题目地址(61. 旋转链表)

https://leetcode-cn.com/problems/rotate-list/

入选理由

1. 难度低，适合链表开篇

2. 考察频率高，不瞒您说，我在面试中就被问到过

标签

- 链表

难度

- 简单

前置知识

-求单链表的倒数第 N 个节点

题目描述

```
给定一个链表，旋转链表，将链表每个节点向右移动 k 个位置，其中 k 是非负数。

示例 1:

输入: 1->2->3->4->5->NULL, k = 2
输出: 4->5->1->2->3->NULL
解释:
向右旋转 1 步: 5->1->2->3->4->NULL
向右旋转 2 步: 4->5->1->2->3->NULL
示例 2:

输入: 0->1->2->NULL, k = 4
输出: 2->0->1->NULL
解释:
向右旋转 1 步: 2->0->1->NULL
向右旋转 2 步: 1->2->0->NULL
```

```
向右旋转 3 步: 0->1->2->NULL
向右旋转 4 步: 2->0->1->NULL
```

思路

首先我们看下如何返回链表倒数第 k 个节点。

1. 采用快慢指针
2. 快指针与慢指针都以每步一个节点的速度向后遍历
3. 快指针比慢指针先走 k 步
4. 当快指针到达终点时，慢指针正好是倒数第 k 个节点

伪代码：

```
快指针 = head;
慢指针 = head;
while (快指针.next) {
  if (k-- <= 0) {
    慢指针 = 慢指针.next;
  }
  快指针 = 快指针.next;
}
```

我们将上面的代码改为真实代码。

JS Code:

```
let slow = (fast = head);
while (fast.next) {
  if (k-- <= 0) {
    slow = slow.next;
  }
  fast = fast.next;
}
```

有了上面的知识，我们来看下具体如何解决这道题。

算法描述：

1. 获取单链表的倒数第 1 （尾节点）与倒数第 2 个节点
2. 将倒数第 2 个节点的 next 指向 null
3. 将尾节点的 next 指向 head（拼起来）

4. 返回倒数第 1 个节点

经过这样的处理，我们**旋转了一位**，而题目是要旋转 k 位，实际上我们只需要将上面的算法微调即可。将 1 改成 k ， 2 改成 k + 1。

算法描述：

1. 获取单链表的倒数第 k 与倒数第 k + 1 个节点
2. 将倒数第 k + 1 个节点的 next 指向 null
3. 将尾节点 next 指向 head（拼起来）
4. 返回倒数第 k 个节点

例如链表 A -> B -> C -> D -> E 右移 2 位，依照上述步骤为：

1. 获取节点 C 与 D
2. A -> B -> C -> null, D -> E
3. D -> E -> A -> B -> C -> nul
4. 返回节点 D

注意：假如链表节点长度为 len， 则右移 K 位与右移动 k % len 的效果是一样的 就像是长度为 1000 米的环形跑道， 你跑 1100 米与跑 100 米到达的是同一个地点

据此不难写出如下伪代码：

```
获取链表的长度
k = k % 链表的长度
获取倒数第k + 1,倒数第K个节点与链表尾节点
倒数第k + 1个节点.next = null
链表尾节点.next = head
return 倒数第k个节点
```

代码

代码支持：JS，Java, Python，CPP

```
var rotateRight = function (head, k) {
  if (!head || !head.next) return head;
  let count = 0,
    now = head;
  while (now) {
```

```
    now = now.next;
    count++;
  }
  k = k % count;
  let slow = (fast = head);
  while (fast.next) {
    if (k-- <= 0) {
      slow = slow.next;
    }
    fast = fast.next;
  }
  fast.next = head;
  let res = slow.next;
  slow.next = null;
  return res;
};
```

Java Code:

```
class Solution {
    public ListNode rotateRight(ListNode head, int k) {
        if(head == null || head.next == null) return head;
        int count = 0;
        ListNode now = head;
        while(now != null){
            now = now.next;
            count++;
        }
        k = k % count;
        ListNode slow = head, fast = head;
        while(fast.next != null){
            if(k-- <= 0){
                slow = slow.next;
            }
            fast = fast.next;
        }
        fast.next = head;
        ListNode res = slow.next;
        slow.next = null;
        return res;
    }
}
```

Python Code:

```
class Solution:
    def rotateRight(self, head: ListNode, k: int) -> ListNode:
        # 双指针
        if head:
            p1 = head
```

```
        p2 = head
        count = 1
        i = 0
        while i < k:
            if p2.next:
                count += 1
                p2 = p2.next
            else:
                k = k % count
                i = -1
                p2 = head
            i += 1

        while p2.next:
            p1 = p1.next
            p2 = p2.next

        if p1.next:
            tmp = p1.next
        else:
            return head
        p1.next = None
        p2.next = head
        return tmp
```

C++ Code

```
ListNode* rotateRight(ListNode* head, int k) {
    if (head == nullptr
        || head->next == nullptr
        || k == 0)
        return head;

    int len = 1;
    ListNode* cur = head;
    while (cur->next != nullptr) {
        cur = cur->next;
        len++;
    }

    k %= len;

    ListNode* fast = head;
    ListNode* slow = head;

    while (fast->next != nullptr) {
        if (k-- <= 0) {
            slow = slow->next;
        }
        fast = fast->next;
    }
```

```
    fast->next = head;
    ListNode* new_head = slow->next;
    slow->next = nullptr;
    return new_head;
}
```

复杂度分析

- 时间复杂度：节点最多只遍历两遍，时间复杂度为$O(n)$
- 空间复杂度：未使用**额外**的空间，空间复杂度$O(1)$

上一页 下一页