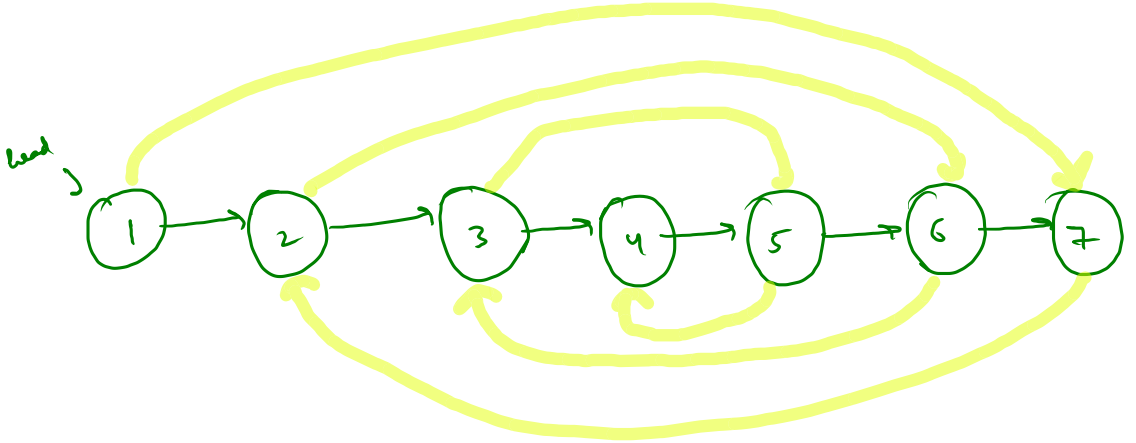
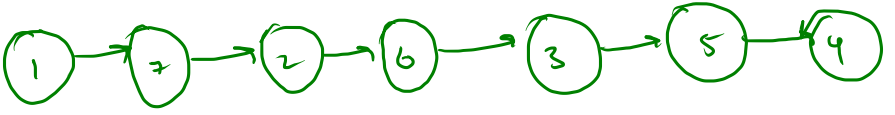
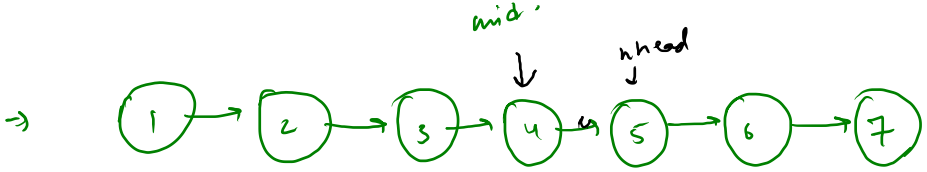
Fold a LL

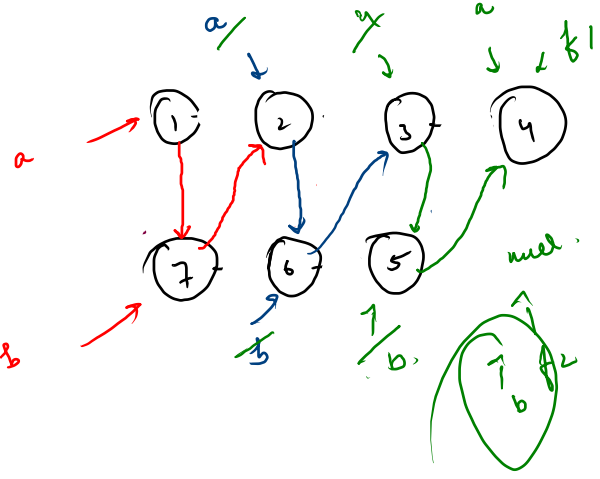


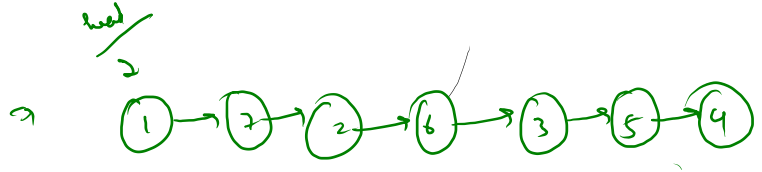
I.P.

