删除链表中重复节点

题目描述

在一个排序的链表中,存在重复的结点,请删除该链表中重复的结点,重复的结点不保留,返回链表头指针。 例如,链表 1->2->3->3->4->4->5 处理后为 1->2->5

示例1

```
输入
```

 $\{1, 2, 3, 3, 4, 4, 5\}$

返回值

 $\{1, 2, 5\}$

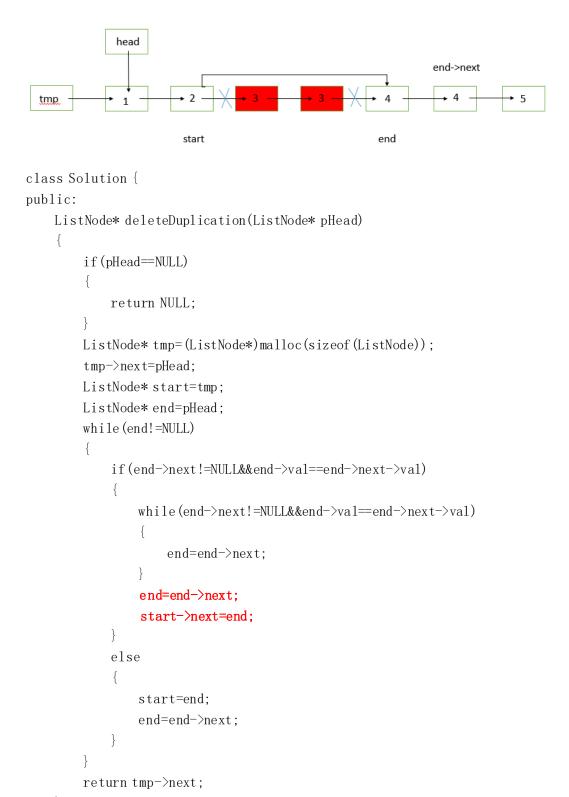
```
head
                      end->next
 tmp
 start
             end
class Solution {
public:
    ListNode* deleteDuplication(ListNode* pHead)
    {
        if (pHead==NULL)
            return NULL;
        ListNode* tmp=(ListNode*)malloc(sizeof(ListNode));
        tmp->next=pHead;
        ListNode* start=tmp;
        ListNode* end=pHead;
        while (end! = NULL)
            if (end->next!=NULL&&end->val==end->next->val)
                while (end->next!=NULL&&end->val==end->next->val)
                    end=end->next;
                end=end->next;
                start->next=end;
```

```
}
             else
                 start=end;
                 end=end->next;
        return tmp->next;
};
              head
                                end->next
                                  → 3 -
  tmp
                        end
              start
class Solution {
public:
    ListNode* deleteDuplication(ListNode* pHead)
         if(pHead == NULL)
             return NULL;
        ListNode* tmp=(ListNode*)malloc(sizeof(ListNode));
         tmp->next=pHead;
        ListNode* start=tmp;
        ListNode* end=pHead;
        while (end!=NULL)
             if (end->next!=NULL&&end->val==end->next->val)
                 \label{lem:while} while (end->next!=NULL\&\&end->val==end->next->val)
                 {
                     end=end->next;
                 end=end->next;
                 start->next=end;
             else
                 start=end;
                 end=end->next;
```

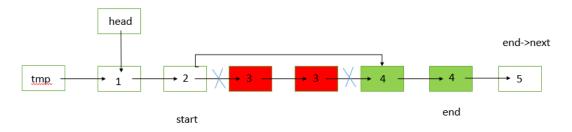
```
}
                                             return tmp->next;
};
                                                                        head
                                                                                                                                                                                                                              end->next
            tmp
                                                                                                                              start
                                                                                                                                                                                 end
 class Solution {
 public:
                       ListNode* deleteDuplication(ListNode* pHead)
                                               if (pHead==NULL)
                                                                   return NULL;
                                             ListNode* tmp=(ListNode*)malloc(sizeof(ListNode));
                                               tmp->next=pHead;
                                             ListNode* start=tmp;
                                             ListNode* end=pHead;
                                             while (end!=NULL)
                                                                   if(end-\next!=NULL\&\&end-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\nex
                                                                                         while(end->next!=NULL&&end->val==end->next->val)
                                                                                                              end=end->next;
                                                                                        end=end->next;
                                                                                        start->next=end;
                                                                   else
                                                                                        start=end;
                                                                                        end=end->next;
                                             return tmp->next;
```

```
};
                                                                                head
                                                                                                                                                                                                                                                                                                  end->next
                                                                                                                                       start
                                                                                                                                                                                                                                                   end
 class Solution {
 public:
                        ListNode* deleteDuplication(ListNode* pHead)
                                                if (pHead==NULL)
                                                                     return NULL;
                                               ListNode* tmp=(ListNode*)malloc(sizeof(ListNode));
                                                tmp->next=pHead;
                                               ListNode* start=tmp;
                                               ListNode* end=pHead;
                                               while (end!=NULL)
                                                                     if (end->next!=NULL&&end->val==end->next->val)
                                                                                             \label{lem:while} while (end-\next!=NULL\&\&end-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-\next-
                                                                                             {
                                                                                                                   end=end->next;
                                                                                           end=end->next;
                                                                                           start->next=end;
                                                                     else
                                                                                           start=end;
                                                                                            end=end->next;
                                               return tmp->next;
```

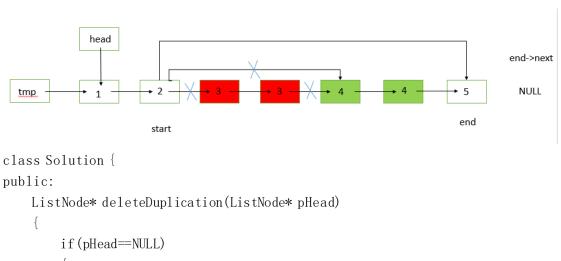
};



};



```
class Solution {
public:
    ListNode* deleteDuplication(ListNode* pHead)
    {
        if(pHead==NULL)
            return NULL;
        ListNode* tmp=(ListNode*)malloc(sizeof(ListNode));
        tmp->next=pHead;
        ListNode* start=tmp;
        ListNode* end=pHead;
        while (end! =NULL)
            if (end->next!=NULL&&end->val==end->next->val)
                while (end->next!=NULL&&end->val=end->next->val)
                     end=end->next;
                end=end->next;
                start->next=end;
            else
                start=end;
                end=end->next;
        return tmp->next;
};
```



```
public:
            return NULL;
        ListNode* tmp=(ListNode*)malloc(sizeof(ListNode));
        tmp->next=pHead;
        ListNode* start=tmp;
        ListNode* end=pHead;
        while (end!=NULL)
            if (end->next!=NULL&&end->val==end->next->val)
                while (end->next!=NULL&&end->val==end->next->val)
                {
                    end=end->next;
                end=end->next;
                start->next=end;
            else
                start=end;
                end=end->next;
        return tmp->next;
};
```