删除倒数第N个节点\_LeetCode\_19\_RemoveNthNodeFromEndOfList\_Medium

# 删除倒数第N个节点\_LeetCode\_19\_RemoveNthNodeFromEndOfList\_Medium

## 题目介绍

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*LeetCode\_19\_RemoveNthNodeFromEndOfList\_Medium\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* LeetCode\_19\_RemoveNthNodeFromEndOfList\_Medium\_删除倒数第N个节点

\* https://leetcode.com/problems/remove-nth-node-from-end-of-list/description/

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

\* 难度：Medium

\*

\* 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?

## 思路分析

\* 思路分析：双指针法+临时头节点。

\* 双指针一个在前front，一个在后behind，两者相差n+1步；

\* 目的是当front==null时，behind正好是待删除节点的前一个节点。

\* 怎么保证front和behind之间相差n+1步呢？

\* 让behind后退一步，front向前走n步即可。

\* 为什么需要临时头节点呢,也就是为什么需要behind后退一步呢？

\*当n等于链表长度时，如1->2,n=2;删除的就是原来的头节点1，此时方法必须返回2，

\* 而front先走2步，front==null;而behind不动，若一开始初始化为head,删除的将是head.next;

\* 即删除的是2，而让behind后退一步，删除的正好是1，返回dummy.next即可。

## Java代码

/\*\*

\* 双指针法

\*/

public ListNode **removeNthFromEnd**(ListNode head, int n) {

if (head == null || n < 1) return null;

//dummy意思为傀儡，仿制品;这里意思为傀儡头节点

ListNode dummy = new ListNode(-1);//相当于behind后退一步

dummy.next = head;

ListNode front = head, behind = dummy;

while (n-- > 0) front = front.next;//front 先走n步

while (front != null) {

front = front.next;

behind = behind.next;

}

//删除behind后一个节点

behind.next = behind.next.next;

return dummy.next;

}