今日所学

* 对于删除链表，如果头指针是指向第一个元素，需要在新建一个超前指针newHead，这样当第一个元素也需要被删除的时候就很方便了。
* 判断是否是2的幂，可以根据这个数n和n-1相与，如果为0，则是2的幂。
* 反转二叉树，可以用递归和迭代来求，归根结底就是将每个节点的左右子树互换
* 反转链表，除了用双指针，还可以用递归的方式。

题目一

#### **[204. Count Primes](https://leetcode-cn.com/problems/count-primes/)----Sieve of Eratosthenes算法**

常规的解法这里就不说了，都会做，介绍一下新的高效算法，如果2是一个素数，那么2\*2，2\*3，2\*4...2\*i都不是素数了，同理如果3是素数，那么3\*2，3\*3，3\*4...也都不是了。

* 外部循环表示不需要到n3而是n的根号因为：



当过了12的根号以后，乘积前后反过来了，所以如果在[2,sqrt(n)]这个区间没有发现可除的因子，就可以断定n是素数了，后面只是相反了而已。

* 同时内部循环并不需要都从2开始，因为会有很多冗余，比如

I=2时 会同时计算2\*2，2\*3，2\*4...

I=3时，又计算了3\*2，3\*3，3\*4

I=4时，计算了4\*3，4\*3，4\*4

发现，对i来说不需要在乘比他小的数了，因为在前面比他小的数早就乘过了，所以只需要从j=i开始

源代码如下：

    public int countPrimes(int n) {

        boolean[] isPrime=new boolean[n];

        Arrays.fill(isPrime,true);

        for(int i=2;i<=Math.sqrt(n);i++)

        {

            if(!isPrime[i])continue;

            for(int j=i;i\*j<n;j++)isPrime[i\*j]=false;

        }

        int cnt=0;

        for(int i=2;i<n;i++)

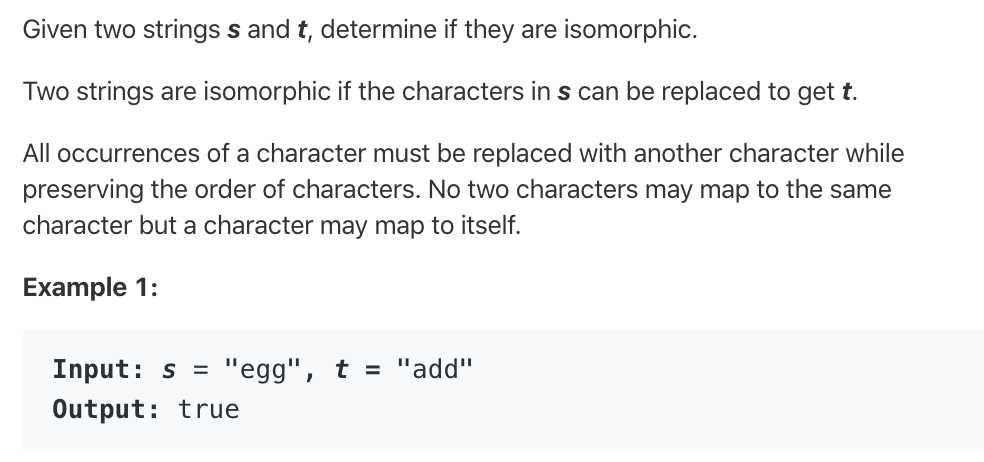
            if(isPrime[i])cnt++;

        return cnt;

    }

题目二

#### [205. Isomorphic Strings](https://leetcode-cn.com/problems/isomorphic-strings/)（同构字符串）



就是s和t的每个字符都要一一对应

1.首先想到的就是hashmap，重复判断两次，不能只判断一次，因为hashmap只能判断key是否有重复，并不会判断value是否有，所以先把s的元素做key，t的元素做value，之后在反过来判断一次。

2.这里再介绍一个新的方法，就是直接将两个字符都映射到新的类型上去，比如



初始值都为0，表示这个字符都还没出现过，当出现过以后就在该元素ASCII对应的值作为下标赋值，值就是第几个位置出现，如果是第一个元素，就赋1，第二个元素出现，就赋2.注意这里从i+1不能从i开始，因为但i=0的时候如果赋i，会导致出错。

源代码如下：

public boolean isIsomorphic(String s, String t) {

     int[] mapS=new int[128];

     int[] mapT=new int[128];

     for(int i=0;i<s.length();i++){

         if(mapS[s.charAt(i)]==0&&mapT[t.charAt(i)]==0)

         {

             mapS[s.charAt(i)]=i+1;

             mapT[t.charAt(i)]=i+1;

         }

         if(mapS[s.charAt(i)]!=mapT[t.charAt(i)])

         return false;

     }

     return true;

    }

题目三

#### [206. Reverse Linked List](https://leetcode-cn.com/problems/reverse-linked-list/)

使用双指针的方法，肯定要掌握，这里又学到了用递归求解。通过递归一直到链表的最后一个指针（非null指针），然后用

head.next.next=head;

来将head的下一个元素指向head，也就是head和head.next反转了方向，反转完以后要让head.next=null，不然head 和head.next会形成循环，最后返回last，这里的last一直都是链表的最后一个元素，是不会变的。

源代码如下：

//用递归来反转链表

    public ListNode reverseList(ListNode head) {

        //如果链表没有元素，或只有一个元素，直接返回

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

        //此时返回的一直是链表的最后一个元素，让head不停的往回移动一格

        ListNode last=reverseList(head.next);

        head.next.next=head;

        head.next=null;

        return last;

    }