnd(head *ListNode, n int) *ListNode {
 dummy := &ListNode{-1, head}
 A := dummy
 B := dummy
 count := 0
 for A.Next != nil {
  if count < n {
   A = A.Next
  } else {
   A = A.Next
   B = B.Next
  }
  count++;
 }
 B.Next = B.Next.Next
 return dummy.Next
}
```

c++ beats 87.03%

```
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode* dummy = new ListNode(-1);
        dummy->next = head;
        ListNode *A = dummy, *B = dummy;
        int count = 0;
        while (A->next != NULL) {
            if (count < n) {
                A = A->next;
            } else {
                A = A->next;
                B = B->next;
            }
            count++;
        }
        B->next = B->next->next;
        return dummy->next;
    }
};
```

小结

这几个算法的时间复杂度虽然都是o(n)，但写法不一样，系数就不一样。我们在时间复杂度到极限时，就会去苛求代码细节以降低复杂度系数。思路2是我们最直接的想法，需要扫描一遍记录长度。思路1不需要记录长度，用一个指针扫描一遍链表，实际上相当于思路1的长度，这种做法很巧妙，代码量小，控制两个指针走就可以解决问题，不需要其它的操作。进阶思路3就是思路1的改进版，将两个循环合并成一个循环，满足题目所说，一遍扫描就解决。

}

21 电话号码的字母组合
23 括号生成