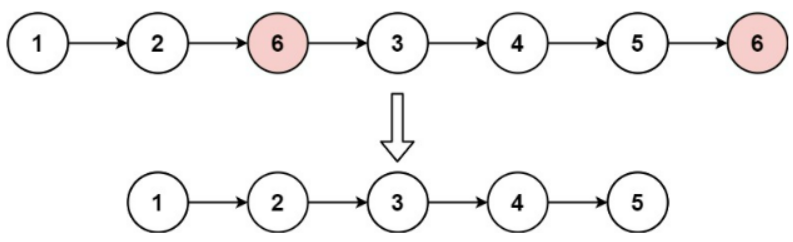
<https://leetcode.com/problems/remove-linked-list-elements/>

Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return *the new head*.

**Example 1:**



Input: head = [1,2,6,3,4,5,6], val = 6

Output: [1,2,3,4,5]

**Example 2:**

Input: head = [], val = 1

Output: []

**Example 3:**

Input: head = [7,7,7,7], val = 7

Output: []

**Constraints:**

* The number of nodes in the list is in the range [0, 104].
* 1 <= Node.val <= 50
* 0 <= val <= 50

**Attempt 1: 2023-02-09**

**Solution 1: Iterative Solution without using a Previous Pointer (10 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 removeElements(ListNode head, int val) {

// Setup dummy header since we also want to scan on 'head'

ListNode dummyHead = new ListNode();

dummyHead.next = head;

ListNode iter = dummyHead;

while(iter.next != null) {

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

iter.next = iter.next.next;

} else {

iter = iter.next;

}

}

// And since we modify the original linkedlist structure,

// we cannot return original 'head', have to return 'dummyHead.next'

// which refer to the modified likedlist

return dummyHead.next;

}

}

**Refer to**

<https://leetcode.com/problems/remove-linked-list-elements/solutions/1572932/java-three-simple-clean-solutions-w-explanation-iterative-recursive-beats-100/>

In this solution, we create a dummy node and set [dummy.next](http://dummy.next) = head. This node helps us keep track of the new head in case the existing head has to be removed.Since we are not using a previous pointer, we will set the current (cur) node to the dummy node.The list on the left side, including the cur node, has been solved. We now need to solve the list on the right side of the cur node.

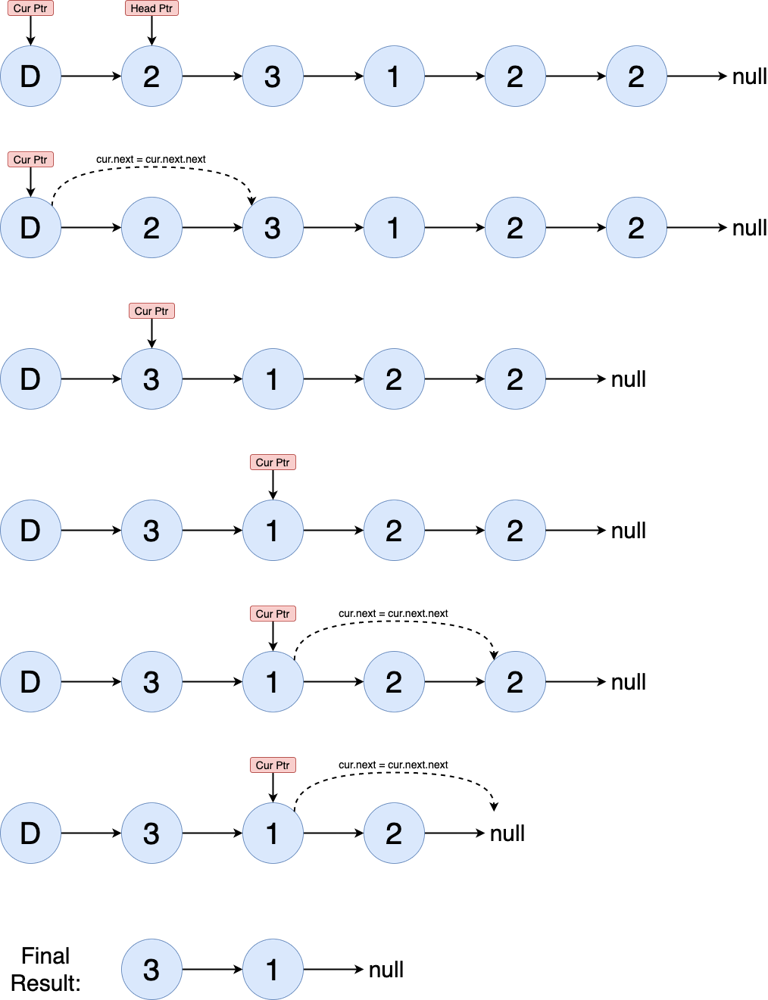
We will check the following two conditions while iterating over the list:

* If [cur.next.val](http://cur.next.val) == val, then we will remove the [cur.next](http://cur.next) node by setting [cur.next](http://cur.next) = [cur.next.next](http://cur.next.next). Please note, we will not move the current pointer in this step as new [cur.next](http://cur.next) has not be validated yet.
* If [cur.next.val](http://cur.next.val) != val, then we can safely move the cur pointer to next node, as next node is a valid node. cur = [cur.next](http://cur.next).

