摘要

本文适用于初学者。它介绍了以下内容： 链表的遍历和删除其末尾的第 n 个元素。

解决方案

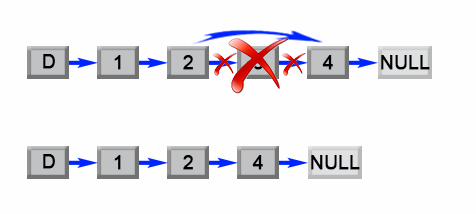
方法一：两次遍历算法

**思路**

我们注意到这个问题可以容易地简化成另一个问题：删除从列表开头数起的第 (L - n + 1)个结点，其中 *L* 是列表的长度。只要我们找到列表的长度 *L*，这个问题就很容易解决。

**算法**

首先我们将添加一个哑结点作为辅助，该结点位于列表头部。哑结点用来简化某些极端情况，例如列表中只含有一个结点，或需要删除列表的头部。在第一次遍历中，我们找出列表的长度 L*L*。然后设置一个指向哑结点的指针，并移动它遍历列表，直至它到达第 (*L*−*n*) 个结点那里。我们把第 (*L*−*n*) 个结点的 next 指针重新链接至第 (*L*−*n*+2) 个结点，完成这个算法。



*图 1. 删除列表中的第 L - n + 1 个元素*

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;

}

**复杂度分析**

* 时间复杂度：*O*(*L*)，

该算法对列表进行了两次遍历，首先计算了列表的长度 *L* 其次找到第 (*L*−*n*) 个结点。 操作执行了2*L*−*n* 步，时间复杂度为 *O*(*L*)。

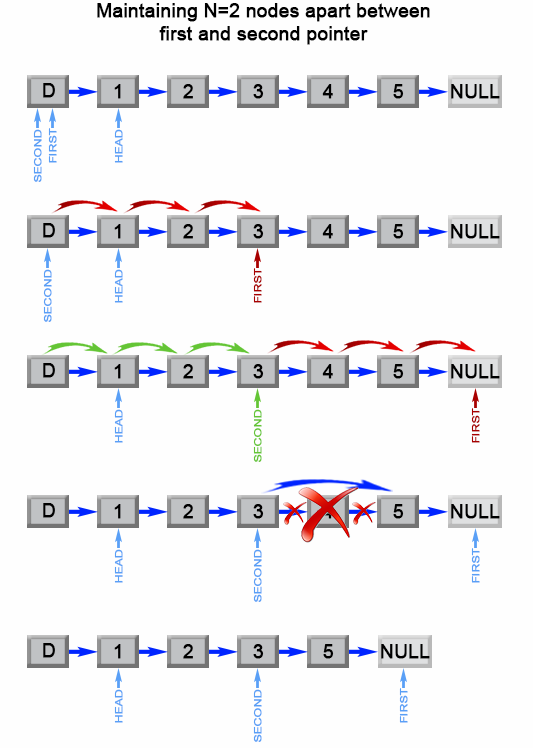
* 空间复杂度：*O*(1)，

我们只用了常量级的额外空间。 

方法二：一次遍历算法

**算法**

上述算法可以优化为只使用一次遍历。我们可以使用两个指针而不是一个指针。第一个指针从列表的开头向前移动 *n*+1 步，而第二个指针将从列表的开头出发。现在，这两个指针被 *n* 个结点分开。我们通过同时移动两个指针向前来保持这个恒定的间隔，直到第一个指针到达最后一个结点。此时第二个指针将指向从最后一个结点数起的第 *n* 个结点。我们重新链接第二个指针所引用的结点的 next 指针指向该结点的下下个结点。



*图 2. 删除链表的倒数第 N 个元素*

*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;*

*}*

**复杂度分析**

* 时间复杂度：*O*(*L*)，

该算法对含有 *L* 个结点的列表进行了一次遍历。因此时间复杂度为 *O*(*L*)。

* 空间复杂度：*O*(1)，

我们只用了常量级的额外空间。