Leetcode problems(LinkedList)

46⟶(876)Middle of the Linkedlist

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def middleNode(self, head: Optional[ListNode]) -> Optional[I
        temp=head
        c=0
        while temp!=None:
            c+=1
        for i in range(c//2):
            temp=temp.next
        return temp
```

47⟶(206)Reverse Linkedlist

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def reverseList(self, head: Optional[ListNode]) -> Optional
        prev=None
        next=None
        curr=head
        while curr!=None:
            next=curr.next
            curr.next=prev
            prev=curr
            curr=next
        head=prev
        return head
```

48⟶(234)Palindromic linkedlist

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def isPalindrome(self, head: Optional[ListNode]) -> bool:
        curr=head
        newnode=ListNode(curr.val)
        a=newnode
        curr=curr.next
        while curr!=None:
            new=ListNode(curr.val)
            a.next=new
            curr=curr.next
            a=a.next
        prev=None
        next=None
        curr=head
        while curr!=None:
            next=curr.next
            curr.next=prev
            prev=curr
            curr=next
        head=prev

        curr=head
        a=newnode
        while curr!=None:
            if curr.val!=a.val:
                return False
            curr=curr.next
            a=a.next
        else:
            return True
```

48⟶(876)Middle of the Linkedlist

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def middleNode(self, head: Optional[ListNode]) -> Optional[
        temp=head
        c=0
        while temp!=None:
            c+=1
            temp=temp.next
        temp=head
        for i in range(c//2):
            temp=temp.next
        return temp
```

49⟶(83)Remove duplicates from sorted list

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def deleteDuplicates(self, head: Optional[ListNode]) -> Opti
        temp=head
        next=None
        if head==None:
            return None
        while temp.next!=None:
            next=temp.next
            if temp.val!=next.val:
                temp=temp.next
            else:
                temp.next=next.next
        return head
```

50⟶(203)Remove List elements

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def removeElements(self, head: Optional[ListNode], val: int
        temp=ListNode()
        curr=temp
        temp.next=head
        while curr.next!=None:
            if curr.next.val==val:
                curr.next=curr.next.next
            else:
                curr=curr.next
        return temp.next
```

51→(2)Add two numbers

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def addTwoNumbers(self, l1: Optional[ListNode], l2: Optiona
        s=""
        s2=""
        while l1!=None:
            s=s+str(l1.val)
            l1=l1.next
        while l2!=None:
            s2=s2+str(l2.val)
            l2=l2.next
        s=s[::-1]
        s2=s2[::-1]
        a=int(s)+int(s2)
        a=str(a)
        a=a[::-1]
        q=str(a)
        z=ListNode()
        curr=z
        for i in q:
            new=ListNode(int(i))
            curr.next=new
            curr=curr.next
        return z.next
```

52⟶(21)Merged two lists

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def mergeTwoLists(self, list1: Optional[ListNode], list2: Op
        curr1=list1
        curr2=list2
        curr3=None
        while curr1!=None and curr2!=None:
            if curr1.val<=curr2.val:
                newnode=ListNode(curr1.val)
                temp=curr3
                if temp==None:
                    curr3=newnode
                else:
                    while(temp.next!=None):
                        temp=temp.next
                    temp.next=newnode
                curr1=curr1.next
            else:
                newnode=ListNode(curr2.val)
                temp=curr3
                if temp==None:
                    curr3=newnode
                else:
                    while temp.next!=None:
                        temp=temp.next
                    temp.next=newnode
                curr2=curr2.next
        while curr1!=None:
            newnode=ListNode(curr1.val)
            temp=curr3
            if temp==None:
                curr3=newnode
            else:
                while temp.next!=None:
                    temp=temp.next
                temp.next=newnode
            curr1=curr1.next
        while curr2!=None:
            newnode=ListNode(curr2.val)
            temp=curr3
            if temp==None:
                curr3=newnode
            else:
                while temp.next!=None:
                    temp=temp.next
                temp.next=newnode
            curr2=curr2.next
        return curr3
```

53⟶(160)Intersection of two linked lists

```
# Definition for singly-linked list.
# class ListNode:
#     def _init_(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNo
        a=headA
        b=headB
        c=0
        while a!=b:
            a=a.next
            b=b.next
            if a==None:
                a=headB
                c+=1
            if b==None:
                b=headA
                c+=1
            if c>=3:
                return None
        return b
```

54⟶(141)LinkedList cycle

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def hasCycle(self, head: Optional[ListNode]) -> bool:
        if head==None:
            return False
        fast=head
        slow=head
        while fast.next!=None and fast.next.next!=None:
            fast=fast.next.next
            slow=slow.next
            if fast==slow:
                return True
        return False
```

55→(237)Delete a node in alinkedlist

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def deleteNode(self, node):
        """
        :type node: ListNode
        :rtype: void Do not return anything, modify node in-plac
        """
        node.val=node.next.val
        node.next=node.next.next
```