O.P.









6) a.next = b;
b.next = f1

7) a = f1
b = f2

steps

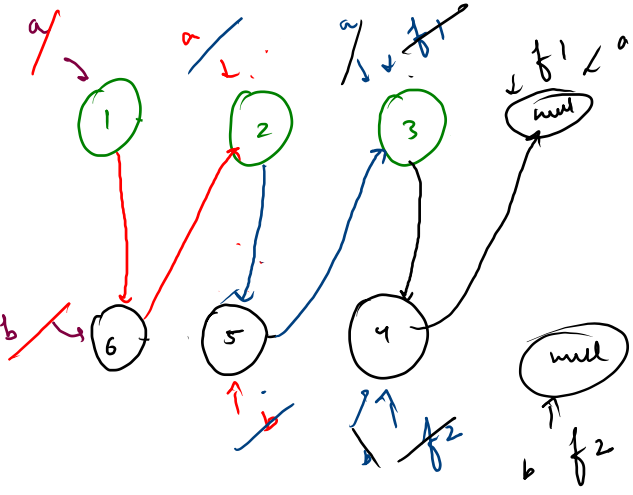
1) Find mid.

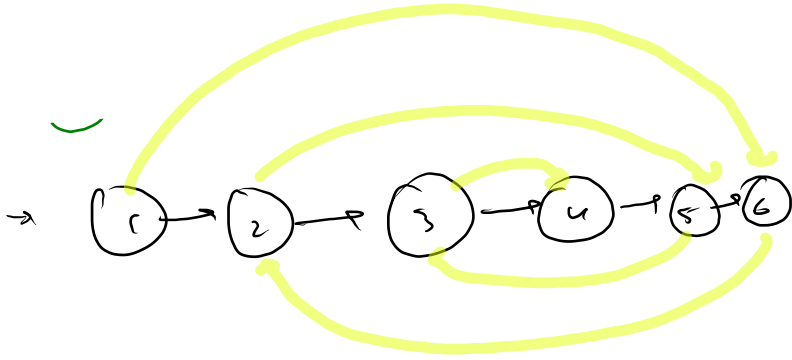
2) nhead = mid.next
mid.next = null;

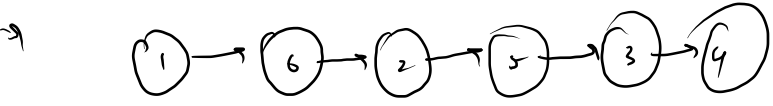
3) Reverse the nhead.

4) Make two pointers
a, b.
and these pointers
initially points to
head of both LL

5) f1 = a.next
f2 = b.next


a
1
2
3
null
b
6
5
4
null
b
f2


1
2
3
4
5
6


1
6
2
5
3
4

```
public static void fold(ListNode head) {
   if(head == null || head.next == null) {
       return;
   }

    ListNode slow = head;
    ListNode fast = head;

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

    ListNode nhead = slow.next;
    slow.next = null;

    nhead = reverse(nhead);

    ListNode a = head;
    ListNode b = nhead;


    while(a != null && b != null) {
        ListNode f1 = a.next;
        ListNode f2 = b.next;

        a.next = b;
        b.next = f1;

        a=f1;
        b=f2;
    }
}
```

```
public static 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;
}
```

