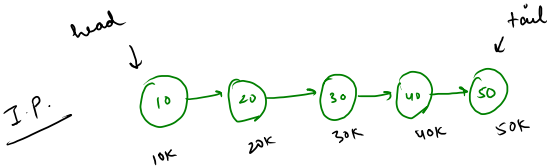
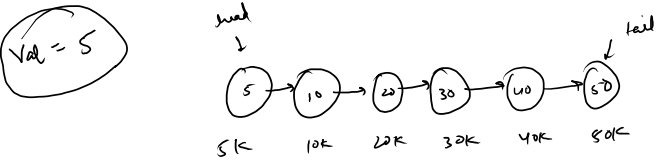
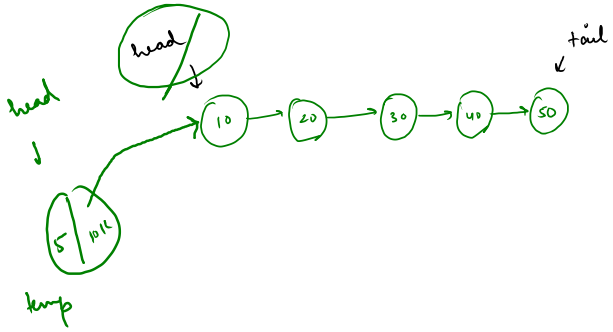
Add First in a LL :-



head = 10K
tail = 50K
size = 5



head = 5K
tail = 50K
size = 6

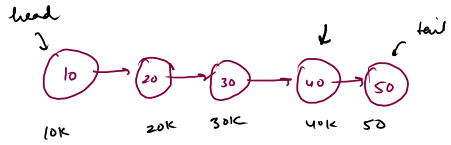



```
Node temp = new Node();
temp.data = val;

if (size == 0) {
    head = temp
    tail = temp
}
else {
    temp.next = head;
    head = temp;
}
size++;
}
```

```
public void addFirst(int val) {
  // write your code here
    Node temp = new Node();
    temp.data = val;

    if (size == 0) {
        head = temp;
        tail = temp;
    } else {
        temp.next = head;
        head = temp;
    }

    size++;
}
```

Remove Last in a LL :-

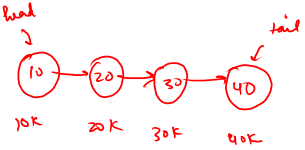
I.P.



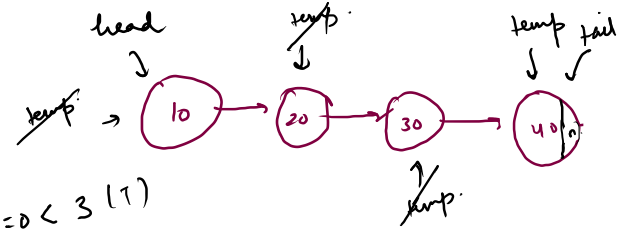
head = 10K
tail = 50K
size = 5

Remove Last

O.P.

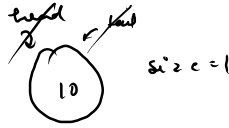


head = 10K
tail = 40K
size = 4



i = 0 < 3 (T)
i = 1 < 3 (T)
i = 2 < 3 (T)
i = 3 < 3 ✗

```
→ (size == 0) {
    ↳ sysout ("List is empty")
  }
→ (size == 1) {
    head = null;
    tail = null;
    size = 0;
  }
  else {
    for (int i = 0; i < size-2; i++)
    {
      temp = temp.next;
    }

    temp.next = null;
    tail = temp
    size--;
  }
```



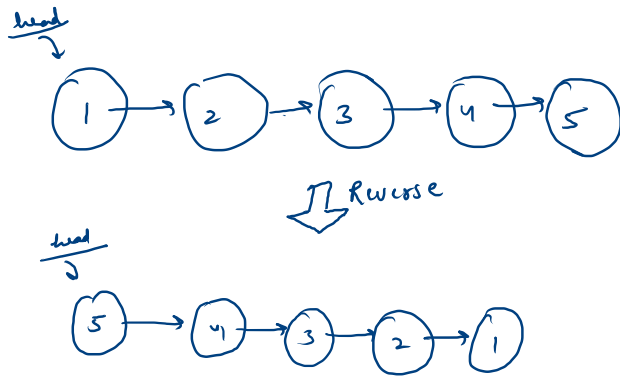
```java
public void removeLast() {
  // write your code here
    if (size == 0) {
        System.out.println("List is empty");
    } else if (size == 1) {
        tail = head = null;
        size--;
    } else {
        Node temp = head;
        for(int i=0;i<size-2;i++) {
            temp = temp.next;
        }

        temp.next = null;
        tail = temp;
        size--;
    }

}
```

