**第八章 离散对数问题两种算法的测**

由于源代码中有互素判断，扩展欧几里得算法求逆元和多余随机策略代码，显得有点多和乱，

Shanks法和pollards Rho部分代码如下：

**Shanks算法**

**public** **static** **long** ShanksBSGS(**long** a,**long** b,**long** n){ //Shanks法解离散对数，参数a为生成元，b为群元素，n为群的模数

**long** starTime=System.*currentTimeMillis*();

**long** N=n-1;

**long** m=(**long**)Math.*ceil*(Math.*sqrt*(N)); //m为为群的阶开方向上取整

BigInteger []XB=**new** BigInteger[(**int**)m];

BigInteger aa=BigInteger.*valueOf*(a); //制作a的幂运算的表格，数组下标为指数值

**for**(**int** i=0;i<m;i++)

{

XB[i]=(aa.pow(i)).mod(BigInteger.*valueOf*(n));

}

aa=BigInteger.*valueOf*(a);

aa=(aa.pow((**int**)m)).mod(BigInteger.*valueOf*(n));

aa=*EeaBig*(aa.toString(),(BigInteger.*valueOf*(n)).toString()); //自己写的欧几里得扩展求逆元算法，可以用库函数替换

//主要为求a的逆元，此处查找的一种优化的方法

//原本X=Xg\*m+Xb,0<=Xg,Xb<m

//换做X=Xg\*m-Xb,而0<=Xg,Xb<=m,X依然可以取到0到m^2-1,增加了一次Xb,却省去求逆元的步骤

BigInteger xx;

**long** Xg=0,Xb=0;

**boolean** f=**false**;

**for**(**int** i=0;i<m;i++) //穷举比较

{

xx=(aa.pow(i)).mod(BigInteger.*valueOf*(n));

**for**(**int** j=0;j<m;j++)

{

**if**(((xx.multiply(BigInteger.*valueOf*(b))).mod(BigInteger.*valueOf*(n)).compareTo(XB[j]))==0) //查找比较

{

Xb=(**long**)j;

Xg=(**long**)i;

f=**true**;

**break** ;

}

} //由于单个数据较小（太懒），检索算法没有做优化只用了原始的顺序检索

**if**(f)

**break**;

}

**long** endTime=System.*currentTimeMillis*();

**double** Time=(endTime-starTime) /1000.0;

System.***out***.println("ShanksBSGS算法执行的时间为 "+Time+"秒");

System.***out***.println();

**return** Xg\*m+Xb;

}

**Pollards Rho算法**

**public** **static** **double** PollardsRho(**long** g,**long** B,**long** N){ //pollard Rho算法解离散对数，参数a为生成元，b为对数，n为群的模数

**long** starTime=System.*currentTimeMillis*();

**long** X1=1,X2=0;

**int** a1=0,a2=0;

**int** b1=0,b2=0;

**long** T=0;

**while**(X1!=X2)

{

a1=(**new** java.util.Random().nextInt()%1000+1000)%1000;

a2=(**new** java.util.Random().nextInt()%1000+1000)%1000;

b1=(**new** java.util.Random().nextInt()%1000+1000)%1000;

b2=(**new** java.util.Random().nextInt()%1000+1000)%1000; //生成四个随机数

System.***out***.println("a1 "+a1+" b1 "+b1+" a2 "+a2+" b2 "+b2);

T=(b1-b2+N-1)%(N-1); //分母模N-1保证是正数

**if**(b1==b2)

**continue**; //分母为零，重新筛选

**if**(!*isrelativelyprime*(BigInteger.*valueOf*(T).toString(), BigInteger.*valueOf*(N-1).toString()))

**continue**; //分母模N-1如果没有逆元，重新筛选

X1=(BigInteger.*valueOf*(g)).pow(a1).multiply(BigInteger.*valueOf*(B).

pow(b1)).mod(BigInteger.*valueOf*(N)).longValue(); //a^i1\*b^j1 mod N

X2=(BigInteger.*valueOf*(g)).pow(a2).multiply(BigInteger.*valueOf*(B).

pow(b2)).mod(BigInteger.*valueOf*(N)).longValue(); //a^i2\*b^j2 mod N

}

BigInteger X\_a=BigInteger.*valueOf*((a2-a1+N-1)%(N-1));

BigInteger x\_b=BigInteger.*valueOf*((b1-b2+N-1)%(N-1)); //确保a1-a2,b2-b1 mod N-1 为正数

x\_b=x\_b.modInverse(BigInteger.*valueOf*(N-1)); //求出b1-b2 mod N-1 的逆元

System.***out***.println("逆元 "+x\_b.toString());

X\_a=X\_a.multiply(x\_b).mod(BigInteger.*valueOf*(N-1)); //逆元乘上a2-a1 mod N-1 得到结果为X\_a

**long** endTime=System.*currentTimeMillis*();

**double** Time=(endTime-starTime) /1000.0;

System.***out***.println("pollard Rho算法执行的时间为 "+Time+"秒");

System.***out***.println();

**return** X\_a.longValue(); //返回结果

}

}

结果截图：





