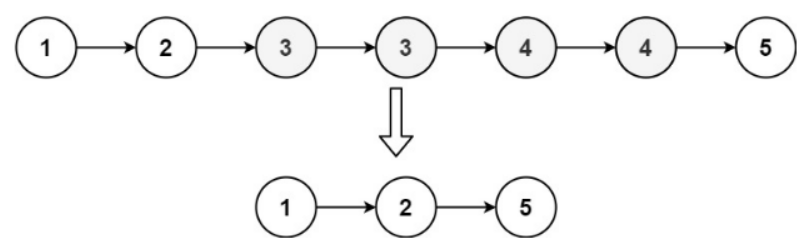
<https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/>

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

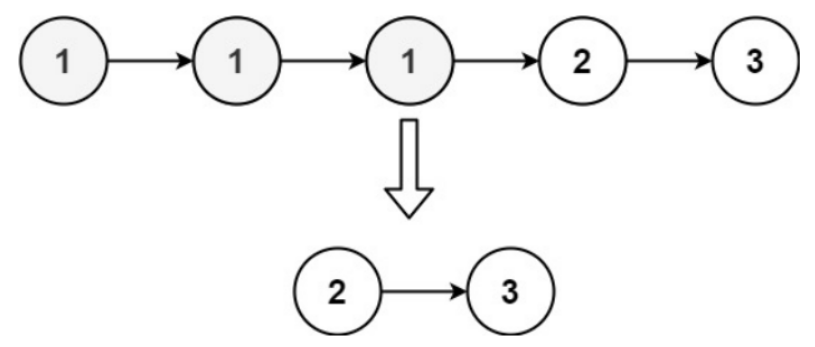
**Example 1:**



Input: head = [1,2,3,3,4,4,5]

Output: [1,2,5]

**Example 2:**



Input: head = [1,1,1,2,3]

Output: [2,3]

**Constraints:**

* The number of nodes in the list is in the range [0, 300].
* -100 <= Node.val <= 100
* The list is guaranteed to be **sorted** in ascending order.

**Attempt 1: 2023-02-11**

**Wrong Solution**

**If just copy the same way from L83.Remove Duplicates from Sorted List, it won't work**

**Test case:**

**Input: 1 -> 1 -> 1 -> 2 -> 3**

**Output: 1->2->3**

**Expected: 2->3**

**Not able to remove the first node '1'**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode deleteDuplicates(ListNode head) {

if(head == null || head.next == null) {

return head;

}

ListNode dummy = new ListNode();

dummy.next = head;

ListNode iter = dummy;

while(iter.next != null && iter.next.next != null) {

if(iter.next.val == iter.next.next.val) {

// Here is the copy from L83.Remove Duplicates from Sorted List, but wrong

iter.next = iter.next.next.next;

} else {

iter = iter.next;

}

}

return dummy.next;

}

}

**Solution 1:  Iterative Solution (60 min)**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode deleteDuplicates(ListNode head) {

if(head == null || head.next == null) {

return head;

}

ListNode dummy = new ListNode();

dummy.next = head;

ListNode iter = dummy;

while(iter.next != null && iter.next.next != null) {

// Detect duplicates exist in list

if(iter.next.val == iter.next.next.val) {

// Declear duplciate value for current section

int dup\_val = iter.next.val;

// Recursively skip duplicate value node one by one,

// and different from L83 is in L82 here we will also

// skip the initial node contains duplicate value

// e.g 1->1->1->2->3

// In L83 we only skip 2nd and 3rd '1' => 1->2->3

// In L82 we skip all three '1' => 2->3

while(iter.next != null && iter.next.val == dup\_val) {

iter.next = iter.next.next;

}

} else {

iter = iter.next;

}

}

return dummy.next;

}

}

Time Complexity: O(n)

Space Complexity: O(1)

**Refer to**

<https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/solutions/2419088/very-easy-100-fully-explained-java-c-python-js-c-python3/>

class Solution {

public ListNode deleteDuplicates(ListNode head) {

// Special case...

if (head == null || head.next == null)

return head;

// create a fake node that acts like a fake head of list pointing to the original head and it points to the original head......

