素数

暴力解法

```
bool isPrime(int val) {
  if (val < 2) return 0;
  for (int i = 2; i < val; ++i)
    if (val % i == 0) return 0;
  return 1;
}
```

优化

很容易发现这样一个事实：如果 x是 a的约数，那么 $\frac{a}{x}$也是a 的约数。

这个结论告诉我们，对于每一对$(x, \frac{a}{x})$ ，只需要检验其中的一个就好了。为了方便起见，我们之考察每一对里面小的那个数。不难发现，所有这些较小数就是$[1, \sqrt{a}]$ 这个区间里的数。

由于 1 肯定是约数，所以不检验它。

```
bool isPrime(int val) {
  if (val < 2) return 0;
  for (int i = 2; i * i <= val; ++i)
    if (val % i == 0) return 0;
  return 1;
}
```

威尔逊定理

在初等数论中，威尔逊定理给定了判定一个数是否为素数的充分必要条件。即：当 p 为素数时，$(p-1)! \equiv -1 \ (mod\ p)$。等价的写法有 $(p-1)! \equiv p-1 \ (mod\ p)$、$p \mid (p-1)!+1 \equiv 0$.

由于阶乘是呈爆炸式增长，其结论实际操作意义不大。但是可以用来化简某些式子。

唯一分解定理

定义

任意一个大于 0 的正整数都能被表示成若干个素数的乘积且表示方法是唯一的；整理可以将相同素数的合并；可以得到 ：$n = {p_1}^{a_1}{p_2}^{a_2}{p_3}^{a_3}......{p_n}^{a_n}$；

```
typedef long long ll;
ll fac[10050], num;
void init(ll n) {
    num = 1;
    ll cpy = n;
    for (int i = 2; i*i <= n; ++i) {
        if (cpy % i == 0) {
            while( cpy % i == 0){
                cpy /= i;
                fac[num++] = i;
            }
        }
    }
```

```
    if (cpy > 1) fac[num++] = cpy;
}
```

反素数(codeforces 27E 有能力的去写一下)

定义

如果某个正整数 n满足如下条件，则称为是反素数： 任何小于n 的正数的约数个数都小于n 的约数个数

特点

1.素数肯定是从 2 开始的连续素数的幂次形式的乘积。

2.数值小的素数的幂次大于等于数值大的素数，即 $n = {p_1}^{a_1}{p_2}^{a_2}{p_3}^{a_3}......{p_n}^{a_n}$ 中，有

$a_1 \geq a_2 \geq a_3 \geq ...... \geq a_n$

解释

1. 如果不是从 2 开始的连续素数，那么如果幂次不变，把素数变成数值更小的素数， 那么此时因子个数不变，但是n 的数值变小了。交换到从2 开始的连续素数的时候 n 值最小。
2. 如果数值小的素数的幂次小于数值大的素数的幂，那么如果把这两个素数交换位置（幂次不变），那么所得的 n 因子数量不变，但是 n 的值变小。

GCD（最大公约数）

```
typedef long long ll;
ll gcd(ll a, ll b) {
  if (b == 0) return a;
  return gcd(b, a % b);
}
```

可以使用内置于algorithm头文件中的__gcd(a,b)函数

LCM（最小公倍数）

```
typedef long long ll;
ll lcm(ll a, ll b) {
  if (b == 0) return 0;
  return a/gcd(b, a % b)*b;
}
```

线性筛

1.素数筛

```
typedef long long ll;
ll v[100005];
void primes(ll n) {
    memset(v,0,sizeof(v));
    for(int i = 2 ; i <= n ; i++) {
        if(v[i]) continue;
        for(int j = i ; j <= n/i ; j++) {
            v[i*j] = 1;
        }
    }
}
```

2.线性筛

```
bool numlist[100000001];
int prime[20000001], cnt;
void work(int n){
    for(int i=2; i<=n; i++){
        if(numlist[i] == false)
            prime[++cnt] = i ;
        for(int j=1; j <= cnt && i * prime[j] <= n; j++){
            numlist[i*prime[j]] = true ;
            if(i%prime[j]==0) // i中也含有Prime[j]这个因子
                break;
        }
    }
    return;
}
```

快速幂

```
typedef long long ll;
ll pow(ll a,ll n,ll p) {
    ll ans = 1;
    while(n) {
        if(n & 1 ) ans = (ans * a) % p;
        a = a * a % p;
        n >>= 1;
    }
    return ans ;
}
```

