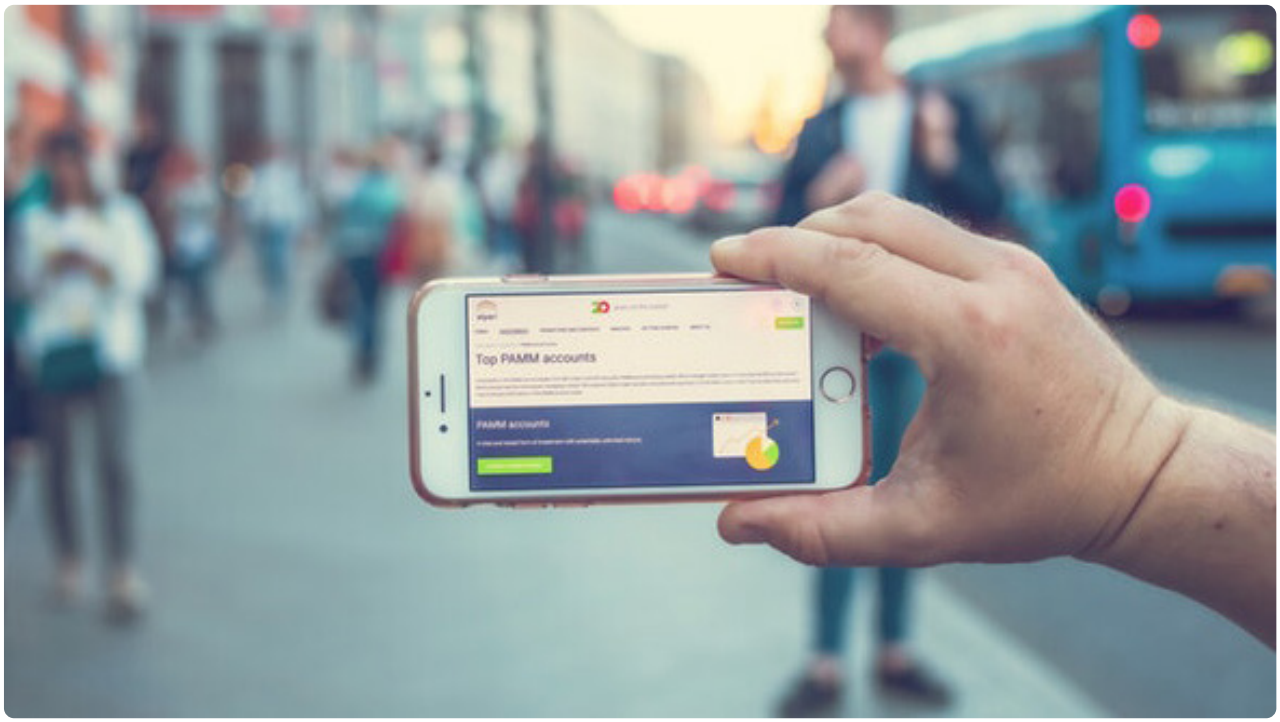
10 合并两个有序链表

更新时间：2019-08-16 10:07:46



> 天才就是百分之二的灵感，百分之九十八的汗水。
>
> ——爱迪生

刷题内容

> 难度: **Easy**
>
> 原题链接：https://leetcode.com/problems/merge-two-sorted-lists/description/。

内容描述

```
将两个有序链表合并为一个新的有序链表并返回。新链表是通过拼接给定的两个链表的所有节点组成的。
示例:
输入：1->2->4, 1->3->4
输出：1->1->2->3->4->4
```

解题方案

思路 1：时间复杂度: O(N) 空间复杂度: O(1)