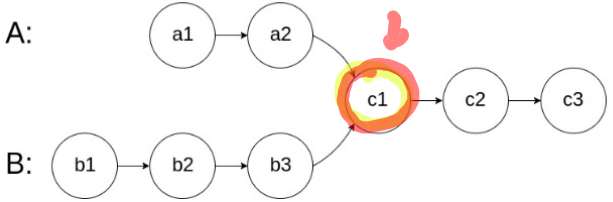
160. Intersection of Two Linked Lists

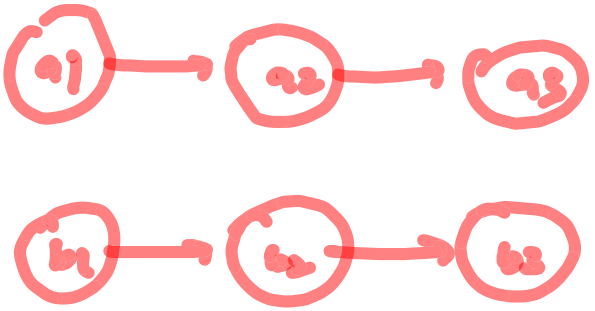
Easy 14.1K 1.2K

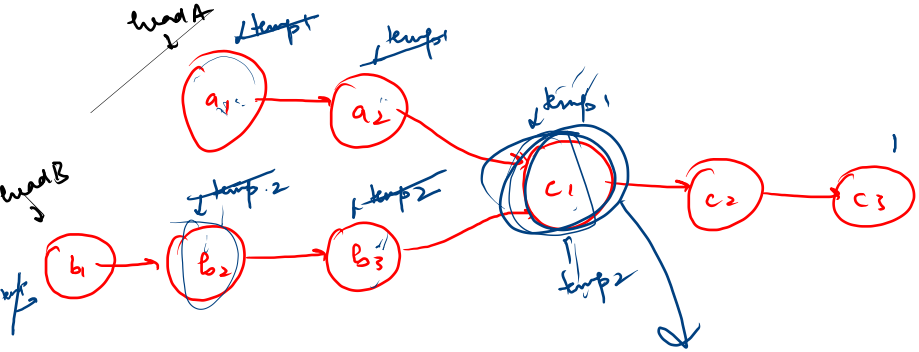
Companies

Given the heads of two singly linked-lists `headA` and `headB`, return *the node at which the two lists intersect*. If the two linked lists have no intersection at all, return `null`.

For example, the following two linked lists begin to intersect at node `c1`:







length A = 5

length B = 6

diff = 1

return c1

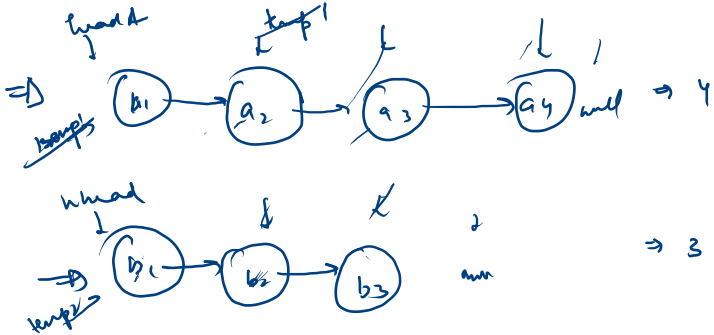
1) Find length of both the LL

⇒ length A = 5

⇒ length B = 6

2) Find diff"

abs (length A - length B)



```java
    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        if (headA == null || headB == null) {
            return null;
        }

        int lengthA = countNodes(headA);
        int lengthB = countNodes(headB);

        int diff = Math.abs(lengthA-lengthB);

        if (lengthA > lengthB) {
            for(int i=1;i<=diff;i++){
                headA = headA.next;
            }
        } else {
            for(int i=1;i<=diff;i++) {
                headB = headB.next;
            }
        }

        while(headA != null) {
            if (headA == headB) {
                return headA;
            }

            headA = headA.next;
            headB = headB.next;
        }
        return null;
    }
}
```

```java
public int countNodes(ListNode head) {
    int count = 0;
    ListNode temp = head;

    while (temp != null){
        count++;
        temp = temp.next;
    }

    return count;
}
```

