链表：

链表定义：

```
public class ListNode{
    public int value;
    public ListNode next;
    public ListNode(int data){
        this.value=data;
    }
}
```

链表相关题：

1.从尾到头打印链表

解题思路：利用栈（或者递归，但是应用递归时，当链表长度非常长时有可能导致函数调用栈溢出）

```
//利用栈
public static void PrintListReverse(ListNode head){
    Stack<ListNode> nodes=new Stack<ListNode>();
    while(head!=null){
        nodes.push(head);
        head=head.next;
    }
    while(!nodes.isEmpty()){
        System.out.println(nodes.pop().value);
    }
}
//利用递归
public static void PrintListReverse1(ListNode head){
    if(head!=null){
```

```
            if(head.next!=null){
                PrintListReverse1(head.next);
            }
            System.out.println(head.value);
        }
    }
```

2.删除链表的节点

要求:时间复杂度为O(1)

解题思路：将待删除节点后一节点值复制到删除节点处，然后修改指针，如果是待删除节点是尾节点，则需要从头开始遍历，但是此方法基于删除节点存在于链表中

```
//删除链表中某节点（假设该节点存在于链表中，且时间复杂度为O(1)）
    public static void DeleteNode(ListNode head,ListNode delNode){
        if(head==null||delNode==null)
            return;
        //如果删除节点是头节点
        if(head==delNode)
            head=null;
        //删除节点为尾节点
        else if(delNode.next==null){
            while(head.next!=delNode)
                head=head.next;
            head.next=null;
        }
        //删除节点为链表中节点
        else{
```

```
            ListNode pos=delNode.next;
            delNode.value=pos.value;
            delNode.next=pos.next;
        }
    }
```

3.删除链表的倒数第k个节点

解题思路：遍历链表至尾节点（遍历过程中k--）,然后根据k值进行判断

```
    public static ListNode removeKth(ListNode head,int k){
        if(head==null||k<1)
            return head;
        ListNode cur=head;
        while(cur!=null){
            k--;
            cur=cur.next;
        }
        //倒数第k个节点为头节点
        if(k==0){
            head=head.next;
        }
        if(k<0){
            cur=head;
            while(++k!=0){
                cur=cur.next;
            }
            //cur为待删除节点的前驱
            cur.next=cur.next.next;
```

```
        }
        return head;
    }
```

4.链表中倒数第k个节点

解题思路：设置两个指针，当第一个指针走了k-1步时，第二个指针开始从头节点遍历，当第一个指针走到尾节点时，第二个指针所指就是倒数第k个节点

```
    public static ListNode PrintKth(ListNode head,int k){
        if(head==null||k==0)
            return null;
        ListNode ahead=head;
        ListNode behind=head;
        for(int i=1;i<k;i++){
            if(ahead.next!=null)
                ahead=ahead.next;
            //如果k大于链表节点数
            else
                return null;
        }
        while(ahead.next!=null){
            ahead=ahead.next;
            behind=behind.next;
        }
        return behind;
    }
```

5.合并两个有序链表

解题思路：

一种是在创建一个链表，然后依次判断连接

另一种是在现有链表上进行连接，空间复杂度较低，为O(1)

```
public static ListNode Merge(ListNode head1,ListNode
head2){
        if(head1==null)
            return head2;
        else if(head2==null)
            return head1;
        ListNode head = null;
        if (head1.value < head2.value) {
            head = head1;
            head.next = Merge(head1.next, head2);
        } else {
            head = head2;
            head.next = Merge(head1, head2.next);
        }
        return head;
    }

//空间复杂度低的形式(在头节点小的链表中合并，空间复杂度为O(1))
    public static ListNode Merge1(ListNode head1,ListNode
head2){
        if(head1==null||head2==null)
            return head1!=null?head1:head2;
        ListNode head=head1.value<head2.value?head1:head2;
        //cur1指向合并链表的头节点
        ListNode cur1=head==head1?head1:head2;
```

```
        ListNode cur2=head==head1?head2:head1;
        ListNode pre=null;
        ListNode pos=null;
        while(cur1!=null&&cur2!=null){
            if(cur1.value<cur2.value){
                pre=cur1;
                cur1=cur1.next;
            }
            else{
                pos=cur2.next;
                pre.next=cur2;
                cur2.next=cur1;
                pre=cur2;
                cur2=pos;
            }
        }
        //连接剩余链表
        pre.next=cur1==null?cur2:cur1;
        return head;
    }
```