先学习一下链表：http://data.biancheng.net/view/160.html 。

首先两个链表都是有序的。这里说的有序一般都指的是升序，我们最后也是想返回一个升序的链表。

首先我们要充分利用两个链表 l1, l2 都是有序的这个条件。我们知道，返回结果 res 中的第一个节点的值，一定是从两个输入的链表 l1,l2 当中的某一个的第一个节点中取得。因此我们每次取出比较小的那一个赋值给结果链表中的当前节点 cur，然后将结果链表当前节点 cur 往后挪动一位，链表中节点值较小的那一个也向后移动，直到某一条输入链表值被取完了，我们就停止这个操作。如果此时还有一条链表没有取完，我们需要将结果链表 res 的当前节点 cur 的 next 直接指向那一个链表就可以了，因为剩下的那个链表当前节点的值肯定比我们结果链表当前节点 cur 的值要更大。

下面我们来看具体代码：

Python beats 93.49%

```
class Solution(object):
    def mergeTwoLists(self, l1, l2):
        """
        :type l1: ListNode
        :type l2: ListNode
        :rtype: ListNode
        """
        if not l1:
            return l2
        if not l2:
            return l1

        dummy = cur = ListNode(-1)
        # 遍历两个链表，每次比较链表头的大小，每次让较小值添加到 dummy 的后面，并且让较小值所在的链表后移一位
        while l1 and l2:
            if l1.val < l2.val:
                cur.next = l1
                l1 = l1.next
            else:
                cur.next = l2
                l2 = l2.next
            cur = cur.next

        # 会出现一条链表遍历完，另外一条链表没遍历完的情况，需要将没遍历的链表添加到结果链表中
        cur.next = l1 if l1 else l2
        return dummy.next
```


Java beats 100%

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        ListNode result = new ListNode(0);
        ListNode prev = result;
        // 遍历两个链表，每次比较链表头的大小，每次让较小值添加到 dummy 的后面，并且让较小值所在的链表后移一位
        while (l1!= null && l2!= null) {
            if (l1.val >= l2.val) {
                prev.next = l2;
                l2 = l2.next;
            } else {
                prev.next = l1;
                l1 = l1.next;
            }
            prev = prev.next;
        }
        // 会出现一条链表遍历完，另外一条链表没遍历完的情况，需要将没遍历的链表添加到结果链表中
        if (l1 != null) {
            prev.next = l1;
        }
        if (l2 != null) {
            prev.next = l2;
        }
        return result.next;
    }
}
```


go beats 100%

```go
func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
 dummy := new(ListNode)
 cur := dummy
    // 遍历两个链表，每次比较链表头的大小，每次让较小值添加到 dummy 的后面，并且让较小值所在的链表后移一位
 for {
  if (l1 == nil && l2 == nil) {
   break
  }
  if l1 == nil {
   cur.Next = l2
   break
  }
  if l2 == nil {
   cur.Next = l1
   break
  }
        // 会出现一条链表遍历完，另外一条链表没遍历完的情况，需要将没遍历的链表添加到结果链表中
  if l1.Val < l2.Val {
   cur.Next = l1
   l1 = l1.Next
   cur = cur.Next
  } else {
   cur.Next = l2
   l2 = l2.Next
   cur = cur.Next
  }
 }
 return dummy.Next
}
```

c++ beats 90.58%

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        ListNode* result = new ListNode(0);
        ListNode* prev = result;
        // 遍历两个链表，每次比较链表头的大小，每次让较小值添加到 dummy 的后面，并且让较小值所在的链表后移一位
        while (l1!= NULL && l2!= NULL) {
            if (l1->val >= l2->val) {
                prev->next = l2;
                l2 = l2->next;
            } else {
                prev->next = l1;
                l1 = l1->next;
            }
            prev = prev->next;
        }
        // 会出现一条链表遍历完，另外一条链表没遍历完的情况，需要将没遍历的链表添加到结果链表中
        if (l1 != NULL) {
            prev->next = l1;
        }
        if (l2 != NULL) {
            prev->next = l2;
        }
        return result->next;
    }
};
```


小结

相信大家也看到了，这里我们巧妙运用了一个 dummy 节点来做我们的操作，这个方法可以很有效地避免一些链表的边界问题。因此希望大家能够养成一个习惯，只要做到链表类的问题，就可以无脑使用一个 dummy 节点。

}

09 有效的括号

11 删除排序数组中的重复项