链表逆序\_LeetCode\_206\_从尾到头打印链表\_剑指Offer\_6

# 反转链表ReverseList

## 题目介绍

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*链表T1：反转链表\_LeetCode\_206. Reverse Linked List\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* 链表T1：反转链表\_LeetCode\_206. Reverse Linked List

\* 题目描述 链表T1：反转链表

\* 输入一个链表，反转链表后，输出新链表的表头。

\* Reverse a singly linked list.

\* Example:

\* Input: 1->2->3->4->5->NULL

\* Output: 5->4->3->2->1->NULL

\* Follow up:

\* A linked list can be reversed either iteratively or recursively.

\* Could you implement both?



## 思路分析

\* 思路分析：两种实现：1、循环实现；2、递归实现。

\* 1. 循环实现。需要三个引用，前一个节点preNode，当前节点currentNode，

\* 后一个节点nextNode；断开链表之前，暂时保存下一节点nextNode；

\* 在头节点head之前引入null，作为head的pre节点。

## Java代码

### While循环实现

/\*\*

\* iteratively : while循环实现

\*/

public ListNode reverseList(ListNode head) {

// if(head == null||head.next == null) return head;//下面的已经处理这种特殊情况

ListNode currentNode = head,preNode = null;

while(currentNode != null){

ListNode nextNode = currentNode.next;

currentNode.next = preNode;

preNode = currentNode;

currentNode = nextNode;

}

return preNode;

}



### 递归实现

/\*\*

\* recursively : 递归实现

\*/

public ListNode reverseList\_recursive(ListNode head) {

if(head == null||head.next == null) return head;

return reverseOne(null,head);

}

/\*\*

\* 该方法递归实现逆序整个链表

\* @param preTail : 已经逆序好的尾节点

\* @param head：剩下链表的头节点

\* @return ： 返回逆序后链表的头节点

\*/

public ListNode reverseOne(ListNode preTail,ListNode head){

if(head == null) return preTail;

ListNode next = head.next;

head.next = preTail;

preTail = head;

return reverseOne(preTail,next);

}

# 反转链表II-升级版

## 题目介绍

\* LeetCode\_92\_Reverse\_Linked\_List\_II\_反转链表II-升级版

\* 难度：Medium

\* DateTime：2018-10-06

\* https://leetcode.com/problems/reverse-linked-list-ii/description/

\*

\* 题目介绍：

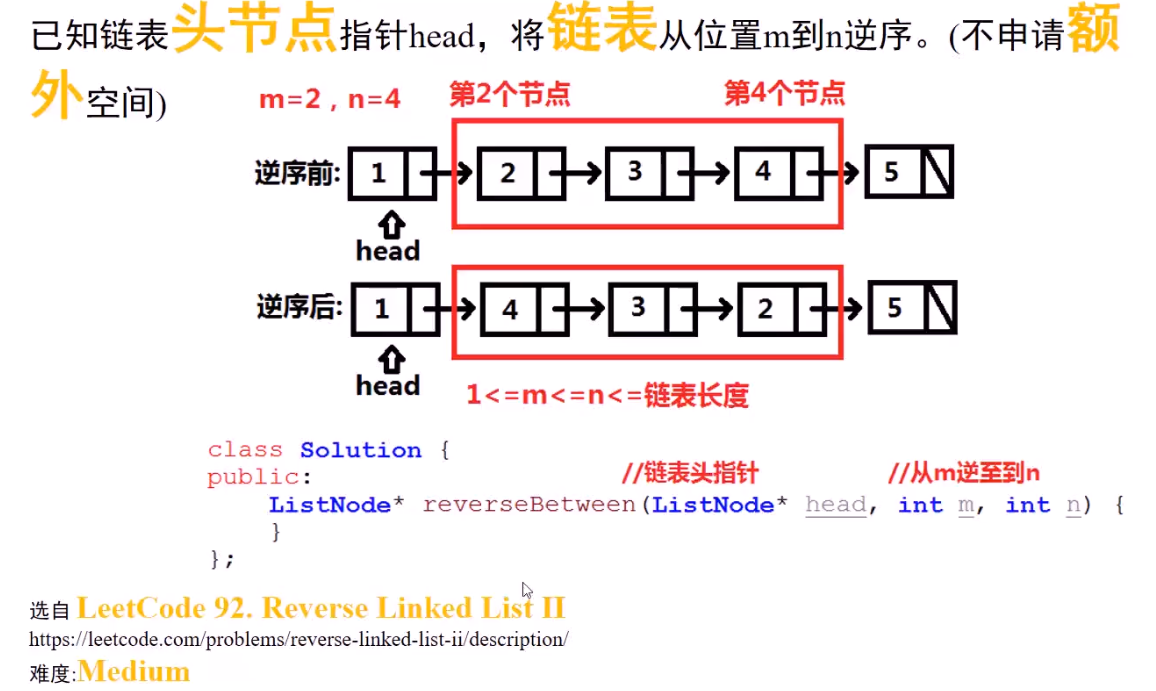
\* Reverse a linked list from position m to n. Do it in one-pass.

