剑指Offer\_关于链表的简单算法题目

# 链表T1：反转链表

## 题目介绍

\* 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。

## 思路分析

\* 思路分析：需要三个引用，断开链表之前，保存下一节点；

\* 在头节点head之前引入null，作为head的pre节点。

## Java代码

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;

}

# 链表T2：删除链表的节点

## 题目介绍

在O(1)时间内删除链表节点。

## 思路分析

\* 思路分析:若遍历寻找待删除的节点前一节点，则时间复杂度为O(n)。

\* 实现方式：**复制后一节点到该节点,然后只需要删除后一节点即可。**

\* 特殊情况：待删除节点是尾节点，①若只有1个节点的情况；②多个节点，该节点是尾节点。

\* //注意：这里假定链表中肯定存在待删除的节点。

## Java代码

public void deleteListNode(ListNode head,ListNode toBeDeleted){

if(head == null || toBeDeleted == null) return;

if(toBeDeleted.next != null){//非尾节点

**ListNode next = toBeDeleted.next;**

**toBeDeleted.val = next.val;**

**toBeDeleted.next = next.next;**

}else{//尾节点

if(head == toBeDeleted){//只有一个节点

**head = null;**

toBeDeleted = null;

}else{//多个节点，只能顺序查找待删除节点的前一节点了

ListNode head1 = head;

**while(head1.next != toBeDeleted){**

**head1 = head1.next;**

**}**

**head1.next = null;**

toBeDeleted = null;

}

}

}

# 链表T3：删除链表中重复的节点deleteDuplication

## 题目介绍

\* 题目描述：**删除链表中重复的节点**。(注意：已经排序了)

\* 在一个排序的链表中，存在重复的结点，请删除该链表中重复的结点，

\* 重复的结点不保留，返回链表头指针。

\* 例如，链表1->2->3->3->4->4->5 处理后为 1->2->5

## 思路分析

\* 思路分析：是排序链表，因此重复元素肯定是相邻的；

\* 和反转链表类似，需要三个引用：preNode、currentNode、nextNode。

\* 当前节点和下一节点的值作判断，若相等，则继续判断，直到找到不相等的节点或null，

\* 让preNode指向nextNode(这里preNode可能为null，所以需要判断是否null，null即表示开头即重复元素)，

\* 若不相等，继续后移即可。

## Java代码

public ListNode deleteDuplication(ListNode head){

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

ListNode preNode = null;

ListNode currentNode = head;

while(currentNode != null){

ListNode nextNode = currentNode.next;

if(nextNode == null) return head;

**if(nextNode.val == currentNode.val){//出现重复节点**

nextNode = nextNode.next;

**while(nextNode != null && nextNode.val == currentNode.val){**

**nextNode = nextNode.next;**

**}**

if(preNode == null) {//头节点就是重复节点

head = nextNode;

}else{

preNode.next = nextNode;

}

//preNode 不变，currentNode指向nextNode

**currentNode = nextNode;**

**}else{//不重复**

preNode = currentNode;

currentNode = nextNode;

}

}

return head;

}

# 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;

}