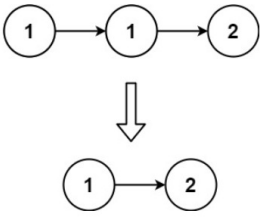
83. Remove Duplicates from Sorted List

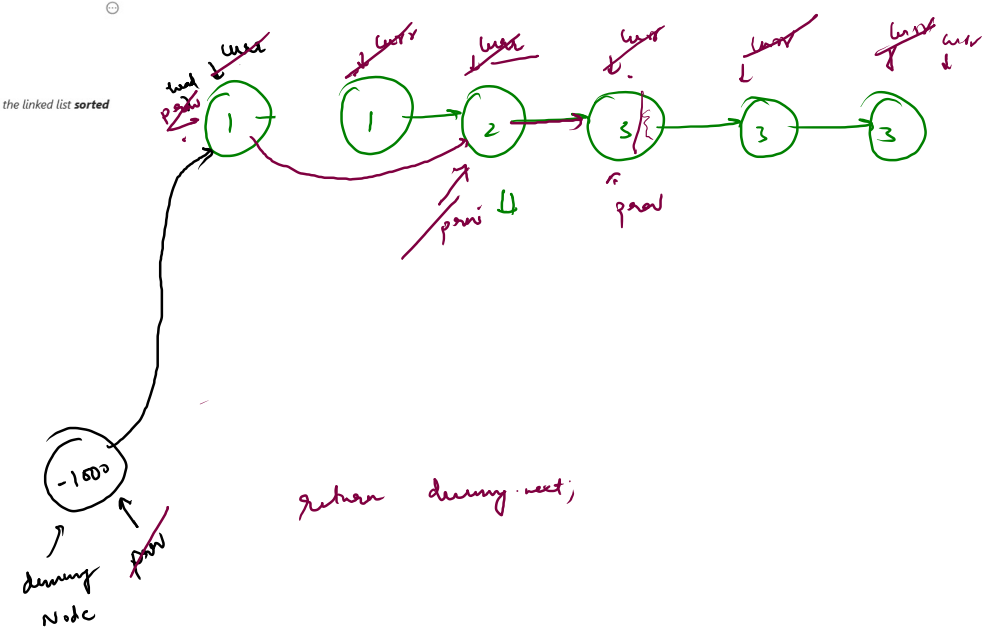
Easy 8K 271

Companies

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:





Steps

1) Make dummy & fill it some random data;

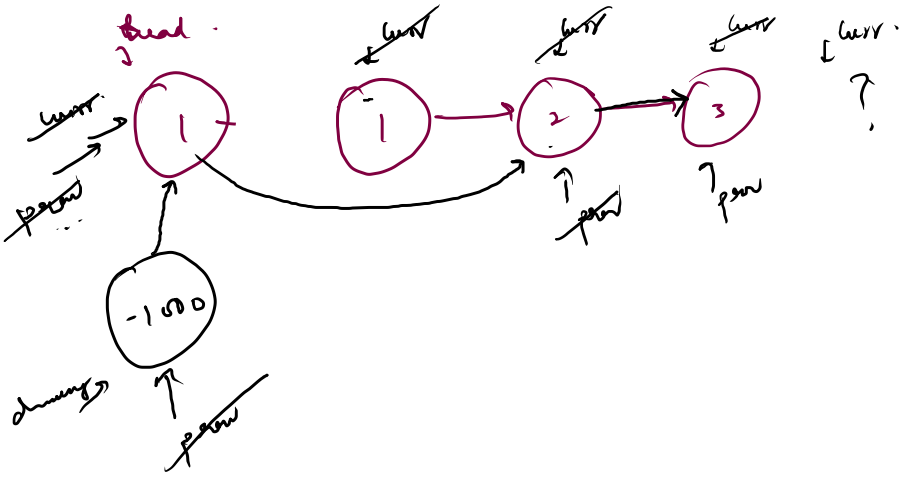
2) Make 2 pointers, points to head & dummy node.