\* Note: 1 ≤ m ≤ n ≤ length of list.

\* Example:

\* Input: 1->2->3->4->5->NULL, m = 2, n = 4

\* Output: 1->4->3->2->5->NULL



## 思路分析

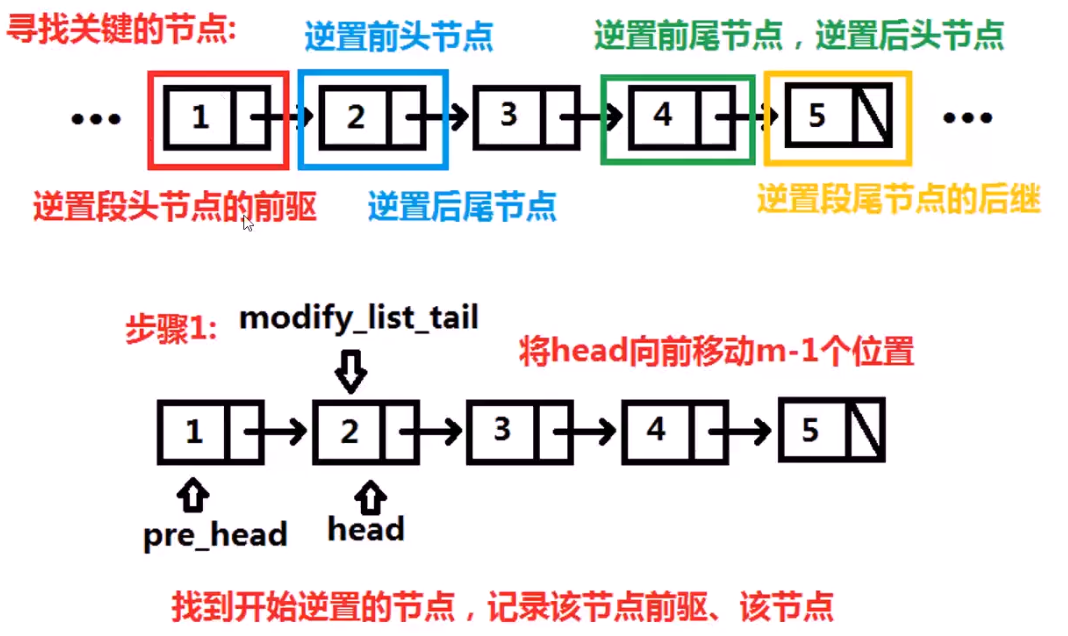
\* 思路分析：

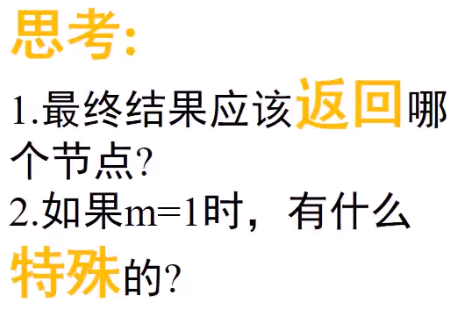
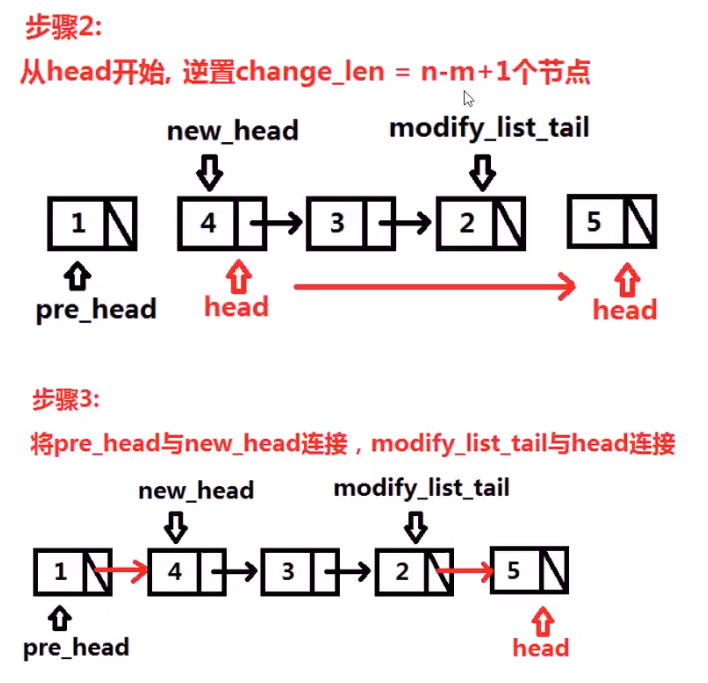
\* 步骤分割：第一步：首先前进m-1步，到达第m节点；

