# 题目

编写代码，移除未排序链表中的重复节点。保留最开始出现的节点。

**示例1:**

输入：[1, 2, 3, 3, 2, 1]

输出：[1, 2, 3]

**示例2:**

输入：[1, 1, 1, 1, 2]

输出：[1, 2]

**提示：**

链表长度在[0, 20000]范围内。

链表元素在[0, 20000]范围内。

**进阶：**

如果不得使用临时缓冲区，该怎么解决？

# 分析

## 方法一：哈希表

**思路：**

我们对给定的链表进行一次遍历，并用一个哈希集合（HashSet）来存储所有出现过的节点。由于在大部分语言中，对给定的链表元素直接进行「相等」比较，实际上是对两个链表元素的地址（而不是值）进行比较。因此，我们在哈希集合中存储链表元素的值，方便直接使用等号进行比较。

具体地，我们从链表的头节点head 开始进行遍历，遍历的指针记为 pos。由于头节点一定不会被删除，因此我们可以枚举待移除节点的前驱节点，减少编写代码的复杂度。

这样枚举有什么好处？试想一下，如果我们直接枚举待移除节点，那么在将它进行移除时，我们本质上是将它的前驱节点连向后继节点。而由于题目给定的链表结构中，我们无法直接访问一个节点的前驱节点。因此，我们不如直接枚举前驱节点 u，那么节点本身就是 u.next，后继节点就是 u.next.next。

在遍历完成后，我们就得到了最终的答案链表。

**代码：**

class Solution {

public:

ListNode\* removeDuplicateNodes(ListNode\* head) {

if (head == nullptr) {

return head;

}

unordered\_set<int> occurred = {head->val};

ListNode\* pos = head;

// 枚举前驱节点

while (pos->next != nullptr) {

// 当前待删除节点

ListNode\* cur = pos->next;

if (!occurred.count(cur->val)) {

occurred.insert(cur->val);

pos = pos->next;

} else {

pos->next = pos->next->next;

}

}

pos->next = nullptr;

return head;

}

};

复杂度分析

时间复杂度：O(N)，其中N是给定链表中节点的数目。

空间复杂度：O(N)。在最坏情况下，给定链表中每个节点都不相同，哈希表中需要存储所有的N个值。

## 方法二：两重循环

考虑题目描述中的「进阶」部分，是否存在一种不使用临时缓冲区（也就是方法一中的哈希表）的方法呢？

不幸的是，在保证方法一时间复杂度O(N) 的前提下，是不存在这样的方法的。因此我们需要增加时间复杂度，使得我们可以仅使用常数的空间来完成本题。一种简单的方法是，我们在给定的链表上使用两重循环，其中第一重循环从链表的头节点开始，枚举一个保留的节点，这是因为我们保留的是「最开始出现的节点」。第二重循环从枚举的保留节点开始，到链表的末尾结束，将所有与保留节点相同的节点全部移除。

方法二的细节部分与方法一类似。第一重循环枚举保留的节点本身，而为了编码方便，第二重循环可以枚举待移除节点的前驱节点，方便我们对节点进行移除。这样一来，我们使用「时间换空间」的方法，就可以不使用临时缓冲区解决本题了。

注意：Python语言会超出时间限制，并不能使用方法二通过本题。

**代码：**

class Solution {

public:

ListNode\* removeDuplicateNodes(ListNode\* head) {

ListNode\* ob = head;

while (ob != nullptr) {

ListNode\* oc = ob;

while (oc->next != nullptr) {

if (oc->next->val == ob->val) {

oc->next = oc->next->next;

} else {

oc = oc->next;

}

}

ob = ob->next;

}

return head;

}

};

复杂度分析：

时间复杂度：O(N^2)，其中N是给定链表中节点的数目。

空间复杂度：O(1)。