

082. Remove Duplicates from Sorted List II

082 Remove Duplicates from Sorted List II

- Linked List

Description

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only *distinct* numbers from the original list.

For example,

Given `1->2->3->3->4->4->5`, return `1->2->5`.

Given `1->1->1->2->3`, return `2->3`.

1. Thought line

To each "distinct element checking loop":

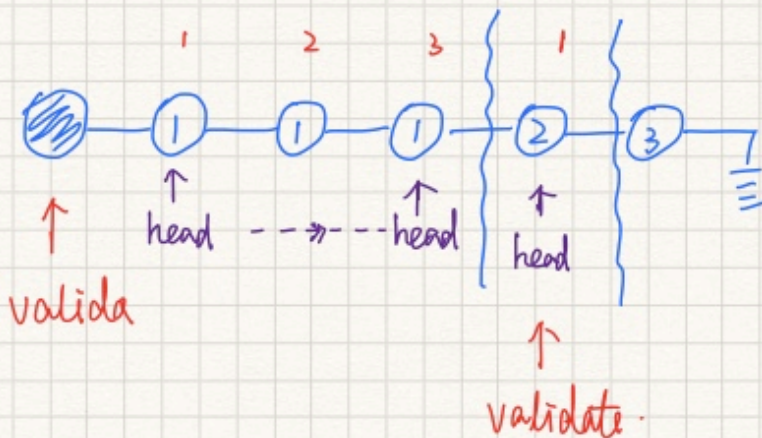
- Count times : countTimes

- If countTimes == 1;

- validatePointer points to this element
- Next "distinct element checking loop":

- If countTimes > 1;

- validatePointer stay
- Next "distinct element checking loop":



NewHead = Head -> next

2. Linked List

```
1 /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode(int x) : val(x), next(NULL) {}
7  * };
8  */
9 class Solution {
10 public:
11     ListNode* deleteDuplicates(ListNode* head) {
12         ListNode* dummyHead = new ListNode(0);
13         dummyHead->next = head;
14         ListNode* validatedPtr = dummyHead;
15         int actElement = 0;
16         while(head!=nullptr){
```

```

17     ++actElement;
18     while(head->next!=nullptr && head->next->val == head->val){
19         ++actElement;
20         head = head->next;
21     }
22     ListNode* newHead = head->next;
23     if (actElement==1){
24         validatedPtr->next = head;
25         validatedPtr = validatedPtr->next;
26     }
27     else{
28         validatedPtr->next = nullptr;
29         head->next = nullptr;
30     }
31     head = newHead;
32     actElement = 0;
33 }
34 return dummyHead->next;
35 }
36 };
37
38
39
40
41 /**
42  * Definition for singly-linked list.
43  * struct ListNode {
44  *     int val;
45  *     ListNode *next;
46  *     ListNode(int x) : val(x), next(NULL) {}
47  * };
48  */
49 class Solution {
50 public:
51     ListNode* deleteDuplicates(ListNode* head) {
52         ListNode* dummyHead = new ListNode(0);
53         dummyHead->next = head;
54         ListNode* validatedPtr = dummyHead;
55         int actElement = 0;
56         while(head!=nullptr){
57             ++actElement;
58             while(head->next!=nullptr && head->next->val == head->val){
59                 ++actElement;
60                 head = head->next;
61             }
62             if (actElement==1){
63                 validatedPtr->next = head;
64                 validatedPtr = validatedPtr->next;
65             }
66             /*
67              @_: if actElement >1, validatedPtr doesn't move
68              @_: dummyHead list is dominated by dummyHead and validatedPtr ONLY.
69              */
70             /*
71             head = head->next;
72             actElement = 0;
73             */
74             validatedPtr->next = nullptr;
75             return dummyHead->next;
76         }
77     };

```