矩阵快速幂

```
const int N=10;
int tmp[N][N];
void multi(int a[][N],int b[][N],int n) {
    memset(tmp,0,sizeof tmp);
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            for(int k=0;k<n;k++)
                tmp[i][j]+=a[i][k]*b[k][j];
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
```

```
            a[i][j]=tmp[i][j];
}
int res[N][N];
void Pow(int a[][N],int n) {
    memset(res,0,sizeof res);//n是幂，N是矩阵大小
    for(int i=0;i<N;i++) res[i][i]=1;
    while(n)
    {
        if(n&1)
            multi(res,a,N);//res=res*a;复制直接在multi里面实现了;
        multi(a,a,N);//a=a*a
        n>>=1;
    }
}
```

组合数处理方法

lucas：$C_n^m \equiv C_{n/p}^{m/p} * C_{n \bmod p}^{m \bmod p} \pmod p$

定义：用于求解 $\binom{n}{m} \pmod p$ 其中，p 为素数且比较小

结论：

所求数同余于它在 p 进制分解下组合数的乘积

对于质数 p，所有 $j \in [1,p)$ 均有

$\binom{p}{j} = \frac{p!}{j!(p-j)!} = \frac{(p-1)!}{(j-1)!(p-j)!} * \frac{p}{j} = \binom{p-1}{j-1} * \frac{p}{j} \equiv 0 \pmod p$

$\therefore (1+j)^p = 1 + \sum_{i=1}^{p-1} \binom{p}{i} * j^i + j^p \equiv 1 + j^p \pmod p$

令 $n = sp+q, m = lp+r$ （$0 \le$ p, $r < q$）

则

$(1+x)^{sp+q} \equiv (1+x)^{sp}(1+x)^q \equiv (1+x^p)^s(1+x)^q \equiv (\sum_{i=0}^{s} \binom{s}{i} * x^{ip}) * (\sum_{j=0}^{q} \binom{q}{j} * x^j) \pmod p$

观察 x^m 项

$\binom{n}{m} * x^m \equiv \binom{s}{l} * x^{lp} * \binom{q}{r} * x^r \pmod p$

$\binom{n}{m} * x^m \equiv (\binom{s}{l} * \binom{q}{r}) * x^m \pmod p$

又因为 p 为素数,所以两边同时约去 x^m

则 $\binom{n}{m} \equiv \binom{s}{l} * \binom{q}{r} \equiv \binom{n/p}{m/p} * \binom{n \bmod p}{m \bmod p} \pmod p$

即 $C_n^m \equiv C_{n/p}^{m/p} * C_{n \bmod p}^{m \bmod p} \pmod p$

证毕!

```
#define ll long long
ll p;
inline ll qpow(ll a,ll b) {
    if(b==1) return a;
    ll t=qpow(a,b/2);
    t=t*t%p;
    if(b&1) t=t*a%p;
    return t;
}
inline ll C(ll n,ll m){
    if(n<m) return 0;
    if(m>n-m) m=n-m;
    ll a=1,b=1;
    for(int i=0;i<m;i++){
        a=(a*(n-i))%p;
        b=(b*(i+1))%p;
```

```
    }
    return a*qpow(b,p-2)%p;
}
inline ll Lucas(ll n,ll m){
    if(m==0) return 1;
    return Lucas(n/p,m/p)*C(n%p,m%p)%p;
}
```

抽屉原理（鸽巢原理）

有 $n+1$ 个物品，想要放到 n 个抽屉里，那么必然会有至少一个抽屉里有两个（或以上）的物品。

这个定理看起来比较显然，证明方法考虑反证法：假如所有抽屉都至多放了一个物品，那么 n 个抽屉至多只能放 n 个物品，矛盾。

欧拉函数

定义

欧拉函数，即 $\varphi(n)$ ，表示的是小于等于n 和 n 互质的数的个数。

互质：公约数只有1的两个整数，叫做互质数。

```
int euler_phi(int n) {
  int m = int(sqrt(n + 0.5));
  int ans = n;
  for (int i = 2; i <= m; i++)
    if (n % i == 0) {
      ans = ans / i * (i - 1);
      while (n % i == 0) n /= i;
    }
  if (n > 1) ans = ans / n * (n - 1);
  return ans;
}
```

