Lab-3

(i)206

class Solution {

    public ListNode reverseList(ListNode head) {

        ListNode prev = null;

        ListNode current = head;

        while(current != null) {

            ListNode next = current.next;

            current.next = prev;

            prev = current;

            current = next;

        }

       return prev;

    }

}

(ii)876

class Solution {

    public ListNode middleNode(ListNode head) {

         ListNode slow, fast, new\_head;

        slow  = fast = head;

        while(fast != null  && fast.next != null)

        {

            slow = slow.next;

            fast = fast.next.next;

        }

        new\_head = slow;

        return new\_head;

    }

}

(iii)21

class Solution {

    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {

         while(list1 != null && list2 != null) {

            if(list1.val <= list2.val) {

                list1.next = mergeTwoLists(list1.next, list2);

                return list1;

            } else {

                list2.next = mergeTwoLists(list2.next, list1);

                return list2;

            }

        }

        return list1 == null ? list2 : list1;

    }

}

(iv)19

class Solution {

    public ListNode removeNthFromEnd(ListNode head, int n) {

         int sz = 0;

        ListNode temp = head;

        while (temp != null){

            temp = temp.next;

            sz++;

        }

        if (n == sz){

            head = head.next;

            return head;

        }

        int i = 1;

        int iToFind = sz - n;

        ListNode prev  = head;

        while (i < iToFind){

            prev  = prev.next;

            i++;

        }

        prev.next = prev.next.next;

        return head;

    }

}

(v)2

class Solution {

    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

        ListNode result = new ListNode();

        ListNode resultLink = result;

        int nextTenCount = 0;

        while (l1 != null && l2 != null) {

            result.val =  (l1.val + l2.val + nextTenCount) % 10;

            nextTenCount = (l1.val + l2.val + nextTenCount) / 10;

            if (l1.next != null || l2.next != null) {

                result.next = new ListNode();

                result = result.next;

            }

            l1 = l1.next;

            l2 = l2.next;

        }

        while (l1 != null) {

            result.val =  (l1.val + nextTenCount) % 10;

            nextTenCount = (l1.val + nextTenCount) / 10;

            if (l1.next != null) {

                result.next = new ListNode();

                result = result.next;

            }

            l1 = l1.next;

        }

        while (l2 != null) {

            result.val =  (l2.val + nextTenCount) % 10;

            nextTenCount = (l2.val + nextTenCount) / 10;

            if (l2.next != null) {

                result.next = new ListNode();

                result = result.next;

            }

            l2 = l2.next;

        }

        if (nextTenCount != 0) {

            result.next = new ListNode();

            result = result.next;

            result.val = nextTenCount;

        }

        return resultLink;

    }

}

(vi)237

class Solution {

    public void deleteNode(ListNode node) {

          node.val=node.next.val;

        node.next=node.next.next;

    }

}

(vii)160

public class Solution {

    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {

        if(headA==null || headB==null)return null;

        ListNode c1=headA;

        ListNode c2=headB;

        while(c1!=c2){

            if(c1==null) c1=headB;

            else c1=c1.next;

            if(c2==null) c2=headA;

            else c2=c2.next;

        }

        return c1;

    }

}

(viii)141

public class Solution {

    public boolean hasCycle(ListNode head) {

         ListNode fast = head;

        ListNode slow = head;

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

            fast = fast.next.next;

            slow = slow.next;

            if (fast == slow) {

                return true;

            }

        }

        return false;

    }

}

(ix)234

class Solution {

    public boolean isPalindrome(ListNode head) {

        ListNode slow = head, fast = head, prev, temp;

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

            slow = slow.next;

            fast = fast.next.next;

        }

        prev = slow;

        slow = slow.next;

        prev.next = null;

        while (slow != null) {

            temp = slow.next;

            slow.next = prev;

            prev = slow;

            slow = temp;

        }

        fast = head;

        slow = prev;

        while (slow != null) {

            if (fast.val != slow.val) return false;

            fast = fast.next;

            slow = slow.next;

        }

        return true;

    }

}

(x)142

public class Solution {

    public ListNode detectCycle(ListNode head) {

        ListNode slow = head, fast = head;

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

            slow = slow.next;

            fast = fast.next.next;

            if (slow == fast) break;

        }

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

        while (head != slow) {

            head = head.next;

            slow = slow.next;

        }

        return head;

    }

}

(xi)25

class Solution {

    public ListNode reverseKGroup(ListNode head, int k) {

         if(k==1) return head;

        ListNode d = new ListNode(0, head), \_node, c, lnode, t;

        int n=0;

        for(t=head;t!=null;n++, t=t.next);

        for(int j=0;j<n/k;j++){

            \_node=d.next; c=\_node.next; lnode=d.next;

            for(int i=1;i<k;i++){

                t=c.next;

                c.next=\_node;

                \_node=c;

                c=t;

            }

            lnode.next=c;

            d.next=\_node;

            d=lnode;

            if(j==0) head=\_node;

        }

        return head;

    }

}