一、约数

1.试除法求约数

求一个正整数x的所有约数,从1~sqrt(x)判断即可。

```
void get_divisors(int x)
{
    int i;
    for(i=1;i<x/i;i++)
    {
        if(x%i==0)
        {
                cout<<i<<" "<<x/i<<" ";
        }
    }
    if(i*i==x)
    cout<<i<<" ";
}
```

2.gcd(greatest common divisor)

(1)最大公因数

a,b都为正整数，a>b,那么最大公约数是既能整除a也能整除b且是最大的那个数。

(2）暴力找最大公因数

从b~1依次查找，能被a,b整除则break;

```
    int a=64,b=48;
    for(int i=b;i>=1;i--)
    {
        if(a%i==0&&b%i==0)
        {
            cout<<i<<endl;
            break;
        }
    }
```

(3)辗转相除法（欧几里得算法)

举个例子温习一下~

```
求64和48最大公因数
64=48*1+16;
48=16*3+0;
余数为0结束，结果为16.
```

纵观全局，我们将问题从求64和48最大公约数转化为求12和6最大公约数，为什么呢?

这就是我们的欧几里得定理：`gcd(a,b)=gcd(b,a mod b)`

证明很简单，设c为a,b最大公约数，a=c*q1,b=c*q2,那么余数一定是这样的形式c*（q1-q2*q3),则c一定是b和a%b的约数，那为啥是最大公约数呢？首先q1和q2互质，因为c是a,b最大公约数，那么q2和（q1-q2*q3)也一定互质，（反证：如果不互质，可以推出q1和q2不互质)

所以问题一步步简化啦，我们可以用循环来做，直到b=0结束，返回a值。

```cpp
    int a=64,b=48;
    while(b!=0)
    {
        int t=a%b;
        a=b;
        b=t;
    }
    cout<<a<<endl;
```

不过最通用的是下面这行代码，大家可以思考一下~

```cpp
int gcd(int a,int b)
{
    return b==0?a:gcd(b,a%b);
}
```

这行代码实现了求a,b最大公约数，但是我们大前提是a>b,那么a<b可以直接用吗？自己模拟一下康康会发生什么?

(4)扩展

1.求a,b最小公倍数(lcm：least common multiple);

lcm(a,b)=a*b/(gcd(a,b));

https://www.acwing.com/problem/content/811/

https://www.acwing.com/problem/content/871/

二、快速幂

1.定义

求a的b次方mod p值。

2.暴力

直接实现b个a相乘（库函数pow()做法),时间复杂度为o(n),对较大的数据易超时。

```
#include<iostream>
#include<math.h>
using namespace std;
const int p=1e9+7;
int main()
{
    int a=2,b=3;
    cout<<pow(2,3)<<endl;//math头文件
    int ans=1;
    for(int i=1;i<=b;i++)
    {
        ans=ans*a%p;
    }
    cout<<ans<<endl;
    return 0;
 }
```

3.快速幂

我们可以将b用二进制表示，举个例子哈，$5 = 101 = 1 * 2^0 + 0 * 2^1 + 1 * 2^2$, 从而 2^5 可写成$2^{1*2^0+0*2^1+1*2^2}$, 再拆成连乘的形式$2^{1*2^0} * 2^{0*2^1} * 2^{1*2^2}$ 玄机了么，由之前的5个数相乘变成了3个数相乘，实际上是由o(n)优化到了o(logn),这个时候你可能会疑惑，可是括号内的一堆堆比如算2^{100}不是要花很多时间吗？但仔细观察，括号里的每一项系数要么是0或1，抛开系数来看他们都是上一次的2倍，整个a^b值的每一项是前一项的平方。

