# 题目

给你一个链表的头节点 head，请你编写代码，反复删去链表中由 总和 值为 0 的连续节点组成的序列，直到不存在这样的序列为止。

删除完毕后，请你返回最终结果链表的头节点。

你可以返回任何满足题目要求的答案。

（注意，下面示例中的所有序列，都是对 ListNode 对象序列化的表示。）

示例 1：

输入：head = [1,2,-3,3,1]

输出：[3,1]

提示：答案 [1,2,1] 也是正确的。

示例 2：

输入：head = [1,2,3,-3,4]

输出：[1,2,4]

示例 3：

输入：head = [1,2,3,-3,-2]

输出：[1]

提示：

给你的链表中可能有 1 到 1000 个节点。

对于链表中的每个节点，节点的值：-1000 <= node.val <= 1000.

# 分析

## 方法一：哈希表

分析：

建立一个dummy节点，指向head，节点值为0。遍历一遍链表，同时记录前缀和，以当前前缀和为key，当前节点为value，存入哈希表中。如果相同前缀和已经存在，就可以直接覆盖掉原有节点。

说明：这里不要陷入，求前后节点的和为零这个问题上，主要关注的其实是和相同。

第二遍重新遍历链表，同时记录前缀和prefix，哈希表中对应prefix的节点是最后一次出现相同前缀和的节点，我们将这个节点的下一个节点，赋值给当前节点的下一个节点，中间跳过的部分总和即为0。

最后我们返回dummy节点的下一节点，作为新的头节点。注意满足题目要求的答案不唯一，可以返回任何满足题目要求的答案。

代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode() : val(0), next(nullptr) {}

\* ListNode(int x) : val(x), next(nullptr) {}

\* ListNode(int x, ListNode \*next) : val(x), next(next) {}

\* };

\*/

class Solution {

public:

ListNode\* removeZeroSumSublists(ListNode\* head) {

ListNode\* dummy = new ListNode(0);

dummy->next = head;

int prefix = 0;

unordered\_map<int, ListNode\*> seen;

for (ListNode\* node = dummy; node; node = node->next) {

prefix += node->val;

seen[prefix] = node;

}

prefix = 0;

for (ListNode\* node = dummy; node; node = node->next) {

prefix += node->val;

node->next = seen[prefix]->next;

}

return dummy->next;

}

};

复杂度分析：

时间复杂度：O(n)，其中n是链表的长度。

空间复杂度：O(n)，其中n是链表的长度。