费马小定理

若 p 是质数，则对于任何整数 a ,有 $a^p \equiv a(mod\, p)$

若 p 为质数 ,$gcd(a,p)=1$,则 $a^{p-1} \equiv 1(mod\, p)$

证明

设一个质数为 p ，我们取一个不为 p 倍数的数 a 。

构造一个序列： $A=\{1,2,3\ldots,p-1\}$ ，这个序列有着这样一个性质：

$$\prod_{i=1}^{n} A_i \equiv \prod_{i=1}^{n}(A_i \times a) \pmod{p}$$

证明：

$$\because (A_i,p)=1,(A_i\times a,p)=1$$

又因为每一个 $A_i \times a \pmod{p}$ 都是独一无二的，且 $A_i \times a \pmod{p} < p$

得证（每一个 $A_i \times a$ 都对应了一个 A_i ）

设 $f=(p-1)!$，则 $f\equiv a\times A_1 \times a \times A_2 \times a \times A_3 \cdots \times A_{p-1} \pmod{p}$

$$a^{p-1}\times f \equiv f \pmod{p}$$
$$a^{p-1}\equiv 1 \pmod{p}$$

欧拉定理

若$gcd(a,m)=1$ ，则 $a^{\varphi(m)}\equiv 1\ (mod\ \ \mathrm{m})$

证明 ¶

实际上这个证明过程跟上文费马小定理的证明过程是非常相似的： **构造一个与 m 互质的数列** ，再进行操作。

设 $r_1,r_2,\cdots,r_{\varphi(m)}$ 为模 m 意义下的一个简化剩余系，则 $ar_1,ar_2,\cdots,ar_{\varphi(m)}$ 也为模 m 意义下的一个简化剩余系。所以 $r_1r_2\cdots r_{\varphi(m)}\equiv ar_1\cdot ar_2\cdots ar_{\varphi(m)}\equiv a^{\varphi(m)}r_1r_2\cdots r_{\varphi(m)} \pmod{m}$ ，可约去 $r_1r_2\cdots r_{\varphi(m)}$ ，即得 $a^{\varphi(m)}\equiv 1 \pmod{m}$ 。

当 m 为素数时，由于 $\varphi(m)=m-1$ ，代入欧拉定理可立即得到费马小定理。

扩展欧拉定理(欧拉降幂)

$$a^b\equiv\begin{cases}a^{b \bmod \varphi(p)}, & \gcd(a,\,p)=1\\ a^b, & \gcd(a,\,p)\neq 1,\,b<\varphi(p)\\ a^{b \bmod \varphi(p)+\varphi(p)}, & \gcd(a,\,p)\neq 1,\,b\geq\varphi(p)\end{cases} \pmod{p}$$

1. 在 a 的 0 次，1 次，。。。，b 次幂模 m 的序列中，前 r 个数（ a^0 到 a^{r-1} ）互不相同，从第 r 个数开始，每 s 个数就循环一次。

   证明：由鸽巢定理易证。

   我们把 r 称为 a 幂次模 m 的循环起始点，s 称为循环长度。（注意：r 可以为 0 ）

   用公式表述为：$a^r \equiv a^{r+s} \pmod m$

2. a 为素数的情况

   令 $m = p^r m'$ ，则 $\gcd(p, m') = 1$ ，所以 $p^{\varphi(m')} \equiv 1 \pmod{m'}$

   又由于 $\gcd(p^r, m') = 1$ ，所以 $\varphi(m') \mid \varphi(m)$ ，所以 $p^{\varphi(m)} \equiv 1 \pmod{m'}$ ，即 $p^\varphi(m) = km' + 1$ ，两边同时乘以 p^r ，得 $p^{r+\varphi(m)} = km + p^r$ （因为 $m = p^r m'$ ）

   所以 $p^r \equiv p^{r+s} \pmod m$ ，这里 $s = \varphi(m)$

3. 推论：$p^b \equiv p^{r+(b-r) \bmod \varphi(m)} \pmod m$

4. 又由于 $m = p^r m'$ ，所以 $\varphi(m) \ge \varphi(p^r) = p^{r-1}(p-1) \ge r$

   所以 $p^r \equiv p^{r+\varphi(m)} \equiv p^{r \bmod \varphi(m)+\varphi(m)} \pmod m$

