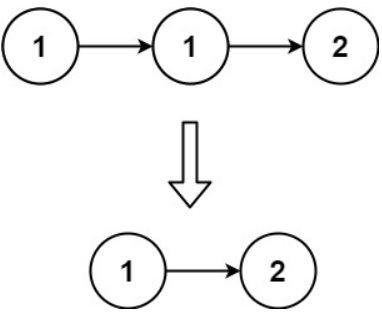
<https://leetcode.com/problems/remove-duplicates-from-sorted-list/>

Given the head of a sorted linked list, *delete all duplicates such that each element appears only once*. Return *the linked list* ***sorted*** *as well*.

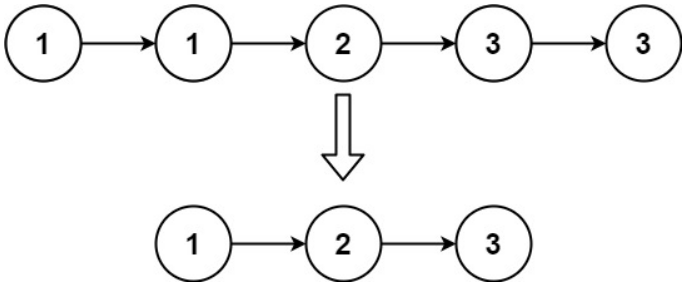
**Example 1:**



Input: head = [1,1,2]

Output: [1,2]

**Example 2:**



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

Output: [1,2,3]

**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**

**Solution 1:  Iterative Solution (10 min)**

**Style 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 iter = head;

        while(iter.next != null) {

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

                iter.next = iter.next.next;

            } else {

                iter = iter.next;

            }

        }

        return head;

    }

}

**Refer to**

<https://leetcode.com/problems/remove-duplicates-from-sorted-list/solutions/28614/my-pretty-solution-java/comments/27524>

public ListNode deleteDuplicates2(ListNode head) {

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

        ListNode list = head;

        while(list.next != null)

        {

            if (list.val == list.next.val)

                list.next = list.next.next;

            else

                list = list.next;

        }

        return head;

    }

**Style 2**

/\*\*

\* 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 iter = head;

        while(iter != null) {

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

                iter.next = iter.next.next;

            }

            iter = iter.next;

        }

        return head;

    }

}

**Refer to**

<https://leetcode.com/problems/remove-duplicates-from-sorted-list/solutions/28614/my-pretty-solution-java/comments/330511>

var deleteDuplicates = function(head) {

  let curr = head;

  while (curr) {

    while (curr.next && curr.next.val === curr.val) {

      curr.next = curr.next.next;

    }

    curr = curr.next;

  }

  return head;

};

**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;

        }

        head.next = deleteDuplicates(head.next);

        return head.val == head.next.val ? head.next : head;

    }

}

**Refer to**

<https://leetcode.com/problems/remove-duplicates-from-sorted-list/solutions/28625/3-line-java-recursive-solution/>

public ListNode deleteDuplicates(ListNode head) {

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

        head.next = deleteDuplicates(head.next);

        return head.val == head.next.val ? head.next : head;

}

**Step by Step explain how recursion works on LinkedList**

**Take an example as 1 -> 1 -> 2:**

**The code first go to the last node as 2 (similar like DFS on tree go to the leaves first), when 2.next == null, it return from level 3 recursion to level 2 recursion,  on level 2 recursion, we compare second node 1 with last node 2,  the value is different, we don't need to skip current node (second node 1) by pointing it to its next node (last node 2), so we just have to return  current node by "return head",  no it return from level 2 recursion to level 1 recursion, on level 1 recursion, we compare first node 1 with second node 1, the value is identical, we need to skip current node (first node 1) by pointing it to its next node (second node 1), so we just have to return current node's next node by "return head.next"**

public class LinkedListSolution {

    private class ListNode {

        int val;

        ListNode next;

        ListNode() {}

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

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

    }

    public ListNode deleteDuplicates(ListNode head) {

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

            return head;

        }

        head.next = deleteDuplicates(head.next);

        return head.val == head.next.val ? head.next.next : head.next;

    }

    public static void main(String[] args) {

        LinkedListSolution l = new LinkedListSolution();

        ListNode node1\_1 = l.new ListNode(1);

        ListNode node1\_2 = l.new ListNode(1);

        ListNode node2 = l.new ListNode(2);

        //ListNode node3\_1 = l.new ListNode(3);

        //ListNode node3\_2 = l.new ListNode(3);

        node1\_1.next = node1\_2;

        node1\_2.next = node2;

        //node2.next = node3\_1;

        //node3\_1.next = node3\_2;

        ListNode result = l.deleteDuplicates(node1\_1);

        System.out.println(result);

    }

}

**Refer to**

[L203.Remove Linked List Elements (Ref.L83)](note://636C112756564A3985BB8B72FF6310AB)