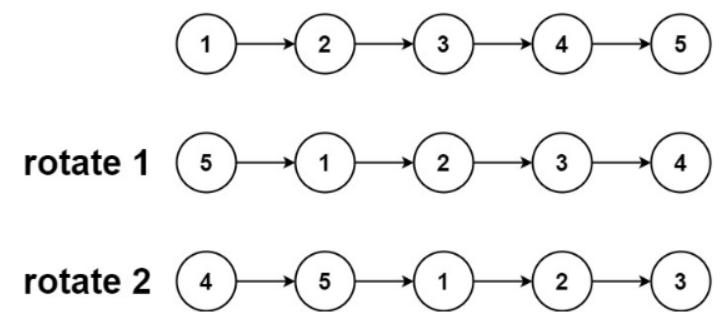
<https://leetcode.com/problems/rotate-list/>

Given the head of a linked list, rotate the list to the right by k places.

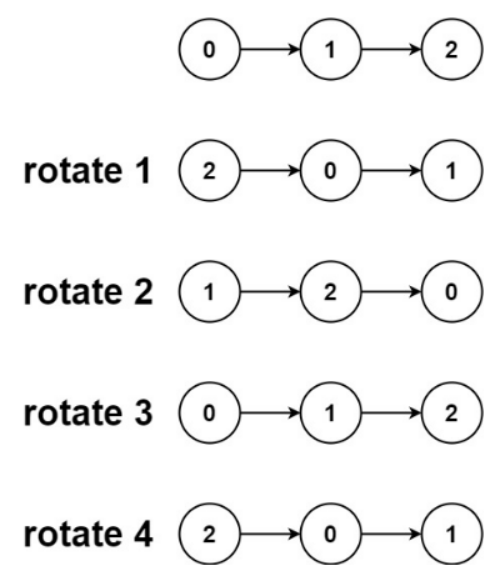
**Example 1:**



Input: head = [1,2,3,4,5], k = 2

Output: [4,5,1,2,3]

**Example 2:**



Input: head = [0,1,2], k = 4

Output: [2,0,1]

**Constraints:**

* The number of nodes in the list is in the range [0, 500].
* -100 <= Node.val <= 100
* 0 <= k <= 2 \* 109

**Attempt 1: 2023-02-21**

**Solution 1: Find length and new head (10 min)**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode rotateRight(ListNode head, int k) {

if(head == null || head.next == null) {

return head;

}

// Find length

ListNode iter = head;

// This way not convenient because besides get length,

// we also have to get 'tail' node and use 'tail' to

// connect original 'head' to build rotation first

//int len = 0;

//while(iter != null) {

// iter = iter.next;

// len++;

//}

int len = 1;

while(iter.next != null) {

iter = iter.next;

len++;

}

// Connect 'iter'(point to tail node now) to original

// 'head' prepare for rotation cut

iter.next = head;

// Find prior node of new head

ListNode iter1 = head;

int i = 0;

while(i < len - k % len - 1) {

iter1 = iter1.next;

i++;

}

ListNode prev = iter1;

ListNode newHead = prev.next;

// Cut between prior node and new head

prev.next = null;

return newHead;

}

}

**Refer to**

<https://leetcode.com/problems/rotate-list/solutions/22735/my-clean-c-code-quite-standard-find-tail-and-reconnect-the-list/>

There is no trick for this problem. Some people used slow/fast pointers to find the tail node but I don't see the benefit (in the sense that it doesn't reduce the pointer move op) to do so. So I just used one loop to find the length first.

class Solution {

public:

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

if(!head) return head;

int len=1; // number of nodes

ListNode \*newH, \*tail;

newH=tail=head;

while(tail->next) // get the number of nodes in the list

{

tail = tail->next;

len++;

}

tail->next = head; // circle the link

if(k %= len)

{

for(auto i=0; i<len-k; i++) tail = tail->next; // the tail node is the (len-k)-th node (1st node is head)

}

newH = tail->next;

tail->next = NULL;

return newH;

}

};