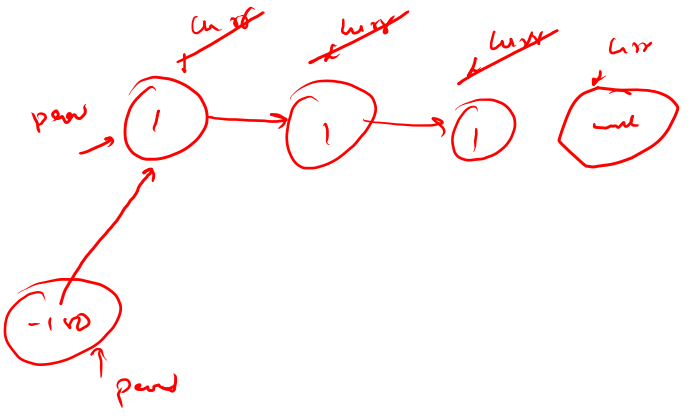
3) Apply the following until (curr != null)

↳ Check prev & curr data.

↳ if same → only move curr node.

↳ prev.next = curr.
prev = curr;
curr = curr.next;



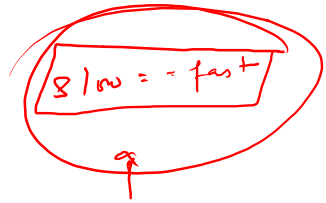


```java
public ListNode deleteDuplicates(ListNode head) {
    if(head == null || head.next == null) {
        return head;
    }

    ListNode dummyNode  = new ListNode(-1000);

    ListNode prev = dummyNode;
    ListNode curr = head;


    while (curr != null) {
        while (curr != null && prev.val == curr.val) {
            curr = curr.next;
        }

        prev.next = curr;

        prev = prev.next;

        if (curr != null) {
            curr = curr.next;
        }
    }

    return dummyNode.next;

}
```

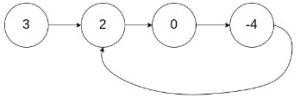
Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

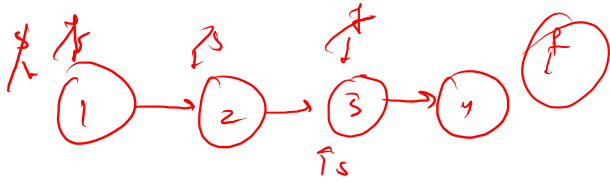
There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter**.

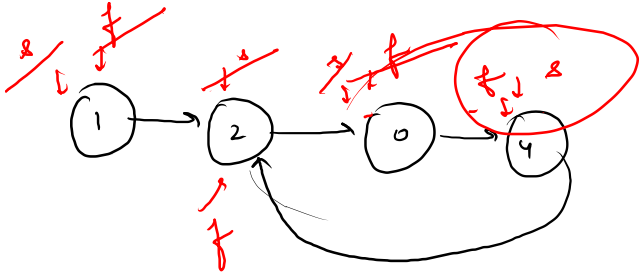
Return `true` *if there is a cycle in the linked list*. Otherwise, return `false`.



Example 1:







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

        ListNode slow = head;
        ListNode fast = head;

