删节点\_删除链表中的重复节点LeetCode\_83\_I\_82\_II\_JianzhiOffer\_18

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*链表：删除链表节点\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* T1:在O(1)时间内删除链表节点。

\* T2：删除有序链表中重复的节点,LeetCode\_83\_RemoveDuplicatesFromSortedList

\*T3：删除有序链表中所有重复的节点,LeetCode\_82\_RemoveDuplicatesFromSortedList\_II

\*/

# T1：在O(1)时间内删除链表节点

## 题目介绍

/\*\*

\* 题目描述：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)，不符合要求。

\* O(1)的实现方式：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

//解释为head=null不起作用？ 因为Java方法只有值传递，对于引用传递都是地址，

//形参与实参都指向同一个对象地址，形参试图改变指向地址，不改变实参；

//但是形参和实参对该指向的对象的操作是等价的，比如形参引用可以改变对象的属性。

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;

}

}

}

# T2：删除有序链表中重复的节点I

## 题目介绍

\* LeetCode\_83\_RemoveDuplicatesFromSortedList

\* 难度：Easy

\* https://leetcode.com/problems/remove-duplicates-from-sorted-list/description/

\*

\* 题目描述：删除有序链表中重复的节点。(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。

\* Given a sorted linked list, delete all duplicates

\* such that each element appear only once.

\* Example 1:

\* Input: 1->1->2 Output: 1->2

\* Example 2:

\* Input: 1->1->2->3->3 Output: 1->2->3

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

## 思路分析

### 思路1：

\* 思路1：保存重复节点片段的头节点，遍历后面，直至遇到第二个重复节点片段的头节点，然后更新操作。

\* 是排序链表，因此重复元素肯定是单个或连续几个；1个也可以统一到多个中。

\* 对于多个连续的相同值的节点，利用duplFirst保存第一个节点，然后利用currNode遍历后续节点，

\* 直至遇到一个非相同值的节点，此时，将duplFirst.next指向该节点，然后令duplFirst=duplFirst.next;

\* 表示遇到了一个新的连续相同节点的开端。

\* 具体实现：

\* 首先，head为null，则直接返回null；

\* 其次，遍历链表，遇到不相同值的节点，则更新duplFirst；

\* 最后，一定要duplFirst.next = null；因为对于最后单个节点无影响，

\* 但是最后结尾的是多个重复值节点时，如->5->5这种情况。

### 思路2：

\* 思路2：保持当前节点始终为一个重复节点片段的头节点，若后一个节点与currNode的值相同，

\* 则令currNode.next指向下一个节点的下一个，即跳过下一个节点，然后不断的遍历下一个nextNode，

\* 直到遇到不同的节点，更新currNode=nextNode。

## Java代码

### 思路1：

/\*\*

\* 思路1：保存重复节点片段的头节点，遍历后面，直至遇到第二个重复节点片段的头节点，然后更新操作。

\*/

public ListNode deleteDuplicates(ListNode head) {

if (head == null) return null;

ListNode duplFirst = head;//重复节点的第一个

ListNode currNode = head.next;//从第二个节点开始遍历

while (currNode != null) {

if (duplFirst.val != currNode.val) {

duplFirst.next = currNode;

duplFirst = duplFirst.next;

}

currNode = currNode.next;

}

duplFirst.next = null;//最后一个节点要设置null

return head;

}

### 思路2：

/\*\*

\* 第二种想法：不断更新currNode的下一个节点。

\*/

public ListNode deleteDuplicates2(ListNode head) {

if (head == null) return null;

ListNode currNode = head;

ListNode nextNode = head.next;

while (nextNode != null) {

if (nextNode.val == currNode.val) {

currNode.next = nextNode.next;//直接跳过后一个值相同的节点

} else {//遇到值不相同的节点

currNode = nextNode;

}

nextNode = currNode.next;

}

return head;

}

# T3：删除有序链表中所有重复的节点II

## 题目介绍

\* LeetCode\_82\_RemoveDuplicatesFromSortedList\_II

\* 难度：Medium

\* https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/description/

\* DateTime:2018-10-07

\*

\* 题目描述：删除有序链表中所有重复的节点，只保留不重复的节点。(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，

\* 重复的结点一个也不保留，返回链表头指针。

\*

\* Given a sorted linked list, delete all nodes that have duplicate numbers,

\* leaving only distinct numbers from the original list.