206. Reverse Linked List

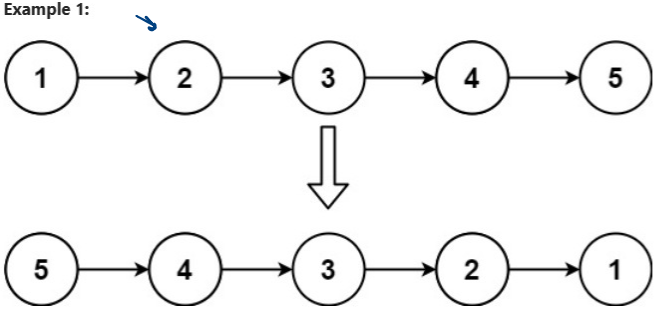
Easy 19.7K 357

Companies

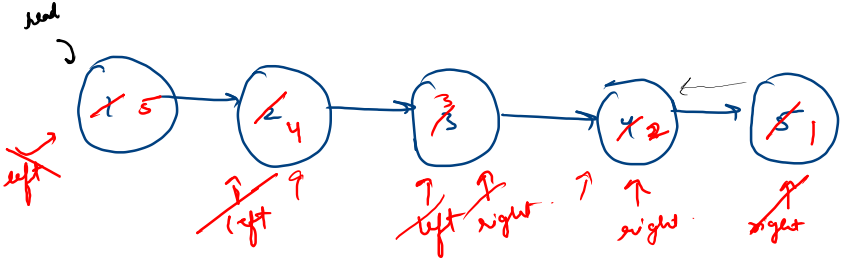
Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

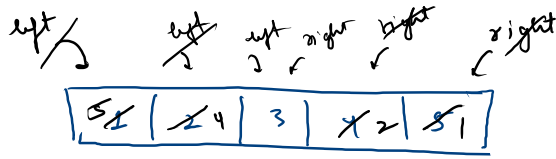


Example 1:




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





T.C.

↳ $O(n^2)$

1st approach (left <= right)

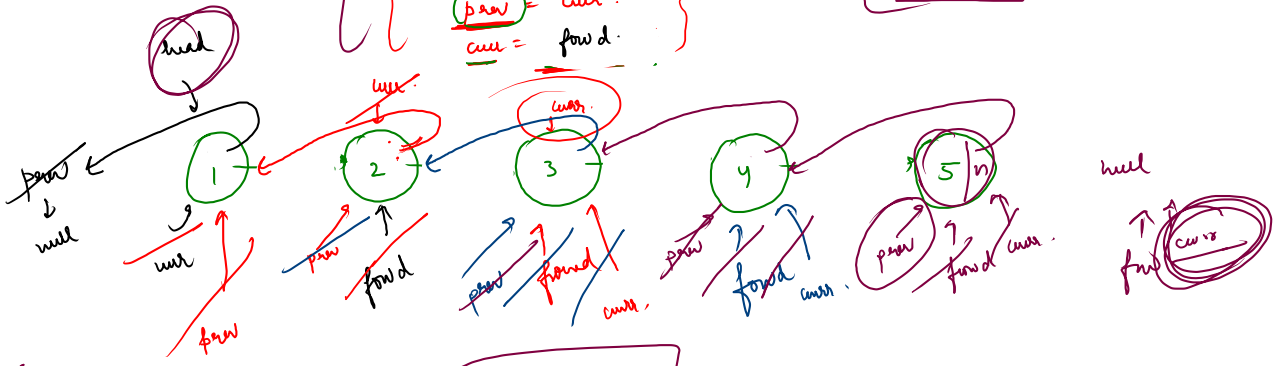
$n, n-1, n-2, n-3, \ldots$

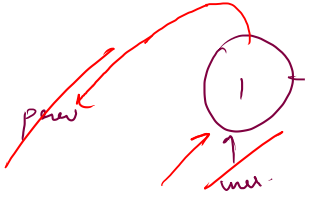
$\frac{n(n+1)}{2} = n^2$

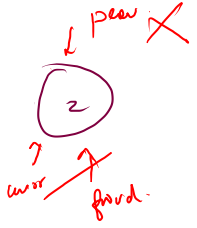
V.V. Imp.

```
while (curr != null)
    forward = curr.next;
    curr.next = prev
    prev = curr
    curr = forward
```