   所以
   $p^b \equiv p^{r+(b-r) \bmod \varphi(m)} \equiv p^{r \bmod \varphi(m)+\varphi(m)+(b-r) \bmod \varphi(m)} \equiv p^{\varphi(m)+b \bmod \varphi(m)} \pmod m$

   即 $p^b \equiv p^{b \bmod \varphi(m)+\varphi(m)} \pmod m$

5. a 为素数的幂的情况

   是否依然有 $a^{r'} \equiv a^{r'+s'} \pmod m$ ？（其中 $s' = \varphi(m), a = p^k$ ）

   答案是肯定的，由 2 知 $p^s \equiv 1 \pmod m)'$ ，所以 $p^{s \times \frac{k}{\gcd(s,k)}} \equiv 1 \pmod{m'}$ ，所以当 $s' = \frac{s}{\gcd(s,k)}$ 时才能有 $p^{s'k} \equiv 1 \pmod{m'}$ ，此时 $s' \mid s \mid \varphi(m)$ ，且 $r' = \lceil \frac{r}{k} \rceil \le r \le \varphi(m)$ ，由 r', s' 与 $\varphi(m)$ 的关系，依然可以得到 $a^b \equiv a^{b \bmod \varphi(m)+\varphi(m)} \pmod m$

6. a 为合数的情况

   只证 a 拆成两个素数的幂的情况，大于两个的用数学归纳法可证。

   设 $a = a_1 a_2, a_i = p_i^{k_i}$ ，a_i 的循环长度为 s_i ；

   则 $s \mid lcm(s_1, s_2)$ ，由于 $s_1 \mid \varphi(m), s_2 \mid \varphi(m)$ ，那么 $lcm(s_1, s_2) \mid \varphi(m)$ ，所以 $s \mid \varphi(m)$ ，$r = \max(\lceil \frac{r_i}{k_i} \rceil) \le \max(r_i) \le \varphi(m)$ ；

   由 r, s 与 $\varphi(m)$ 的关系，依然可以得到 $a^b \equiv a^{b \bmod \varphi(m)+\varphi(m)} \pmod m$ ；

   证毕。

```
// 这套模板不好，可以自己写哦 ！！！
ll qpow(ll a, ll n, ll mod) {
    ll ret = 1;
    while(n) {
        if(n%2) ret = (ret%mod)*(a%mod)%mod;
        a = (a%mod)*(a%mod)%mod;
        n/=2;
    }
    return ret;
}
ll phi(ll x) {
    ll ret = x;
    for(ll i = 2;i*i <= x;i++) {
        if(x%i == 0) {
            ret = ret*(i-1)/i;
            while(x%i == 0)x/=i;
```

```
        }
    }
    if(x > 1)ret = ret*(x-1)/x;
    return ret;
}
```

乘法逆元

逆元定义：如果一个线性同余方程$ax \equiv 1(mod\, b)$ ，则 x 为 $a\, mod\, b$ 的逆元，记作 a^{-1}
快速幂法：

```
typedef long long ll;
ll pow(ll a,ll n,ll p) {
    ll ans = 1;
    a = (a % p + p) % p;
    while(n) {
        if(n & 1 ) ans = (ans * a) % p;
        a = a * a % p;
        n >>= 1;
    }
    return ans ;
}
```

求逆元方法不止这一种，有兴趣可以自行学习；

扩展欧几里得

对于任意整数 a, b，存在一对整数x, y，满足 $ax + by = gcd(a, b)$ 。

```
int exgcd(int a,int b,int &x,int &y) {
    if(b == 0 ) {
        x = 1;
        y = 0;
        return a;
    }
    int d = exgcd(b, a%b , x, y);
    int z = x ;
    x = y;
    y = z - y*(a/b);
    return d;
}
```

线性同余方程

形如 $ax \equiv c \pmod{b}$ 的方程被称为 线性同余方程。

求解方法

根据以下两个定理，我们可以求出同余方程 $ax \equiv c \pmod b$ 的解。

定理 1 ：方程 $ax+by=c$ 与方程 $ax \equiv c \pmod b$ 是等价的，有整数解的充要条件为 $\gcd(a,b) \mid c$ 。

根据定理 1，方程 $ax+by=c$ ，我们可以先用扩展欧几里得算法求出一组 x_0, y_0 ，也就是 $ax_0+by_0=\gcd(a,b)$ ，然后两边同时除以 $\gcd(a,b)$ ，再乘 c 。然后就得到了方程 $acx_0/\gcd(a,b)+bcy_0/\gcd(a,b)=c$ ，然后我们就找到了方程的一个解。

