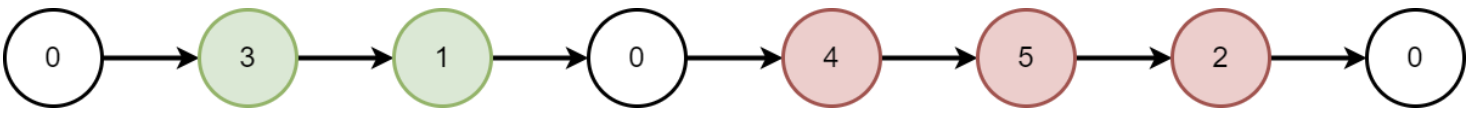
# 题目

给你一个链表的头节点 head ，该链表包含由 0 分隔开的一连串整数。链表的 开端 和 末尾 的节点都满足 Node.val == 0 。

对于每两个相邻的 0 ，请你将它们之间的所有节点合并成一个节点，其值是所有已合并节点的值之和。然后将所有 0 移除，修改后的链表不应该含有任何 0 。

返回修改后链表的头节点 head 。

示例 1：



输入：head = [0,3,1,0,4,5,2,0]

输出：[4,11]

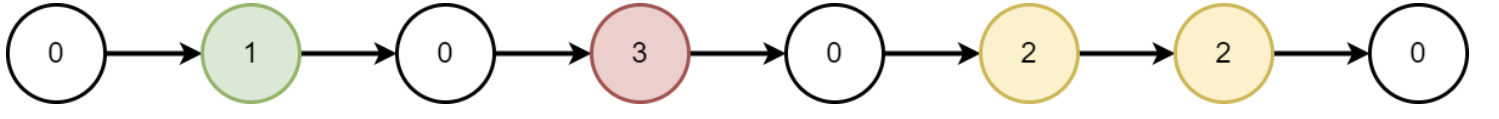
解释：

上图表示输入的链表。修改后的链表包含：

- 标记为绿色的节点之和：3 + 1 = 4

- 标记为红色的节点之和：4 + 5 + 2 = 11

示例 2：



输入：head = [0,1,0,3,0,2,2,0]

输出：[1,3,4]

解释：

上图表示输入的链表。修改后的链表包含：

- 标记为绿色的节点之和：1 = 1

- 标记为红色的节点之和：3 = 3

- 标记为黄色的节点之和：2 + 2 = 4

提示：

列表中的节点数目在范围 [3, 2 \* 105] 内

0 <= Node.val <= 1000

不存在连续两个 Node.val == 0 的节点

链表的开端和末尾节点都满足 Node.val == 0

# 分析

## 方法一：迭代

分析：

我们从链表头节点head的下一个节点开始遍历，并使用一个变量total维护当前遍历到的节点的元素之和。

如果当前节点的值为0，那么我们就新建一个值为total的节点，放在答案链表的尾部，并将total置零，否则我们将值累加进total中。

细节：

为了方便维护答案，我们可以在遍历前新建一个伪头节点dummy，并在遍历完成之后返回dummy的下一个节点作为答案。

代码：

class Solution {

public:

ListNode\* mergeNodes(ListNode\* head) {

ListNode\* dummy = new ListNode();

ListNode\* tail = dummy;

int total = 0;

// 当前节点从head->next开始遍历,因为开端和结尾都满足val=0

// 因此这里遍历的时候直接判断第1个节点的数值即可

for (ListNode\* cur = head->next; cur; cur = cur->next) {

// 如果当前的节点为0,则需要更新tail节点

// 不为0则继续累加,下一次走到val=0的tail节点时更新tail->next为新节点

if (cur->val == 0) {

ListNode\* node = new ListNode(total); //构造新节点

tail->next = node;

tail = tail->next;

total = 0;

}

else {

total += cur->val;

}

}

return dummy->next;

}

};