→ O(n)







prev = curr

```java
class Solution {
    public ListNode reverseList(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }

        ListNode prev = null;
        ListNode curr = head;

        while (curr != null) {
            ListNode fowd = curr.next;

            curr.next = prev;

            prev = curr;
            curr = fowd;
        }

        return prev;
    }
}
```

876. Middle of the Linked List

Easy 10.6K 312

Companies

Given the `head` of a singly linked list, return *the middle node of the linked list.*

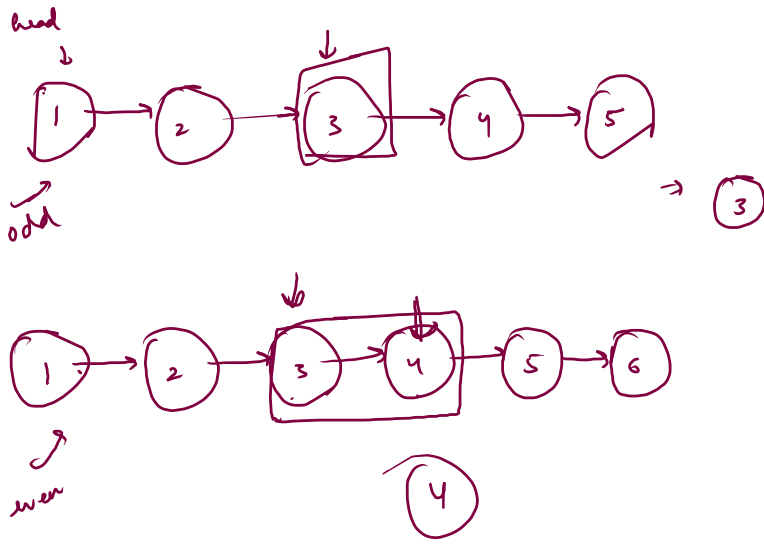
If there are two middle nodes, return **the second middle** node.

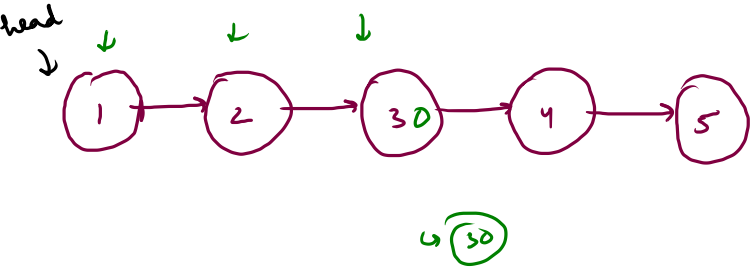
Example 1:

```
Input: head = [1,2,3,4,5]
Output: [3,4,5]
Explanation: The middle node of the list is node 3.
```

Example 2:

```
Input: head = [1,2,3,4,5,6]
Output: [4,5,6]
Explanation: Since the list has two middle nodes with values 3 and 4, we
return the second one.
```

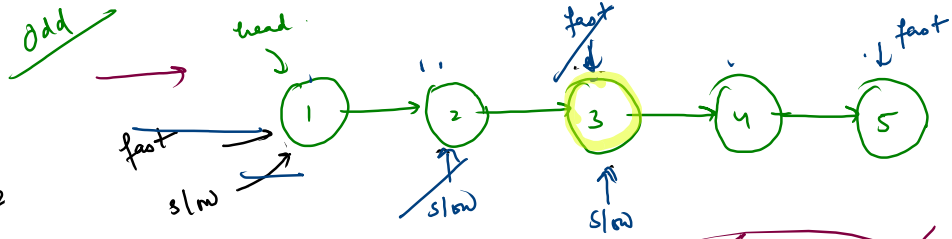




→ Calculate the no. of nodes.

