# 题目

输入一个链表，输出该链表中倒数第k个节点。为了符合大多数人的习惯，本题从1开始计数，即链表的尾节点是倒数第1个节点。例如，一个链表有6个节点，从头节点开始，它们的值依次是1、2、3、4、5、6。这个链表的倒数第3个节点是值为4的节点。

**示例：**

给定一个链表: 1->2->3->4->5, 和 k = 2.

返回链表 4->5.

类似题目：面试题 02.02（返回值不同）

# 分析

具体方法参考《程序员面试手册：概念、编程问题及面试题》 P276

## 方法一：暴力法

从首个节点开始，计算该节点之后的节点数量。如果数量小于n-1，那么算法就结束，并告诉调用方，链表中没有那么多节点。如果大于n-1，那就移动到下一个节点，并计算那个节点之后的节点的数量，直到计算结果恰好等于n-1为止。此时的节点，就是链表的倒数第n个节点。

时间复杂度：由于每经过一个节点就要从该节点开始，计算其后的节点数量，因此时间复杂度为O(n2)。

空间复杂度：O(1)。

## 方法二：哈希表

采用哈希表可以在方法一的基础上降低复杂度。

采用这种方法求解时，需要创建哈希表，并把节点的位置以及该节点地址视为一对键值，放在表格中，哈希表的键就是节点在链表中的位置，而值就是该节点的地址。

把链表遍历一遍后，就可以确定其长度。如果长度为M，那么倒数第n个节点就相当于正数的第M-n+1个节点。由于我们已经知道链表的长度，因此，只需要把第M-n+1个键锁对应的哈希表里面取出即可。

时间复杂度：由于创建哈希表需要一定时间，因此T(m)=O(m)。

空间复杂度：由于创建大小为m的哈希表，因此空间复杂度为O(m)。

## 方法三：二次扫描

## 方法四：双指针

**思路：**

**代码：**

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

ListNode\* getKthFromEnd(ListNode\* head, int k) {

ListNode \*former = head;

ListNode \*latter = head;

for(int i=0;i<k;i++)

{

former = former->next;

}

while(former)

{

former = former->next;

latter = latter->next;

}

return latter;

}

};

复杂度分析：

时间复杂度O(N) ：N为链表长度；总体看，former走了N步，latter走了(N-k)步。

空间复杂度O(1) ：双指针former , latter使用常数大小的额外空间。