方法：直接法

**算法**

这是一个简单的问题，仅测试你操作列表的结点指针的能力。由于输入的列表已排序，因此我们可以通过将结点的值与它之后的结点进行比较来确定它是否为重复结点。如果它是重复的，我们更改当前结点的 next 指针，以便它跳过下一个结点并直接指向下一个结点之后的结点。

public ListNode deleteDuplicates(ListNode head) {

ListNode current = head;

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

if (current.next.val == current.val) {

current.next = current.next.next;

} else {

current = current.next;

}

}

return head;

}

**复杂度分析**

* 时间复杂度：*O*(*n*)，因为列表中的每个结点都检查一次以确定它是否重复，所以总运行时间为 *O*(*n*)，其中 *n* 是列表中的结点数。
* 空间复杂度：*O*(1)，没有使用额外的空间。

**正确性**

我们可以通过定义*循环不变式*来证明此代码的正确性。循环不变式是在循环的每次迭代之前和之后为真的条件。在这种情况下，一个帮助我们证明正确性的循环不变式是这样的：

列表中直到指针 current 的所有结点都不包含重复的元素。

我们可以用归纳法证明这种情况确实是循环不变式。在进入循环之前，current 指向列表的头部。因此，列表中直到 current 的部分只包含头部。因此它不能包含任何重复的元素。现在假设current 现在指向列表中的某个结点（但不是最后一个元素），并且列表中直到 current 的部分不包含重复元素。在另一个循环迭代之后，发生两件事之一。

1. current.next 是 current 的副本。在这种情况下，删除 current.next 中的重复结点，并且current保持指向与之前相同的结点。因此，情况仍然成立；一直到 current 仍然没有重复项。
2. current.next 不是 current 的副本（并且，因为列表已经排序，current.next 也不是 current *之前*出现的任何其他元素的副本）。在这种情况下，current 向前移动一步指向 current.next。因此，情况仍然成立；一直到 current 仍然没有重复项。

在循环的最后一次迭代中，current 必定指向最后一个元素，因为再往后，current.next = null。因此，在循环结束后，直到最后一个元素的所有元素都不包含重复项。