怎么取一个数的每一位?

```cpp
#include<iostream>
using namespace std;
typedef long long ll;
ll quick_power(ll a,ll b,ll p)
{
    ll ans=1%p;
    while(b!=0)
    {
        if(b&1)//取b最后 一位，是0不操作，1则答案*a
        ans=ans*a%p;
        b>>=1;//b的倒数第二位变成最后一位
        a=a*a%p;
    }
    return ans;
}
int main()
{
    ll a,b,p;//数据比较大，一般用longlong
    cin>>a>>b>>p;
    cout<<quick_power(a,b,p);
    cout<<endl;
    return 0;
}
```

https://www.acwing.com/problem/content/877/

三、质数

1.判定质数

1.质数定义：大于1的自然数，只能被1和自身整除。注意：1不是质数也不是合数。

2.如何判定呢？从2~n-1遍历一遍，如果存在被n整除的数，则n不是质数。想想咋优化呢?嘿嘿可以不需要遍历整个n啦，只要到根号n就可以，因为约数是成对存在的，sqrt(n)之前没有，那之后也不存在！注：sqrt函数是求根号下取整，如：sqrt(4)=2,sqrt(3)=1;

```
bool is_prime(int x)
{
    if(x<2)
    return false;
    for(int i=2;i<=x/i;i++)//i<=sqrt(n),i*i<=n 三种实现方法都可以，推荐第一种哦，第二个耗时长，第三个可能会爆int
    {
        if(x%i==0)
        return false;
    }
    return true;
}
```

2.分解质因数

1.唯一分解定理：任何一个大于1的自然数 N,如果N不为**质数**，那么N可以唯一分解成有限个质数的乘积。

2.那怎么求他的质因数呢?

质因数在$2\sim n$之间，所以$2\sim n$遍历，如果找到一个因数i，先求这个因数的个数s，然后 n/i 执行s次分解，再往后找其他因子。想想这样子找出来的怎么保证他是**质**因子呢？？我们是从小到大依次找的，在找到因子i之前，我们已经对i之前的数都处理一遍了，i 既然还能存在说明他就是一个质数，或者反证，i不是质数的话，一定有$2\sim i-1$之间的一个约数，但是，我们之前都已经除干净了啊，这个时候的n不应该再有2~i-1期间的因子啦！

3.优化一下~。我们可以保证如果存在>sqrt(n)的一个质因子，那么他有且只有一个，所以，我们先找出$2\sim\sqrt{n}$的质因子，最后n如果>1的话说明存在大于 $\sqrt{n}$ 的因子并且就是 n.

```
void divide(int n)
{
```

```
    for(int i=2;i<=n/i;i++)
    {
        if(n%i==0)//i能被n整除
        {
            int s=0;
            while(n%i==0)//找个数
            {
                s++;
                n/=i;
            }
            cout<<i<<" "<<s<<endl;
        }
    }
    if(n>1)
    cout<<n<<" "<<"1"<<endl;
}
```

3.质数筛

1.背景：之前我们已经会判断一个数是不是质数了，那么现在如果给你一个区间，判断里头每一个数是不是质数，或者说筛出里头的质数，怎么做呢?

2.直接枚举啦，区间1-n,那么我一个个判断就可以了，但是好费时间，每一次都要从2~sqrt(i),冗余工作出在哪呢？brainstorm!!当我们确定一个数是质数的时候，那么他的倍数一定是合数啦，这样就可以利用之前已经判断好的结果快速解答，而不用从头开始，如果到i了他还没被筛去，那么前面i-1个数中不存在他的因子，所以i一定是质数，但是想想，这样能将所有合数筛去吗？of course,我们将所有质数的倍数筛去了，而一个数总能写成若干质数和1的乘积，所以如果是合数一定被筛去了。接下来介绍两种筛法。

改进：所有数筛——>质数筛——>最小质数筛。

1.埃氏筛法

就是上面这种方法啦，拿质数去筛，将它的倍数都筛掉。

（ps：大家看代码的时候可能会有很多不理解的地方，不知所云，我也是刚学会不久~所以很是理解，先将局部关键的弄明白啦，再回头整理思路比较好（顿悟派~）

```
const int N=1e6+5;
int st[N],prime[N],cnt=-1;//prime存储质数。st存是否被筛掉，被筛掉了（合数）1，否则存0。cnt是prime当前位置
void get_prime(int n)
{
    memset(st,0,sizeof(st));//st数组初始化0
    for(int i=2;i<=n;i++)
    {
        if(!st[i])//没被筛掉，说明是质数
        {
            prime[++cnt]=i;//i是质数存进去
            for(int j=i+i;j<=n;j=j+i)//将2-n内i的倍数筛掉
            {
                st[j]=1;
            }
        }
    }
}
```