定理 2 ：若 $\gcd(a,b)=1$ ，且 x_0, y_0 为方程 $ax+by=c$ 的一组解，则该方程的任意解可表示为： $x=x_0+bt, y=y_0-at$ ，且对任意整数 t 都成立。

根据定理 2，可以求出方程的所有解。但在实际问题中，我们往往被要求求出一个最小整数解，也就是一个特解 $x, t=b/\gcd(a,b), x=(x \bmod t+t) \bmod t$ 。

中国剩余定理（CRT)

在《孙子算经》中有这样一个问题："今有物不知其数，三三数之剩二（除以3余2），五五数之剩三（除以5余3），七七数之剩二（除以7余2），问物几何？"这个问题称为"孙子问题"，该问题的一般解法国际上称为"中国剩余定理"。具体解法分三步：
找出三个数：从3和5的公倍数中找出被7除余1的最小数15，从3和7的公倍数中找出被5除余1 的最小数21，最后从5和7的公倍数中找出除3余1的最小数70。
用15乘以2（2为最终结果除以7的余数），用21乘以3（3为最终结果除以5的余数），同理，用70乘以2（2为最终结果除以3的余数），然后把三个乘积相加$15*2+21*3+70*2$得到和233。
用233除以3，5，7三个数的最小公倍数105，得到余数23，即$233\%105=23$。这个余数23就是符合条件的最小数。

就这么简单。我们在感叹神奇的同时不禁想知道古人是如何想到这个方法的，有什么基本的数学依据吗？

我们将"孙子问题"拆分成几个简单的小问题，从零开始，试图揣测古人是如何推导出这个解法的。

首先，我们假设n_1是满足除以3余2的一个数，比如2，5，8等等，也就是满足$3*k+2$（$k \geq 0$）的一个任意数。同样，我们假设n_2是满足除以5余3的一个数，n_3是满足除以7余2的一个数。

有了前面的假设，我们先从n_1这个角度出发，已知n_1满足除以3余2，能不能使得n_1+n_2的和仍然满足除以3余2？进而使得$n_1+n_2+n_3$的和仍然满足除以3余2？

这就牵涉到一个最基本数学定理，如果有$a\%b=c$，则有$(a+k*b)\%b=c$(k为非零整数)，换句话说，如果一个除法运算的余数为c，那么被除数与k倍的除数相加（或相减）的和（差）再与除数相除，余数不变。这个是很好证明的。　　以此定理为依据，如果n_2是3的倍数，n_1+n_2就依然满足除以3余2。同理，如果n_3也是3的倍数，那么$n_1+n_2+n_3$的和就满足除以3余2。这是从n_1的角度考虑的，再从n_2，n_3的角度出发，我们可推导出以下三点： 为使$n_1+n_2+n_3$的和满足除以3余2，n_2和n_3必须是3的倍数。 为使$n_1+n_2+n_3$的和满足除以5余3，n_1和n_3必须是5的倍数。 为使$n_1+n_2+n_3$的和满足除以7余2，n_1和n_2必须是7的倍数。　　因此，为使$n_1+n_2+n_3$的和作为"孙子问题"的一个最终解，需满足： n_1除以3余2，且是5和7的公倍数。 n_2除以5余3，且是3和7的公倍数。 n_3除以7余2，且是3和5的公倍数。　　所以，孙子问题解法的本质是从5和7的公倍数中找一个除以3余2的数 n_1 ，从3和7的公倍数中找一个除以5余3的数n_2，从3和5的公倍数中找一个除以7余2的数n_3，再将三个数相加得到解。在求n_1，n_2，n_3时又用了一个小技巧，以n_1为例，并非从5和7的公倍数中直接找一个除以3余2的数，而是先找一个除以3余1的数，再乘以2。也就是先求出5和7的公倍数模3下的逆元，再用逆元去乘余数 这里又有一个数学公式，如果$a\%b=c$，那么 $(a*k)\%b=a\%b+a\%b;+a\%b=c+c+...+c=k*c(k>0)$,也就是说，如果一个除 法的余数为$c$，那么被除数的$k$倍与除数相除的余数为$k*c$。展开式中已证明。　　最后，我们还要清楚一点，$n_1+n_2+n_3$只是问题的一个解，并不是最小的解。如何得到最小解？我们只需要从中最大限度的减掉