**Time Complexity:** O(N) --> Each Node in the list is visited once

**Space Complexity:** O(1) --> Constant space is used for this solution *Where, N = Length of the input list.*

**Example to understand this solution** Input LinkedList: [2, 3, 1, 2, 2]In this input Linked List we are removing the value 2.



class Solution {

public ListNode removeElements(ListNode head, int val) {

if (head == null) {

return null;

}

ListNode dummy = new ListNode();

dummy.next = head;

ListNode cur = dummy;

while (cur.next != null) {

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

cur.next = cur.next.next;

// Here cannot move cur to cur.next as we need to validate the next node.

} else {

cur = cur.next;

}

}

return dummy.next;

}

}

**Solution 2: Iterative Solution using a Previous Pointer (10 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 removeElements(ListNode head, int val) {

ListNode dummy = new ListNode();

dummy.next = head;

ListNode prev = dummy;

ListNode cur = head;

while(cur != null) {

if(cur.val == val) {

prev.next = cur.next;

} else {

prev = cur;

}

cur = cur.next;

}

return dummy.next;

}

}

**Refer to**

<https://leetcode.com/problems/remove-linked-list-elements/solutions/1572932/java-three-simple-clean-solutions-w-explanation-iterative-recursive-beats-100/>

✔️ **Approach 2: Iterative Solution using a Previous Pointer**

Same as in the previous solution, we create a dummy node and set [dummy.next](http://dummy.next) = head. **This node helps us keep track of the new head in case the existing head has to be removed.** We will initialize, prev node to dummy node and cur node to head node. The list on the left side, including the prev node, has been solved. We now need to solve the list starting from the cur node.

We will check the following two conditions while iterating over the list:

* If cur.val == val, then we will remove the cur node by setting [prev.next](http://prev.next) = [cur.next](http://cur.next).
* If cur.val != val, then we can safely move the prev pointer to cur node, as cur node is a valid node. prev = cur.

**Time Complexity:** O(N) --> Each Node in the list is visited once.

**Space Complexity:** O(1) --> Constant space is used for this solution *Where, N = Length of the input list.*

class Solution {

public ListNode removeElements(ListNode head, int val) {

if (head == null) {

return null;

}

ListNode dummy = new ListNode();

dummy.next = head;

ListNode cur = head;

ListNode prev = dummy;

while (cur != null) {

if (cur.val == val) {

prev.next = cur.next;

} else {

prev = cur;

}

cur = cur.next;

}

return dummy.next;

}

}

**Solution 3: Recursive Solution (30 min, Divide and Conquer: 1.base case -> 2.递归成为更小的问题 -> 3.进行当前层的处理计算)**

/\*\*

\* 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 removeElements(ListNode head, int val) {

// 1.Base case

// When the input node is an empty node, then there is

// nothing to delete, so we just return a null node back

if(head == null) {

return null;

}

// 2.递归成为更小的问题

// We apply the same thing to every other node until it

// reaches null

head.next = removeElements(head.next, val);

// 3.进行当前层的处理计算

// When the head of the input node is the target we want to

// delete, we just return head.next instead of head to skip it,

// else we will return head

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

}

}

**Refer to**

<https://leetcode.com/problems/remove-linked-list-elements/solutions/57306/3-line-recursive-solution/>

public ListNode removeElements(ListNode head, int val) {

if (head == null) return null;

head.next = removeElements(head.next, val);

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

}

**When the input node is an empty node, then there is nothing to delete, so we just return a null node back. (That's the first line)**

**When the head of the input node is the target we want to delete, we just return head.next instead of head to skip it. (That's the third line), else we will return head.**

**We apply the same thing to every other node until it reaches null. (That's the second line).**

<https://leetcode.com/problems/remove-linked-list-elements/solutions/1572932/java-three-simple-clean-solutions-w-explanation-iterative-recursive-beats-100/>

✔️ **Approach 3: Recursive Solution**

In this solution, removeElements function returns the head of the solved list (nodes with val are removed).Once we get the solved rightSideHead from the recursion call, rightSideHead node will point to the solved list. Now, we have two choices:

* If head.val == val --> Current node needs to be removed, so return the rightSideHead as it points to the solved right side list.
* If head.val != val --> Update the next pointer of current list, as the next node might have been removed. And then return the current node.

**Time Complexity:** O(N) --> Each Node in the list is visited once.

**Space Complexity:** O(N) --> Recursion Stack space *Where, N = Length of the input list.*

class Solution {

public ListNode removeElements(ListNode head, int val) {

if (head == null) {

return null;

}

// Once removeElements call is done, right side of the list is solved.

ListNode rightSideHead = removeElements(head.next, val);

if (head.val == val) {

return rightSideHead;

}

head.next = rightSideHead;

return head;

}

}