**云南大学数学与与统计学院**

**上机实践报告**

|  |  |  |
| --- | --- | --- |
| **课程名称**：近代密码学实验 | **年级**：2013 | **上机实践成绩**： |
| **指导教师**：陆正福 | **姓名**：金洋 |  |
| **上机实践名称**：基础算法实验 | **学号**：20131910023 | **上机实践日期**：9.7 |
| **上机实践编号**：No.02 | **组号**： | **上机实践时间**：16:37 |

**一、实验目的**

熟悉密码学编程平台和编程资源

1. **实验内容**
2. 编程实现求两个整数的最大公约数的欧几里得算法
3. 编程实现求两个整数的最大公约数和线性组合的扩展欧几里得算法
4. 编程实现整数的模逆算法
5. 编程实现整数的快速模幂算法

**三、实验环境**

个人计算机，Java 8平台

对于非信息与计算科学专业的学生，可以选择任意编程平台

**四、实验记录与实验结果分析**

（注意记录实验中遇到的问题。实验报告的评分依据之一是实验记录的细致程度、实验过程的真实性、实验结果的解释和分析。**如果涉及实验结果截屏，应选择白底黑字。**）

1. 编程实现求两个整数a,b的最大公约数的欧几里得算法。

算法描述如下：

1. 令；
2. ；
3. 除以得到商和余数，即

， 

1. 若，则，算法结束；
2. 若，，返回（3）；

主要代码如下：

**public** **int** euclidean(**int** a,**int** b) {

**int**[] r,q;

r=**new** **int**[(**int**) (2\*Math.*log*(b)/Math.*log*(2)+3)];

q=**new** **int**[(**int**) (2\*Math.*log*(b)/Math.*log*(2)+3)];

r[0]=a;

r[1]=b;

**int** i=0;

**do** {

i++;

q[i]=r[i-1]/r[i];

r[i+1]=r[i-1]%r[i];

}**while** (r[i+1]!=0);

**return** r[i];

}

大整数下的欧几里得算法主要代码：

**public** BigInteger euclidean(BigInteger a,BigInteger b) {

**if** (b.compareTo(zero)==0) **return** a;

**return** euclidean(b,a.mod(b));

}

1. 编程实现求两个整数的最大公约数和线性组合的扩展欧几里得算法

基本算法：对于不完全为 0 的非负整数 a,b, gcd(a,b)表示 a, b的最大公约数，必然存在整数对 u, v ，使得gcd(a,b)=au+bv.

证明：设 a>b。

　1）显然当 b=0，gcd（a，b）=a。此时 u=1，v=0；

　2）ab≠0 时

　　设 ;



　　根据朴素的欧几里德原理有 gcd(a,b)=gcd(b,a% b);



　　即:

　　得：; ;

     这样我们就得到了求解  的方法：的值基于.

上面的思想是以递归定义的，因为 gcd 不断的递归求解一定会有个时候 b=0，所以递归可以结束。

主要代码如下：

**public** **int** extendedEuclidean(**int** a,**int** b) {

**if** (b==0) {

u=1;

v=0;

**return** a;

}

**int** r= extendedEuclidean(b,a%b);

**int** t=u;

u=v;

v=t-a/b\*v;

**return** r;

}

扩展欧几里得算法主要代码（大整数版本）：

**public** BigInteger extendedEuclidean(BigInteger a,BigInteger b) {

**if** (b.compareTo(zero)==0) {

u=**new** BigInteger("1");

v=**new** BigInteger("0");

**return** a;

}

BigInteger r= extendedEuclidean(b,a.mod(b));

BigInteger t=u;

u=v;

v=t.subtract(a.divide(b).multiply(v));

**return** r;

}

1. 编程实现整数的模逆算法

法一：利用拓展欧几里得算法，当a,p 互素时，有， 则，所以，主要代码如下：

System.***out***.print("整数的模逆算法,请输入代求元a与素数p:");

a=input.nextInt();

b=input.nextInt();

FA.extendedEuclidean(a, b);

System.***out***.println("a^(-1)="+Math.*abs*(FA.getU()) % b);

大整数版本：

a=input.nextBigInteger();

BigInteger p=input.nextBigInteger();

FA.extendedEuclidean(a, p);

System.***out***.println("a^(-1)="+FA.getU().mod(p).abs());