2.线性筛法（欧拉筛法）

线性筛法是对埃氏筛法的改进，埃氏筛法一个合数可能会被重复筛去，这就是可以优化的地方，那么我们改进一下，每一个数只能被他的最小质因子筛去，这样时间复杂度将降低啦，那怎么保证一个数只能被最小质因子筛去呢？这是本算法的关键

```
int prime[N],st[N],cnt=-1;//prime存储质数。st存是否被筛掉，被筛掉了（合数）1，否则存0。cnt是prime当前位置
void get_prime(int n)
{
    memset(st,0,sizeof(st));//初始化为0
    for(int i=2;i<=n;i++)//判断1~n是否是质数
    {
        if(!st[i])
        prime[++cnt]=i;//没被筛去存进prime
        for(int j=0;prime[j]<=n/i;j++)//从最小的质数来筛
```

```
        {
            st[prime[j]*i]=1;//prime[j]*i一定是合数，并且prime[j]一定是prime[j]*i的最小质因子，因为是从最小的质数开始            筛哒！
            if(i%prime[j]==0)break;//如果prime[j]是i的因数，说明之后prime[j]不再是prime[j]*i的最小质因子，退出
        }    （//敲重点！！这2行代码是关键，可以先读第二行再读第一行）
    }
}
```

https://www.acwing.com/problem/content/869/

https://www.acwing.com/problem/content/451/

https://www.acwing.com/problem/content/870/

四、高精度算法

1.概念：

c++里int类型最大数是2*10^9左右，long long 类型10的19左右，现在给你两个1000位的数，注意是1000位！1后面跟了1000个0！！目标是计算这两个数的加减乘除，束手无策毫无办法？不不不，我们可以用一个数组来存这个数的所有位，手动模拟四大运算。霍霍其实这个比上面几大算法都简单哦。思路其实不复杂，直接看代码比较好。

1.字符串逆序转化为数

2.再运算。

2.高精度加法

现在小脑袋里快速运转一个78+156的加法，最右边对齐，然后从最右边开始加，满10进位。所以倒着存两个数组更方便哦。直接看代码比较好=_=

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+5;
int main()
{
    string s1,s2;
    cin>>s1>>s2;
    int n1=s1.length();//求s1长度
    int n2=s2.length();
    int a[N],b[N],top1=-1,top2=-1;
    memset(a,0,sizeof(a));//初始化
    memset(b,0,sizeof(b));
    for(int i=n1-1;i>=0;i--)
    a[++top1]=s1[i]-'0';//a数组存s1的逆序，b数组存s2的逆序，因为输入是字符输入，所以记得-"0"
    for(int i=n2-1;i>=0;i--)
    b[++top2]=s2[i]-'0';
    int i=0,t=0;//t表示进位，不进位t=0
    int mx=max(top1,top2);
    while(i<=mx)//加完后的数保存到a中
    {
        int s=a[i]+b[i]+t;//求新数
        a[i]=s%10;//新数个位存a数组
        t=s/10;//新数十位为进制
        i++;
    }
    if(t!=0)
    cout<<t;//最后都算完了，进制不为0的记得输出t,比如2+9
    for(int i=mx;i>=0;i--)
    cout<<a[i];//逆序输出
    cout<<endl;
    return 0;
}
```

3.高精度减法

再做一个减法，无论是算123-32还是32-123，我们都是拿较大的数减较小的数，然后看要不要加负号，而且也要逆着存数哦。看代码！

ps: vector也是一个容器来存储数，和数组的功能差不多，不过有很多方便的地方，比如可以直接求这一列数的长度等等，是c++封装好的一个类，可以在csdn上搜一下c++vector基本操作哦，push_back是插入一个数，pop_back是弹出一个数。

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+5;
bool cmp(vector<int>&A,vector<int>&B)//compare:比较A，B大小
{
    if(A.size()!=B.size())
    return A.size()>B.size();//直接根据长度判定大小
    for(int i=A.size()-1;i>=0;i--)//长度一样比较具体位大小
    {
        if(B[i]>A[i])
        return 0;
        if(B[i]<A[i])
        return 1;
    }
    return 1;
}
vector<int> sub(vector<int>&A,vector<int>&B)//substract:A-B
{
    int t=0;//进位为1或0
    vector<int> C;
    for(int i=0;i<A.size();i++)
    {
        int s=A[i]-t;//算a位值-进制
        if(i<B.size())//当b数组已经减完了的时候就不用减啦
        s=s-B[i];
        if(s<0)//需要往前借10
        {
            C.push_back(s+10);
            t=1;
```

```
        }
        else
        {
            C.push_back(s);
            t=0;
        }
    }
    while(C.size()>1&&C.back()==0)//前置0删除
    {
        C.pop_back();
    }
    return C;
}
int main()
{
    string a,b;
    cin>>a>>b;
    vector<int> A,B,C;
    for(int i=a.size()-1;i>=0;i--)
    A.push_back(a[i]-'0');//逆序存入A中，
    for(int i=b.size()-1;i>=0;i--)
    B.push_back(b[i]-'0');//逆序存入B中
    if(cmp(A,B))//如果A≥B
    C=sub(A,B);
    else
    {
        cout<<"-";
        C=sub(B,A);
    }
    for(int i=C.size()-1;i>=0;i--)//逆序输出
    cout<<C[i];
   return 0;

}
```