\*

\* Example 1:

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

\* Output: 1->2->5

\*

\* Example 2:

\* Input: 1->1->1->2->3

\* Output: 2->3

## 思路分析

### **方法1：左右判断法**。

\* 主要思路：currNode与上一个删除节点和后一个节点都不相同的时候，才保留，否则就删除当前节点。

\* S1：设置临时头节点。构造一个临时头节点tempHead,该节点值任意。

\* S2：while循环删除所有重复节点。对于第一个head节点，前面没有已经删除的元素，deleteNode为null，所有判断时，若deleteNode == null，

\* 说明currNode 与deleteNode值不相同。<不能初始化deleteNode为tempHead的值，因为链表只是说有序，

\* 但是没有说是升序还是降序，无法随意指定tempHead的值。经测试：MAX\_VALUE、或MIN\_VALUE或head.val+/-1都不可以>

\* S3：处理最后一个节点。最后一个节点currNode.next==null,已经跳出循环，因此需要单独处理。

\* 若最后一个currNode的值与deleteNode不相同，则preNode.next = currNode即可，否则preNode.next = null;

### **方法2：嵌套while循环**。

\* S1：设置临时头节点。构造一个临时头节点tempHead,该节点值任意。

\* S2：遍历节点。三个变量：preNode、currNode、nextNode

\* 循环体内，判断currNode与nextNode值是否相等，

\* 若不相等，则正常往下走；

\* 若相等，则利用嵌套while循环找到相等的最后一个节点，处理一下，直接跳过这一段相同值的节点。

\*

\* 综上：方法2更好理解，建议掌握好方法2.

## Java代码

### 方法1：左右判断法。

public ListNode deleteDuplicatesII(ListNode head) {

if (head == null) return null;

//S1:设置临时头节点

ListNode tempHead = new ListNode(0);//临时头节点

tempHead.next = head;

//S2：while循环删除所有重复节点

ListNode preNode = tempHead;

ListNode deletedNode = null;

ListNode currNode = head;

while (currNode.next != null) {

if ((deletedNode != null && currNode.val == deletedNode.val) ||

currNode.val == currNode.next.val) {

deletedNode = currNode;

} else {

preNode.next = currNode;//删除当前节点

preNode = preNode.next;

}

currNode = currNode.next;

}

//处理最后一个节点currNode

if (deletedNode != null && currNode.val == deletedNode.val) {

preNode.next = null;

} else {

preNode.next = currNode;

}

return tempHead.next;

}

### 方法2：嵌套while循环法。

public ListNode deleteDuplicatesII2(ListNode head) {

if (head == null) return null;

//S1:设置临时头节点

ListNode tempHead = new ListNode(0);//临时头节点

tempHead.next = head;

//S2:嵌套循环

ListNode preNode = tempHead;

ListNode currNode = head;

// ListNode nextNode = currNode.next;//单独设置这个nextNode会带来很多麻烦不如直接用currNode.next替代；

//原因：因为currNode和nextNode都要往后走一步，只需要currNode = currNode.next;即可。

//两个变量的时候，还需要nextNode = currNode.next;这时候无法保证currNode是不是非null的。

while(currNode != null && currNode.next != null){

if(currNode.val != currNode.next.val){

preNode = currNode;

currNode = currNode.next;

}else{

currNode = currNode.next;

while(currNode.next != null && currNode.val == currNode.next.val) {

currNode = currNode.next;

}

currNode = currNode.next;

preNode.next = currNode;

}

}

return tempHead.next;

}