⇒ $\frac{5}{2}$ = 3

↳ Iterate over the LL 2 no. of times



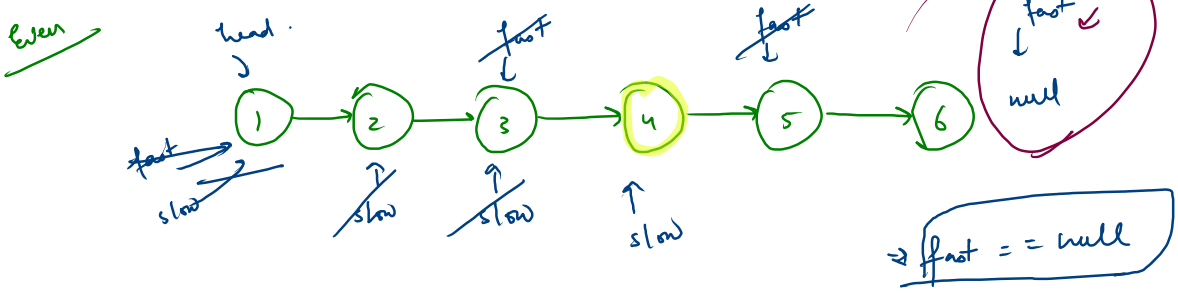
1) fast = head

slow = head

2) slow = slow.next;

fast = fast.next.next;

⇒ In case of odd length ⇒ fast.next == null.