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

法二：当mod的是一个素数时，利用费马小定理得到，可以借助快速模幂算法实现。

1. 编程实现整数的快速模幂算法

根据书本P25的算法”The Fast Powering Algorithm”得到如下快速模幂程序：

**public** **long** fastPowering(**int** g,**int** A,**int** N) {

**int** r=(**int**) (Math.*log*(A)/Math.*log*(2));

**long**[] a=**new** **long**[r+1];

**int**[] b=**new** **int**[r+1];

**long**[] tp=**new** **long**[r+1];

**int** i;

/\* 2^i \*/

tp[0]=1;

**for** (i=1;i<=r;i++) tp[i]=2\*tp[i-1];

i=r;

**int** G=A;

**while** (G>0) {

//System.out.println("Jin"+i+" "+G+" "+tp[i]);

**if** (G>=tp[i]) {

b[i]=1;

G-=tp[i];

}

**else** b[i]=0;

i--;

}

a[0]=g % N;

**for** (i=1;i<=r;i++) a[i]=a[i-1]\*a[i-1] % N;

**long** ans=1;

**for** (i=0;i<=r;i++)

**if** (b[i]==1) ans=ans \* a[i] % N;

**return** ans;

}

另一种节省空间的快速模幂算法（书本P52）：

Input Positive integer N, g, A.

① Set a=g, and b=1.

② Loop While A>0

③ If A=1 mod 2， set b=b\*a mod N;

④ Set a=a\*a mod N, A=A/2;

⑤ If A>0, goto ②;

⑥ Return b(即为 )

解释：快速模幂算法（书本P25）中将指数A分解为



其中，

即为A的二进制形式，以上算法中③④⑤中在做的即为从低位开始求A的二进制形式。

节省空间的快速模幂算法主要代码（大整数版本）：

**public** BigInteger fastPowering(BigInteger g,BigInteger A,BigInteger N) {

BigInteger a=g;

BigInteger b=**new** BigInteger("1");

BigInteger one=**new** BigInteger("1");

BigInteger two=**new** BigInteger("2");

**while** (A.compareTo(zero)!=0) {

**if** (A.mod(two).compareTo(one)==0) b=b.multiply(a).mod(N);

a=a.multiply(a);

A=A.divide(two);

}

**return** b;

}

1. 完整程序

**FundamentalAlgorithms.java**

**package** MC02;

**public** **class** FundamentalAlgorithms {

**protected** **int** u,v;//拓展欧几里得算法中的系数

**public** FundamentalAlgorithms() {

}

**public** **int** getU() {

**return** u;

}

**public** **int** getV() {

**return** v;

}

**public** **int** euclidean(**int** a,**int** b) {

**int**[] r,q;

r=**new** **int**[(**int**) (2\*Math.*log*(b)/Math.*log*(2)+3)];

q=**new** **int**[(**int**) (2\*Math.*log*(b)/Math.*log*(2)+3)];

r[0]=a;

r[1]=b;

**int** i=0;

**do** {

i++;

q[i]=r[i-1]/r[i];

r[i+1]=r[i-1]%r[i];

}**while** (r[i+1]!=0);

**return** r[i];

}

**public** **int** extendedEuclidean(**int** a,**int** b) {

**if** (b==0) {

u=1;

v=0;

**return** a;

}

**int** r= extendedEuclidean(b,a%b);

**int** t=u;

u=v;

v=t-a/b\*v;

**return** r;

}

**public** **long** fastPowering(**int** g,**int** A,**int** N) {

**int** r=(**int**) (Math.*log*(A)/Math.*log*(2));

**long**[] a=**new** **long**[r+1];

**int**[] b=**new** **int**[r+1];

**long**[] tp=**new** **long**[r+1];

**int** i;

/\* 2^i \*/

tp[0]=1;

**for** (i=1;i<=r;i++) tp[i]=2\*tp[i-1];

i=r;

**int** G=A;

**while** (G>0) {

//System.out.println("Jin"+i+" "+G+" "+tp[i]);

**if** (G>=tp[i]) {

b[i]=1;

G-=tp[i];

}

**else** b[i]=0;

i--;

}

a[0]=g % N;

**for** (i=1;i<=r;i++) a[i]=a[i-1]\*a[i-1] % N;

**long** ans=1;

**for** (i=0;i<=r;i++)

**if** (b[i]==1) ans=ans \* a[i] % N;

**return** ans;

}

}

**TestFundamentalAlgorithms.java**

**package** MC02;

**import** java.util.Scanner;

**public** **class** TestFundamentalAlgorithms {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

FundamentalAlgorithms FA=**new** FundamentalAlgorithms() ;

Scanner input=**new** Scanner(System.***in***);