4.高精度乘法

高精度乘法是一个高精度数乘以另一个int范围内的数，和我们平常的乘法不太一样，比如203*34,并不是算4*203,3*203,最后对齐相加，而是将int型数b看做一个整体，与高精度数B每一位做乘法，规则就和203*4这样计算，不过进位就不是1~9了，可能会更大。

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+5;
vector<int> mul(vector<int>&B,int b)
{
    vector<int> C;
    int t=0;
    for(int i=0;i<B.size();i++)//B的每一位与b做运算
    {
        t=(t+B[i]*b);
        C.push_back(t%10);
        t=t/10;
    }
    //比如19*8，最后结果是152，我们只存了2,5，最后还要将1也存进去，也就是最后的进位
    while(t!=0)
    {
        C.push_back(t%10);
        t/=10;
    }
    return C;
}
int main()
{
    string A;
    int b;
    cin>>A>>b;
    vector<int> B;
    for(int i=A.size()-1;i>=0;i--)
    B.push_back(A[i]-'0');//逆序存储
    vector<int> C;
    C=mul(B,b);
    if(b==0)
    cout<<"0"<<endl;//结果为0时防止输出00000
    else
    for(int i=C.size()-1;i>=0;i--)
    cout<<C[i];
    return 0;
```

```
}
```

5.高精度除法

一个高精度数除以一个int型整数。

高精度除法比乘法更简单，按照平时的流程运行一个除法就可以啦！先在草稿纸上算算802/3，最后的余数不用管，赶紧温习一下~

正序存。

```
#include<iostream>
#include<string>
#include<vector>
using namespace std;
const int N=2e5+5;
typedef long long ll;
int main()
{
    string s;
    int b;
    cin>>s>>b;
    vector<int> A,B;//A中存高精度数，B中存结果
    for(int i=0;i<s.size();i++)
    A.push_back(s[i]-'0');//正序存到A中
    int t=0;//t表示上一位数字
    for(int i=0;i<A.size();i++)
    {
        t=t*10+A[i];
        B.push_back(t/b);
        t=t%b;
    }
    int i=0;
    while(B[i]==0&&i<B.size()-1)//去掉前置0，是B.size()-1不是B.size()
    i++;
    for(;i<B.size();i++)
    {
```

```
        cout<<B[i];
    }
    cout<<endl<<t<<endl;
    return 0;
}
```

https://www.acwing.com/problem/content/793/

https://www.acwing.com/problem/content/794/

https://www.acwing.com/problem/content/795/

https://www.acwing.com/problem/content/796/