数论--质因数分解

文章目录

1、用试除法分解质因子

2、用Pollard\_rho启发式方法分解质因子

   任何一个正整数nnn都可以唯一分解为有限个素数的乘积：n=pc1pc2...pcm

都是正整数，pi都是素数且从小到大。

  质因数分解有重要工程意义。在密码学中，需要对高达百位以上的十进制数分解质因子，因此发明了很多高效率的方法1。不过，大数的质因子分解是个难题，比寻找大素数要难得多，密码算法RSA就利用了大数难以分解的原理。

1、用试除法分解质因子

   分解质因子也可以用前面提到的试除法。求n的质因子：

   （1）第一步，求最小质因子p1。逐个检查从2到的所有素数，如果它能整除n，就是最小质因子。然后连续用p1除n，目的是去掉n中的p1，得到n1。

   （2）第二步，再找n1的最小质因子。逐个检查从p1到的所有素数。从p1开始试除，是因为n1没有比p1小的素因子，而且n1的因子也是n的因子。

   （3）继续以上步骤，直到找到所有质因子。

   最后，经过去除因子的操作后，如果剩下一个大于1的数，那么它也是一个素数，是n的最大质因子。这种情况可以用一个例子说明。大于sqrt(n)的素数也可能是n的质因子，例如6119 = 29\*211，找到29后，因为29 ≥  ，说明211是素数，也是质因子。

   试除法的复杂度是，效率很低。不过，在算法竞赛中，数据规模不大，所以一般就用试除法。

   下面是试除法的代码2。因为试除法的效率不高，所以n用int型，没有用long long。

int p[20]; //p[]记录因子，p[1]是最小因子。一个int数的质因子最多有10几个

int c[40]; //c[i]记录第i个因子的个数。一个因子的个数最多有30几个

void factorization(int n){

int m = 0;

for(int i = 2; i\*i <= n; i++)

if(n%i == 0){

p[++m] = i, c[m] = 0;

while(n%i == 0) //把n中重复的因子去掉

n/=i, c[m]++;

}

if(n>1) //没有被除尽，是素数

p[++m] = n, c[m] = 1;

}

2、用Pollard\_rho启发式方法分解质因子

  试除法的复杂度是，也就是说，对到B的整数进行试除，可以完全获得到B2的任意数的因子分解；用本节的pollard\_rho算法，用同样的工作量，可以对到B4的数进行因子分解。需要指出的是，pollard\_rho算法也仍然是一种低效的方法，比试除法好一点点，只能在算法竞赛的小规模数据中用用。

  思考一个问题：如何快速找到一个大数的因子？不能像试除法那样从小到大一个个检查，太慢了。可以挑一些数来“试”，运气好说不定就碰到一个。这就是pollard\_rho算法的思路，它使用了一个“随机”的方法来找。算法的主要内容只有2个：

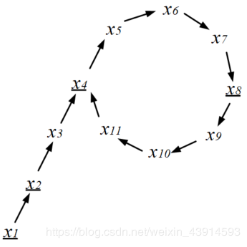
  （1）“随机”函数。实际上不是随机，而是一个启发函数：，其中x的初值x1和c是随机数。计算的结果是生成了一个x序列，这个序列的前一部分x1，x2，...，xj−1 不重复，后面的xj，xj+1，...，xi会重复并形成回路。rho指希腊字母"ρ"，不重复的序列是ρ的“尾巴”，重复的回路是ρ的“身体”。

图1 rho的“尾巴”和“身体”

  （2）计算n的一个因子。计算d=gcd(y−xi,n)，其中y是第2k个x，即第1、2、4、8、…个，见上图中划线的x。如果d ≠ 1且d ≠ n，d就是n的一个因子，原因很简单，gcd是求最大公约数，所以d肯定是n的因子。

  从上面的描述可以看出，pollard\_rho算法极为简单，读者可能怀疑它是否真的有效。确实，在一次x序列中，很可能计算不出因子，需要多次“随机”的x序列才能算出一个因子。令人惊讶的是，这个算法的效果还不错，它可以用次计算找到n的一个小因子p。

  pollard\_rho的编码非常简单，见下面代码中的pollard\_rho()函数。由于执行一次pollard\_rho()只返回一个因子，要得到所有的因子，需要再写一个findfac()函数多次调用pollard\_rho()，递归求得所有素因子。

poj 1811 部分代码（pollard\_rho）

//poj 1811题：输入一个整数n,2<=N<2^54，判断它是否为素数，如果不是，输出最小素因子。

typedef long long ll;

ll Gcd (ll a,ll b){ return b? Gcd(b, a%b):a;}

ll pollard\_rho (ll n){ //返回一个因子，不一定是素因子

ll i=1, k=2;

ll c = rand()%(n-1)+1;

ll x = rand()%n;

ll y = x;

while (true){

i++;

x = (mult\_mod(x,x,n)+c) % n; //mult\_mod(x,x,n)功能是(x\*x) mod n

ll d = Gcd(y>x?y-x:x-y, n); //重要：保证gcd的数大于等于0

if (d!=1 && d!=n) return d; //算出一个因子

if (y==x) return n; //已经出现过，直接返回

if (i==k) { y=x; k=k<<1;}

}

}

void findfac (ll n){ //找所有的素因子

if (miller\_rabin(n)) { //用miller\_rabin判断是否为素数

factor[tol++] = n; //存素因子

return;

}

ll p = n;

while (p>=n)

p = pollard\_rho(p); //找到一个因子

findfac(p); //继续寻找更小的因子

findfac(n/p);

}