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

            if(slow == fast) {
                return true;
            }
        }

        return false;
    }
}
```

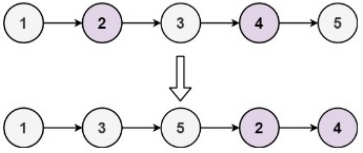
Given the `head` of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list*.

The **first** node is considered **odd**, and the **second** node is **even**, and so on.

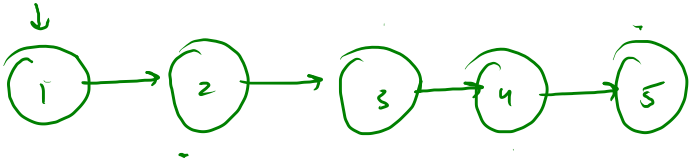
Note that the relative order inside both the even and odd groups should remain as it was in the input.

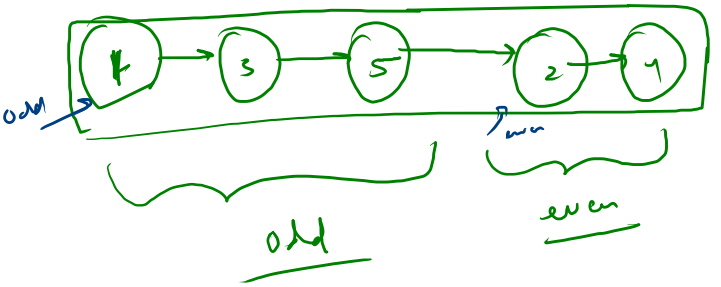
You must solve the problem in `O(1)` extra space complexity and `O(n)` time complexity.

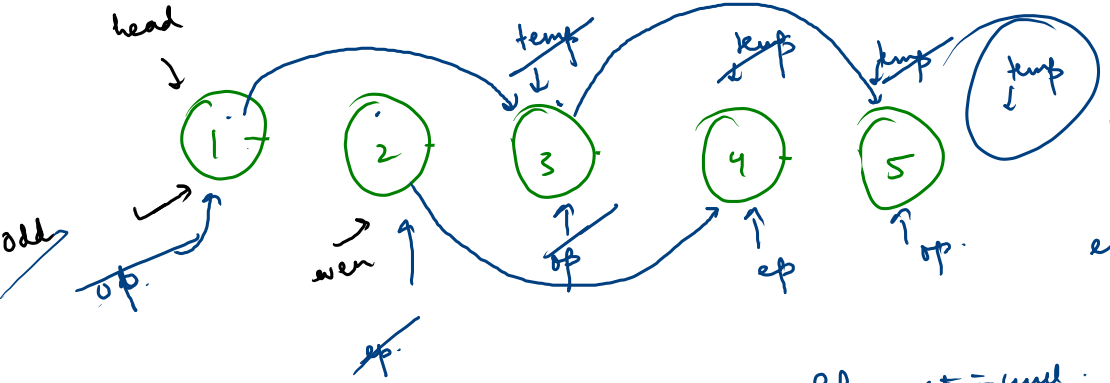
Example 1:




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

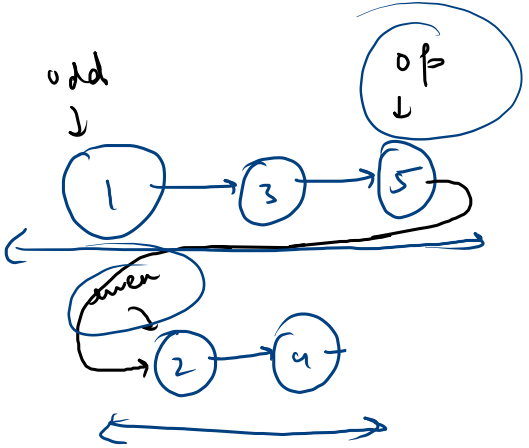






ep.next = null.

op.next = even.



→ 1 → 3 → 5 → 2 → 4

Steps:

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

2) even = head.next, odd = head, ep = even, op = odd.

count = 3, temp = even.next;

3) while(temp != null)
→ count → even
↳ ep.next = temp.
ep = temp.

odd ↳ op.next = temp
op = temp.

4) ep.next = null;
op.next = even;

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

    ListNode odd = head;
    ListNode even = head.next;

    ListNode op = odd;
    ListNode ep = even;
    ListNode temp = even.next;

    int count = 3;

    while (temp != null) {
        if(count%2 == 0) {
            ep.next = temp;
            ep =temp;
        } else {
            op.next =temp;
            op = temp;
        }

        temp = temp.next;
        count++;
    }

    ep.next = null;
    op.next = even;

    return odd;
}
```