# 题目

给定一个链表的第一个节点 head ，找到链表中所有出现多于一次的元素，并删除这些元素所在的节点。

返回删除后的链表。

示例 1:

图示

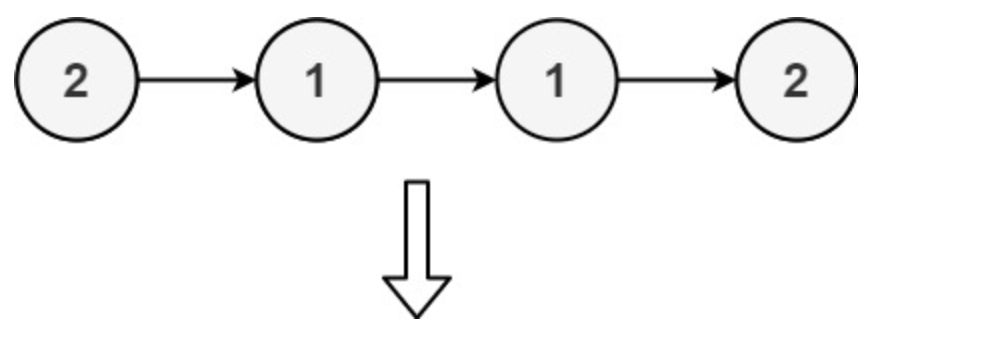
描述已自动生成

输入: head = [1,2,3,2]

输出: [1,3]

解释: 2 在链表中出现了两次，所以所有的 2 都需要被删除。删除了所有的 2 之后，我们还剩下 [1,3] 。

示例 2:



输入: head = [2,1,1,2]

输出: []

解释: 2 和 1 都出现了两次。所有元素都需要被删除。

示例 3:

图示

描述已自动生成

输入: head = [3,2,2,1,3,2,4]

输出: [1,4]

解释: 3 出现了两次，且 2 出现了三次。移除了所有的 3 和 2 后，我们还剩下 [1,4] 。

提示：

链表中节点个数的范围是 [1, 105]

1 <= Node.val <= 105

# 分析

要解决这个问题，我们需要删除链表中所有出现次数多于一次的元素。核心思路是先统计每个元素的出现次数，再根据统计结果筛选出只出现一次的元素，构建新的链表。

思路分析

1、统计元素出现次数：遍历链表，使用哈希表记录每个元素的值及其出现次数。

2、筛选并构建新链表：再次遍历链表，对于每个节点，若其值在哈希表中仅出现一次，则保留该节点；否则，跳过该节点。

3、处理边界情况：为了简化头节点的处理，可使用虚拟头节点（dummy node）作为新链表的起始，最后返回虚拟头节点的下一个节点。

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) {}

\* };

\*/

#include <unordered\_map>

using namespace std;

class Solution {

public:

ListNode\* deleteDuplicatesUnsorted(ListNode\* head) {

if (head == nullptr) return nullptr;

// 步骤1：统计每个元素的出现次数

unordered\_map<int, int> count;

ListNode\* curr = head;

while (curr != nullptr) {

count[curr->val]++;

curr = curr->next;

}

// 步骤2：构建新链表，只保留出现次数为1的元素

ListNode\* dummy = new ListNode(0); // 虚拟头节点，简化头节点处理

ListNode\* prev = dummy; // 新链表的当前尾节点

curr = head;

while (curr != nullptr) {

if (count[curr->val] == 1) {

// 元素只出现一次，加入新链表

prev->next = curr;

prev = prev->next;

}

curr = curr->next;

}

// 截断新链表的尾部（避免残留原链表的节点）

prev->next = nullptr;

return dummy->next;

}

};

代码解释

1、统计出现次数：

- 使用unordered\_map<int, int>类型的count记录每个元素的值及其出现次数。

- 遍历原链表，对每个节点的val进行计数，时间复杂度为O(n)（n为链表长度）。

2、构建新链表：

- 创建虚拟头节点dummy，避免处理头节点为空的特殊情况。

- 用prev指针跟踪新链表的当前尾节点，初始指向dummy。

- 再次遍历原链表，对于每个节点，若其val的出现次数为1，则将其连接到prev的后面，并更新prev；否则，直接跳过该节点。

3、截断尾部：

- 遍历结束后，prev指向新链表的最后一个节点，需将其next设为nullptr，避免残留原链表中未被选中的节点。

复杂度分析

时间复杂度：O(n)，其中n是链表的长度。两次遍历链表（统计次数和构建新链表）均为O(n)，哈希表的插入和查询操作平均为O(1)。

空间复杂度：O(k)，其中k是链表中不同元素的个数。哈希表需存储每个不同元素的出现次数，最坏情况下k = n（所有元素均不重复）。

该方法通过两次遍历和一次哈希表统计，高效地筛选出只出现一次的元素，实现了删除所有重复元素的需求，且逻辑清晰、易于理解。