ListNode fake = new ListNode(0);

fake.next = head;

ListNode curr = fake;

// Loop till curr.next and curr.next.next not null

while(curr.next != null && curr.next.next != null){ // curr.next means the next node of curr pointer and curr.next.next means the next of next of curr pointer...

// if the value of curr.next and curr.next.next is same...

// There is a duplicate value present in the list...

if(curr.next.val == curr.next.next.val) {

int duplicate = curr.next.val;

// If the next node of curr is not null and its value is eual to the duplicate value...

while(curr.next !=null && curr.next.val == duplicate) {

// Skip those element and keep updating curr...

curr.next = curr.next.next;

}

}

// Otherwise, move curr forward...

else{

curr = curr.next;

}

}

return fake.next; // Return the linked list...

}

}

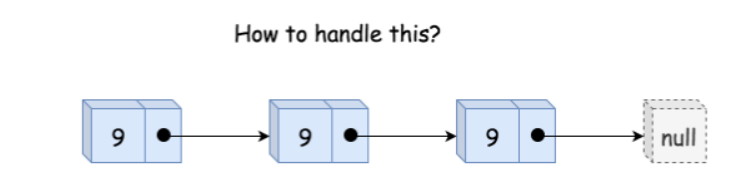
**Refer to**

<https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/solutions/952302/remove-duplicates-from-sorted-list-ii/>

#### **Approach 1: Sentinel Head + Predecessor**

**Sentinel Head**

Let's start from the most challenging situation: the list head is to be removed.



*Figure 1. The list head is to be removed.*

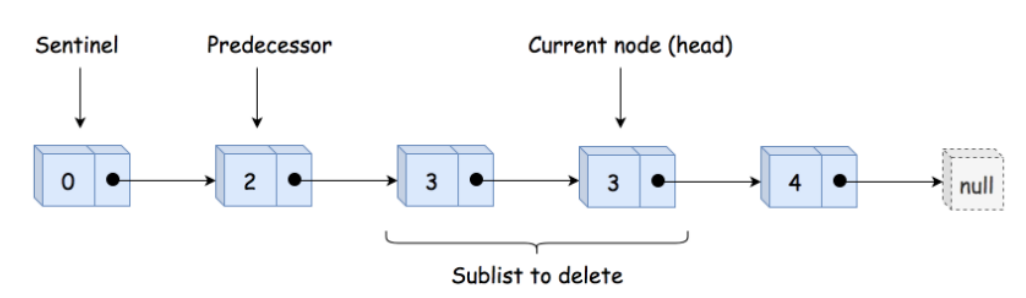
The standard way to handle this use case is to use the so-called [Sentinel Node](https://en.wikipedia.org/wiki/Sentinel_node). Sentinel nodes are widely used for trees and linked lists as pseudo-heads, pseudo-tails, etc. They are purely functional and usually don't hold any data. Their primary purpose is to standardize the situation to avoid edge case handling.

For example, let's use here pseudo-head with zero value to ensure that the situation "delete the list head" could never happen, and all nodes to delete are "inside" the list.

**Delete Internal Nodes**

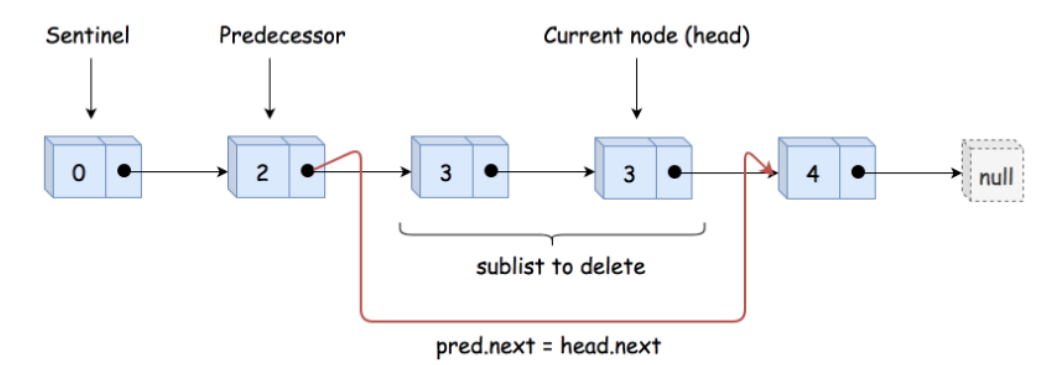
The input list is sorted, and we can determine if a node is a duplicate by comparing its value to the node *after* it in the list. Step by step, we could identify the current sublist of duplicates.