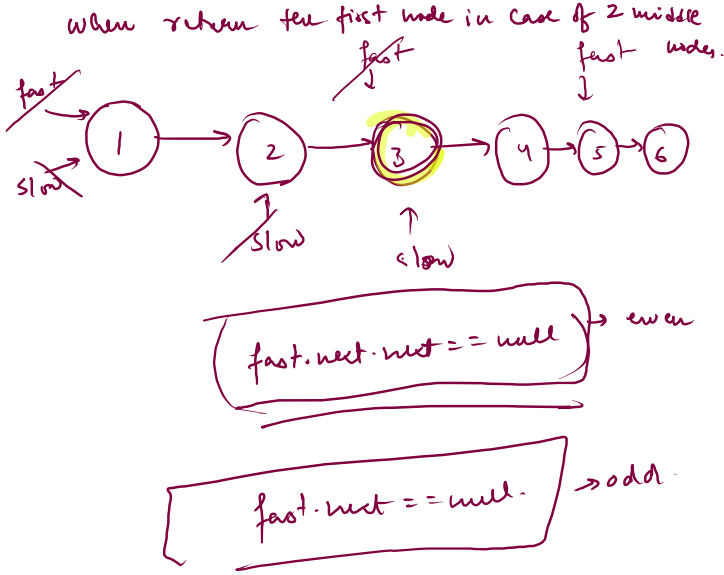
⇒ fast == null

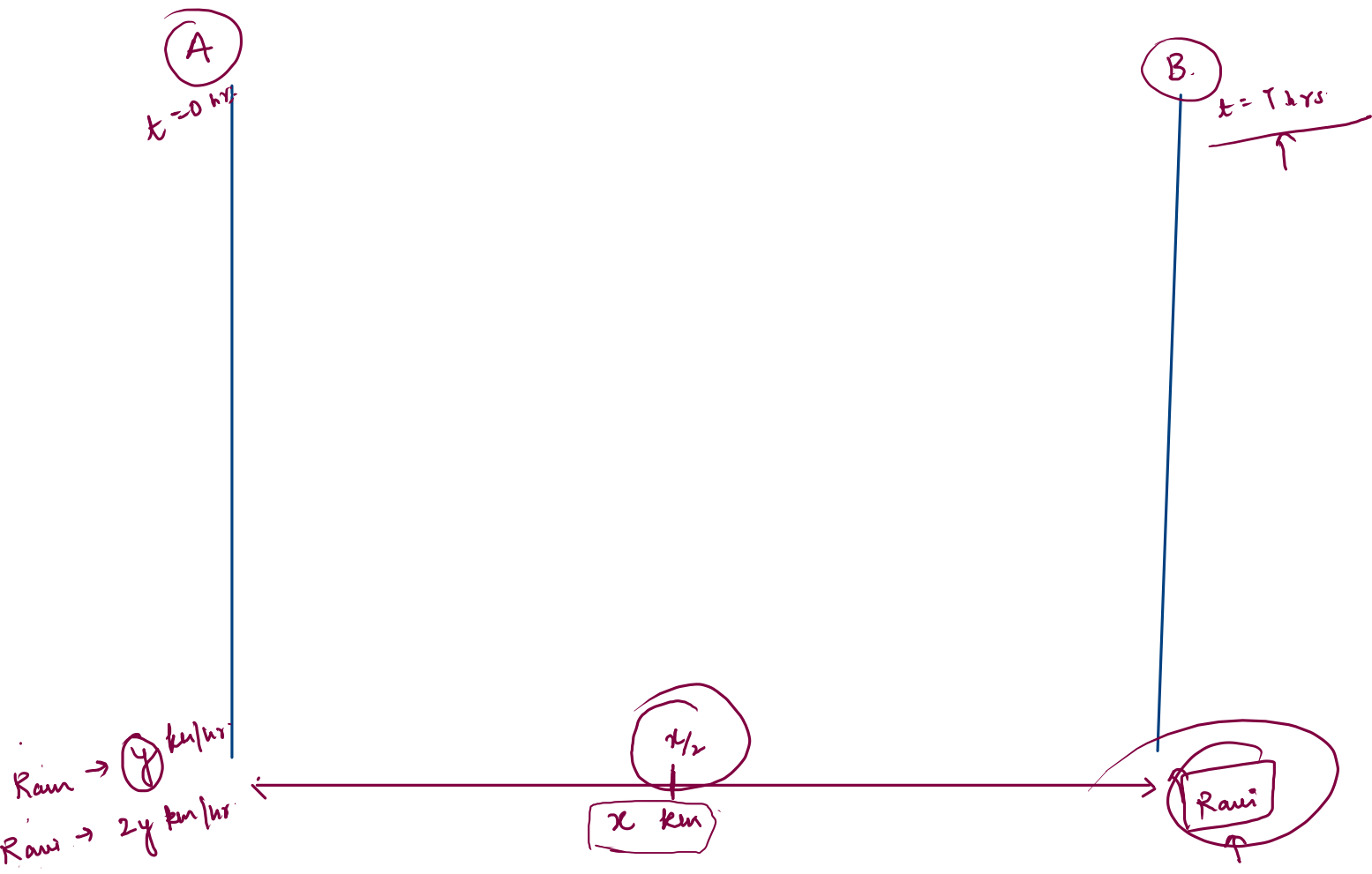
slow = x km/hr.

fast = 2x km/hr.

Reverse Middle

```java
public ListNode middleNode(ListNode head) {
    if (head == null || head.next == null) {
        return head;
    }

    ListNode fast = head;
    ListNode slow = head;

    while (fast != null && fast.next != null) {
        fast = fast.next.next;
        slow = slow.next;
    }

    return slow;
}
```



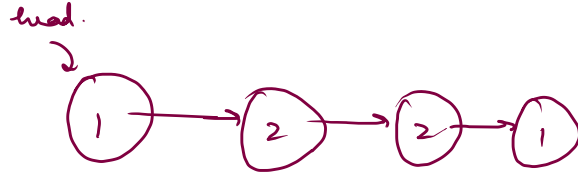

A
t = 0 hr
B.
t = T hrs
Ravi → y km/hr
Ravi → 2y km/hr
x/2
x km
Ravi

234. Palindrome Linked List

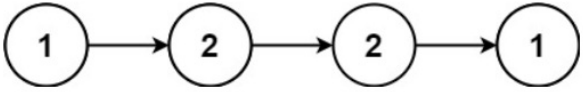
Easy 15.3K 819

Companies

Given the `head` of a singly linked list, return `true` *if it is a palindrome or* `false` *otherwise.*

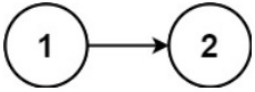


Example 1:



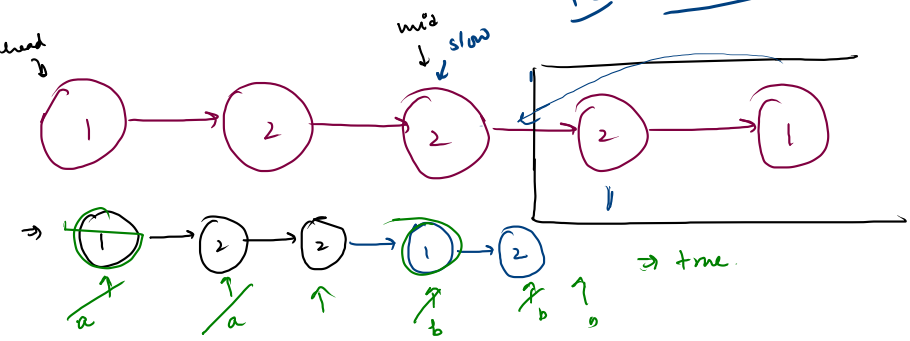

```
Input: head = [1,2,2,1]
Output: true
```

Example 2:




```
Input: head = [1,2]
Output: false
```

T.C O(n)

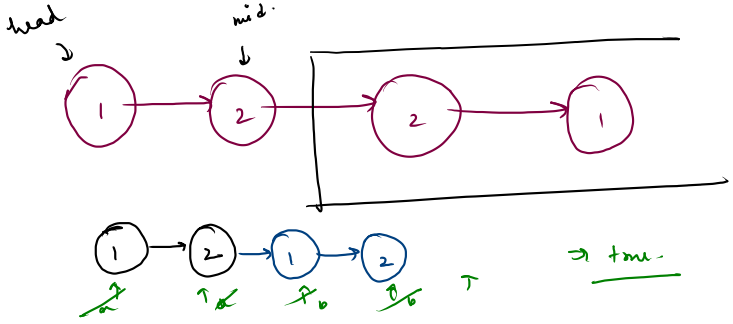


⇒ true.

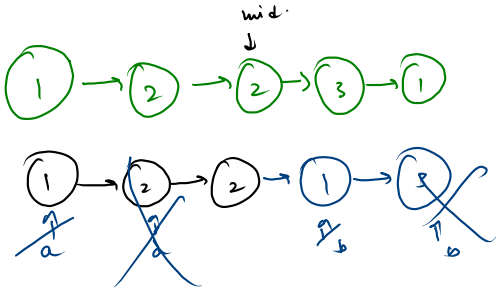
1) Find middle of LL

Is even → consider first as middle.

2) Reverse the LL after mid part.



⇒ true.



⇒ false

with n

ni + ni

```java
public boolean isPalindrome(ListNode head) {
    if (head == null || head.next == null) {
        return true;
    }

    ListNode fast = head;
    ListNode slow = head;

    while(fast.next != null && fast.next.next != null) {
        fast = fast.next.next;
        slow = slow.next;
    }

    slow.next = reverse(slow.next);

    ListNode a = head;
    ListNode b = slow.next;

    while (b != null) {
        if (a.val != b.val) {
            return false;
        }
        a = a.next;
        b = b.next;
    }

    return true;
}
```

```java
public ListNode reverse(ListNode head) {
    if (head == null || head.next == null) {
        return head;
    }

    ListNode prev = null;
    ListNode curr = head;

    while (curr != null) {
        ListNode fowd = curr.next;

        curr.next = prev;

        prev = curr;
        curr = fowd;
    }

    return prev;
}
```

→ 10 mins (code + Dry Run)

237. Delete Node in a Linked List

Medium 3.9K 1.1K

Companies

There is a singly-linked list `head` and we want to delete a node `node` in it.

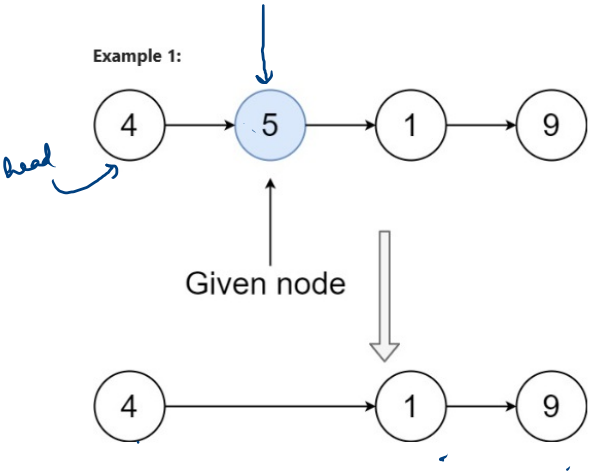
You are given the node to be deleted `node`. You will **not be given access** to the first node of `head`.