\* 第二步：反转m->n节点;

\* 第三步：反转链表的头尾处理。

\* 两种实现。while循环实现。(递归不适合）





## Java代码

### 整体

/\*\*

\* 循环实现

\*/

public ListNode reverseBetween(ListNode head, int m, int n) {

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

ListNode head1 = head;

ListNode preTail = null;//需要逆置节点的前驱节点

//第一步：前进至第m节点，即行走m-1步;注意考虑m为1的情况，此时preTail=null

int i = 1;

while(head1 != null && i++ < m){

preTail = head1;

head1 = head1.next;

}

//第二步：反转m->n节点

if(head1 == null) return head;//head1为null，不需要后翻转了

ListNode preNode = null;

ListNode reverseListPreHead = head1;

ListNode nextNode;

while (head1 != null && m++ <= n) {

nextNode = head1.next;

head1.next = preNode;

preNode = head1;

head1 = nextNode;

}

//第三步：反转链表的头尾处理

reverseListPreHead.next = head1;//处理尾巴

if (preTail != null) {//处理头部

preTail.next = preNode;

return head;

}else{

return preNode;//若preTail为null,返回翻转链表段的最后一个节点

}

}



### 拆分成两部分

/\*\*

\* 循环实现:将上述整体中翻转部分节点部分拆分出来，原理一样的。

\*/

public ListNode reverseBetween\_2(ListNode head, int m, int n) {

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

ListNode head1 = head;

ListNode preTail = null;//需要逆置节点的前驱节点

//第一步：前进至第m节点，即行走m-1步;注意考虑m为1的情况，此时preTail=null

int i = 1;

while(head1 != null && i++ < m){

preTail = head1;

head1 = head1.next;

}

//第二步：反转m->n节点

if(head1 == null) return head;//head1为null，不需要后翻转了

ListNode reverseListTail = reverseListBetween(head1,m,n);

//第三步：反转链表的头处理

if (preTail != null) {//处理头部

preTail.next = reverseListTail;

return head;

}else{

return reverseListTail;//若preTail为null,返回翻转链表段的最后一个节点

}

}

/\*\*

\* while循环实现的翻转部分

\* @param head1

\* @return 返回的是翻转链表的最后一个节点，也就是新的头节点

\*/

public ListNode reverseListBetween(ListNode head1,int m,int n) {

ListNode reverseListPreHead = head1;

ListNode preNode = null;

ListNode nextNode;

while (head1 != null && m++ <= n) {

nextNode = head1.next;

head1.next = preNode;

preNode = head1;

head1 = nextNode;

}

reverseListPreHead.next = head1;//处理尾巴

return preNode;

}

# 从尾到头打印链表\_剑指Offer\_6

## 题目介绍

/\*\*\*\*\*\*剑指Offer6:从尾到头打印链表\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* 题目描述

\* 输入一个链表，按链表值从尾到头的顺序返回一个ArrayList。

## 思路分析

\* 方法1：先反转链表，然后打印；前提：允许反转链表。(不建议)

\* 方法2：利用栈的FILO特点；

\* 方法3：递归。注意递归深度。（不建议）

## Java代码

### 利用栈实现

/\*\*

\* 方法2：利用栈实现

\*/

public ArrayList<Integer> **printListFromTailToHead2**(ListNode listNode) {

ArrayList<Integer> result = new ArrayList<Integer>();

Deque<Integer> stack = new ArrayDeque<Integer>();

ListNode currentNode = listNode;

while(currentNode != null){

**stack.push(currentNode.val);//压栈**

currentNode = currentNode.next;

}

**while(!stack.isEmpty()) result.add(stack.pop());//依次从栈中取出**

return result;

}

### 利用递归方法

/\*\*

\* 方法3： 递归方法实现

\*/

public ArrayList<Integer> printListFromTailToHead3(ListNode listNode) {

ArrayList<Integer> result = new ArrayList<Integer>();

recursiveMethod(listNode,result);

return result;

}

/\*\*

\* 递归辅助方法

\*/

public void recursiveMethod(ListNode listNode,ArrayList<Integer> result){

if(listNode == null) return;

recursiveMethod(listNode.next,result);

result.add(listNode.val);

}

### 利用反转链表实现

/\*\*

**\* 方法1： 反转链表**

\*/

public ArrayList<Integer> printListFromTailToHead1(ListNode listNode) {

ArrayList<Integer> result = new ArrayList<Integer>();

ListNode preNode = null,currentNode = listNode;

while(currentNode != null){

ListNode nextNode = currentNode.next;

currentNode.next = preNode;

preNode = currentNode;

currentNode = nextNode;

}

while(preNode != null) {

result.add(preNode.val);

preNode = preNode.next;

}

return result;

}