掉3，5，7的公倍数105即可。道理就是前面讲过的定理“如果$a\%b=c$，则有$(a-k*b)\%b=c$”。所以（$n_1+n_2+n_3$）$\%105$就是最终的最小解。　　这样一来就得到了中国剩余定理的公式：

设正整数

$$m_1, m_2, \ldots, m_k$$

两两互素，则同余方程组

$$\begin{aligned} x &\equiv a_1 (mod\ m_1) \\ x &\equiv a_2 (mod\ m_2) \\ x &\equiv a_3 (mod\ m_3) \\ &\cdot \\ &\cdot \\ &\cdot \\ x &\equiv a_k (mod\ m_k) \end{aligned}$$

有整数解。并且在模

$$M = m_1 \cdot m_2 \cdot \ldots \cdot m_k$$

下的解是唯一的，解为

$$x \equiv (a_1 M_1 M_1^{-1} + a_2 M_2 M_2^{-1} + \ldots + a_k M_k M_k^{-1}) mod\ M$$

其中$M_i = M/m_i$，而M_i^{-1}为M_i模m_i的逆元。

```
//扩展欧几里得算法
int exgcd(int a,int b,int &x,int &y) {
    if(b == 0 ) {
        x = 1;
        y = 0;
        return a;
    }
    int d = exgcd(b, a%b , x, y);
    int z = x ;
    x = y;
    y = z - y*(a/b);
    return d;
}


//中国剩余定理
int CRT(int A[],int B[],int n){ //A[]存放余数  B[]存放两两互质的数
    int m = 1,ans = 0;
    for(int i = 0;i < n;++i)
        m = m * B[i];
    for(int i = 0;i < n;++i){
        int x,y;
        int Mi = m / B[i];
        exgcd(Mi,B[i],x,y);
        ans = (ans + Mi * x * A[i]) % m;
    }
    if(ans <= 0) ans += m;
```

```
    return ans;
}
```

扩展中国剩余定理(EXCRT)

求解模数不互质情况下的线性方程组：

普通的中国剩余定理要求所有的

$$m_i$$

互素，那么如果不互素呢，怎么求解同余方程组？

这种情况就采用两两合并的思想，假设要合并如下两个方程：

$$x = a_1 + m_1 x_1$$
$$x = a_2 + m_2 x_2$$

那么得到：

$$a_1 + m_1 x_1 = a_2 + m_2 x_2 \quad \Rightarrow \quad m_1 x_1 + m_2 x_2 = a_2 - a_1$$

我们需要求出一个最小的x使它满足：

那么x_1和x_2就要尽可能的小，于是我们用扩展欧几里得算法求出x_1的最小正整数解，将它代回$a_1 + m_1 x_1$，得到 x 的一个特解 x'，当然也是最小正整数解。

所以x的通解一定是 x' 加上 $lcm(m1, m2) * k$ ，这样才能保证 x 模m和m_2的余数是a_1和 a_2 。由此，我们把这个 x' 当做新的方程的余数，把$lcm(m1, m2)$当做新的方程的模数。（这一段是关键）

合并完成：

$$x \equiv x' \ (mod \ \ lcm(m1, m2))$$

```
ll exgcd(ll a,ll b,ll &x,ll &y) {
    if(b == 0 ) {
        x = 1;
        y = 0;
        return a;
    }
    ll d = exgcd(b, a%b , x, y);
    ll z = x ;
    x = y;
    y = z - y*(a/b);
    return d;
}
//逐一合并大法
ll CRT(int w[],int b[],int k)//w为除数，b为余数，k为有多少式子
{
    ll Wi=w[0],ret=b[0];
    for(int i=1;i<k;++i)
    {
        ll wi=w[i];
        ll bi=b[i];
        ll x,y;
        ll gcd=exgcd(Wi,wi,x,y);
        ll c=bi-ret;
        if(c%gcd)//表示没有结果
            return -1;
```

```
        ll W=wi/gcd;
        ret+=wi*(((c/gcd*x)%W+W)%W);
        wi*=W;
    }

    if(!ret)//表示余数全部为零
    {
        ret=1;
        for(int i=0;i<k;++i)
            ret=ret*w[i]/__gcd(ret,(ll)w[i]);//使用库函数求最小公倍数
    }
    return ret;
}
```

自适应辛普森法（有点难 自学）

HDU1724