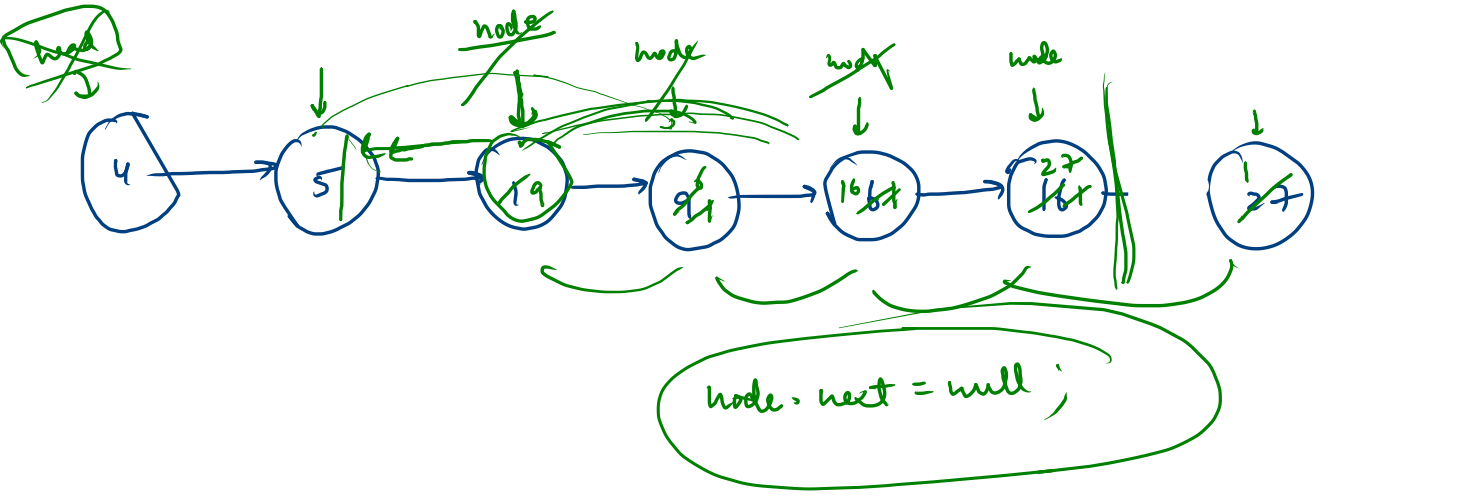
All the values of the linked list are **unique**, and it is guaranteed that the given node `node` is not the last node in the linked list.

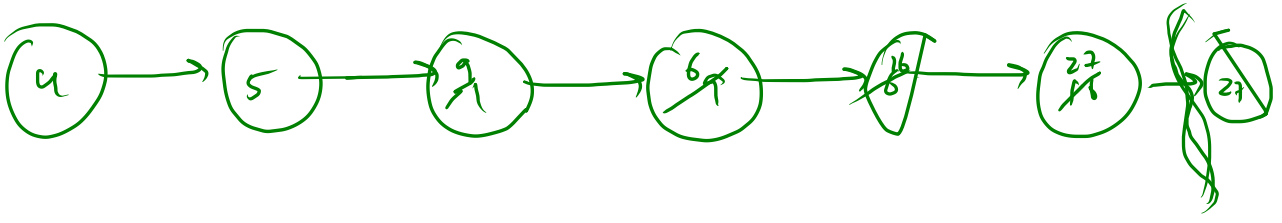
Delete the given node. Note that by deleting the node, we do not mean removing it from memory. We mean:

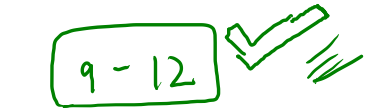
- The value of the given node should not exist in the linked list.
- The number of nodes in the linked list should decrease by one.
- All the values before `node` should be in the same order.
- All the values after `node` should be in the same order.

Example 1:




node
node
node
node
node
4
5
19
9
16
27
1
node.next = null;


4
5
9
6
8
27
27




```
class Solution {
    public void deleteNode(ListNode node) {
        while (node.next.next != null) {
            node.val = node.next.val;
            node = node.next;
        }

        node.val = node.next.val;
        node.next = null;
    }
}
```