6.反转链表

思路：记录前驱和后继，依次反转

```
    public static ListNode reverse(ListNode head){
        ListNode pre=null;
        ListNode pos=null;
        while(head!=null){
```

```
            pos=head.next;
            head.next=pre;
            pre=head;
            head=pos;
        }
        return pre;
    }
    //反转链表一部分
    public static ListNode reversePart(ListNode head,int
from,int to){
        int len=0;
        ListNode fpre=null;
        ListNode tpos=null;
        ListNode cur=head;
        //求出len
        while(cur!=null){
            len++;
            fpre=len==from-1?cur:fpre;
            tpos=len==to+1?cur:tpos;
            cur=cur.next;
        }
        if(from>to||from<1||to>len){
            return null;
        }
        ListNode node1=fpre==null?head:fpre.next;
```

```
        ListNode node2=node1.next;
        node1.next=tpos;
        ListNode pos=null;
        //node1为from节点
        while(node2!=tpos){
            pos=node2.next;
            node2.next=node1;
            node1=node2;
            node2=pos;
        }
        if(fpre!=null){
            fpre.next=node1;
            return head;
        }
        return node1;
    }
```

7.两个链表的第一个公共节点

解题思路：第一个公共节点之后的两个链表是重合的，所以先求出两个链表的长度差，让较长链表先走一部分，然后二者一起移动找到相同节点

```
public static ListNode commonLN(ListNode head1,ListNode head2){
        int len1=0;
        int len2=0;
        ListNode node1=head1;
        ListNode node2=head2;
        while(node1!=null){
```

```
            len1++;
            node1=node1.next;
        }
        while(node2!=null){
            len2++;
            node2=node2.next;
        }
        int distance=len1>len2?len1-len2:len2-len1;
        ListNode cur1=len1>len2?head1:head2;
        ListNode cur2=len1>len2?head2:head1;
        while(distance!=0){
            distance--;
            cur1=cur1.next;
        }
        while(cur1!=null&&cur2!=null&&cur1.value!=cur2.value)
{
            cur1=cur1.next;
            cur2=cur2.next;
        }
        return cur1;
    }
```

8.打印两个有序链表的公共部分

思路;节点值小的移动直到找到相等点打印

```
public static void PrintCommon(ListNode head1,ListNode
head2){
        while(head1!=null&&head2!=null){
```

```
            if (head1.value < head2.value)
                head1 = head1.next;
            else if (head2.value < head1.value)
                head2 = head2.next;
            else {
                System.out.println(head1.value);
                head1=head1.next;
                head2=head2.next;
            }
        }
        System.out.println();
    }
```

9.找到链表中环的入口节点

解题思路：1）先确定链表中是否包含环（利用一快一慢指针，一个一次走一步，另一个一次走两步）

2）首先求出环中节点个数n，其中一个指针先走n步，后一个指针开始，二者重合处就是环的入口节点

```
public  static ListNode ENOfLoop(ListNode head){
        if(head==null||head.next==null)
            return null;
        ListNode ahead=head.next;
        ListNode behind=ahead.next;
        ListNode meetNode=null;
       //判断是否有环
        while(ahead!=null&&behind!=null){
            if(ahead==behind){
```

```
            meetNode=ahead;
            break;
        }
        ahead=ahead.next;
        behind=behind.next;
        if(behind!=null)
            behind=behind.next;
    }
    if(meetNode==null)
        return null;
    //求出环中节点数count
    int count=1;
    ListNode cur=meetNode;
    while(cur.next!=meetNode){
        count++;
        cur=cur.next;
    }
    //利用两个指针，一个先走count步，另一个开始走，二者重合点就是入口点
    ListNode node1=head;
    for(int i=0;i<count;i++)
        node1=node1.next;
    ListNode node2=head;
    while(node1!=node2){
        node1=node1.next;
        node2=node2.next;
```

```
    }
    return node1;
}
```