Now it's time to delete it using pointer manipulations. Note that the first node in the duplicates sublist should be removed as well. That means that we have to track the predecessor of duplicates sublist, *i.e.*, the last node *before* the sublist of duplicates.



*Figure 2. The sentinel head, the predecessor, and the sublist of duplicates to delete.*

Having predecessor, we skip the entire duplicate sublist and make predecessor to point to the node *after* the sublist.



*Figure 2. Delete the sublist of duplicates.*

**Implementation**

class Solution {

public ListNode deleteDuplicates(ListNode head) {

// sentinel

ListNode sentinel = new ListNode(0, head);

// predecessor = the last node

// before the sublist of duplicates

ListNode pred = sentinel;

while (head != null) {

// if it's a beginning of duplicates sublist

// skip all duplicates

if (head.next != null && head.val == head.next.val) {

// move till the end of duplicates sublist

while (head.next != null && head.val == head.next.val) {

head = head.next;

}

// skip all duplicates

pred.next = head.next;

// otherwise, move predecessor

} else {

pred = pred.next;

}

// move forward

head = head.next;

}

return sentinel.next;

}

}

**Complexity Analysis**

* Time complexity: O(N) since it's one pass along the input list.
* Space complexity: O(1) because we don't allocate any additional data structure.

**Solution 2:  Recursive Solution (30 min)**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode deleteDuplicates(ListNode head) {

if(head == null || head.next == null) {

return head;

}

int cur\_val = head.val;

ListNode p = head.next;

if(p.val == cur\_val) {

while(p != null && p.val == cur\_val) {

p = p.next;

}

return deleteDuplicates(p);

} else {

head.next = deleteDuplicates(p);

return head;

}

}

}

=============================================================================

OR remove ListNode p = head.next, and make more similar to L83. Remove Duplicates from Sorted List

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode deleteDuplicates(ListNode head) {

if(head == null || head.next == null) {

return head;

}

// If current node is not unique, return deleteDuplicates with head.next

if(head.next.val == head.val) {

while(head.next != null && head.next.val == head.val) {

head.next = head.next.next;

}

// 'head' also removed, hence no connection between 'head' and 'head.next',

// hence no "head.next = deleteDuplicates(head.next)", and we only want

// start recursion with 'head.next', so directly "deleteDuplicates(head.next)"

return deleteDuplicates(head.next);

// If current node is unique, link it to the result of next list made by recursive call, similar as how L83.Remove Duplicates from Sorted List

} else {

head.next = deleteDuplicates(head.next);

return head;

}

}

}

Time Complexity: O(n)

Space Complexity: O(1)

**Refer to**

<https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/solutions/28355/simple-and-clear-c-recursive-solution/>

class Solution {

public:

ListNode\* deleteDuplicates(ListNode\* head) {

if (!head) return 0;

if (!head->next) return head;

int val = head->val;

ListNode\* p = head->next;

if (p->val != val) {

head->next = deleteDuplicates(p);

return head;

} else {

while (p && p->val == val) p = p->next;

return deleteDuplicates(p);

}

}

};

<https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/solutions/28339/my-recursive-java-solution/>

if current node is not unique, return deleteDuplicates with [head.next](http://head.next).

If current node is unique, link it to the result of next list made by recursive call.

public ListNode deleteDuplicates(ListNode head) {

if (head == null) return null;

if (head.next != null && head.val == head.next.val) {

while (head.next != null && head.val == head.next.val) {

head = head.next;

}

return deleteDuplicates(head.next);

} else {

head.next = deleteDuplicates(head.next);

}

return head;

}