System.***out***.print("求两个整数的最大公约数的欧几里得算法,请输入a与b:");

**int** a=input.nextInt();

**int** b=input.nextInt();

FA=**new** FundamentalAlgorithms();

System.***out***.println("gcd(a,b)="+FA.euclidean(a,b));

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

System.***out***.print("求两个整数的最大公约数和线性组合的扩展欧几里得算法,请输入a与b:");

a=input.nextInt();

b=input.nextInt();

**int** r=FA.extendedEuclidean(a, b);

System.***out***.println("gcd(a,b)="+r);

System.***out***.println("au+bv="+a+"·"+FA.getU()+"+"+b+"·"+FA.getV()+"="+r);

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

System.***out***.print("整数的模逆算法,请输入代求元a与素数p:");

a=input.nextInt();

**int** p=input.nextInt();

FA.extendedEuclidean(a, p);

System.***out***.println("a^(-1)="+Math.*abs*(FA.getU()) % p);

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

System.***out***.print("整数的快速模幂算法g^A mod N,请输入g,A,N:");

**int** g=input.nextInt();

**int** A=input.nextInt();

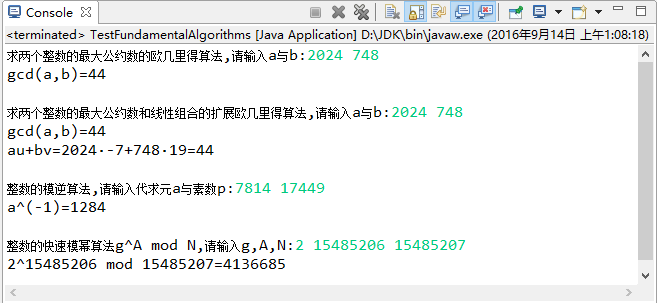
**int** N=input.nextInt();

System.***out***.println(g+"^"+A+" mod "+N+"="+FA.fastPowering(g,A,N));

}

}

运行结果：



大整数版本：

**FundAl.java**

**package** MC02;

**import** java.math.BigInteger;

**public** **class** FundAl {

**protected** BigInteger u;

**protected** BigInteger v;

**protected** BigInteger zero;

**public** FundAl() {

zero=**new** BigInteger("0");

}

**public** BigInteger getU() {

**return** u;

}

**public** BigInteger getV() {

**return** v;

}

**public** BigInteger euclidean(BigInteger a,BigInteger b) {

**if** (b.compareTo(zero)==0) **return** a;

**return** euclidean(b,a.mod(b));

}

**public** BigInteger extendedEuclidean(BigInteger a,BigInteger b) {

**if** (b.compareTo(zero)==0) {

u=**new** BigInteger("1");

v=**new** BigInteger("0");

**return** a;

}

BigInteger r= extendedEuclidean(b,a.mod(b));

BigInteger t=u;

u=v;

v=t.subtract(a.divide(b).multiply(v));

**return** r;

}

**public** BigInteger fastPowering(BigInteger g,BigInteger A,BigInteger N) {

BigInteger a=g;

BigInteger b=**new** BigInteger("1");

BigInteger one=**new** BigInteger("1");

BigInteger two=**new** BigInteger("2");

**while** (A.compareTo(zero)!=0) {

**if** (A.mod(two).compareTo(one)==0) b=b.multiply(a).mod(N);

a=a.multiply(a);

A=A.divide(two);

}

**return** b;

}

}

**TestFundAl.java**

**package** MC02;

**import** java.math.BigInteger;

**import** java.util.Scanner;

**public** **class** TestFundAl {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

FundAl FA=**new** FundAl() ;

Scanner input=**new** Scanner(System.***in***);

System.***out***.print("求两个整数的最大公约数的欧几里得算法,请输入a与b:");

BigInteger a=input.nextBigInteger();

BigInteger b=input.nextBigInteger();

FA=**new** FundAl();

System.***out***.println("gcd(a,b)="+FA.euclidean(a,b));

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

System.***out***.print("求两个整数的最大公约数和线性组合的扩展欧几里得算法,请输入a与b:");

a=input.nextBigInteger();

b=input.nextBigInteger();

BigInteger r=FA.extendedEuclidean(a, b);

System.***out***.println("gcd(a,b)="+r);

System.***out***.println("au+bv="+a+"·"+FA.getU()+"+"+b+"·"+FA.getV()+"="+r);

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

System.***out***.print("整数的模逆算法,请输入代求元a与素数p:");

a=input.nextBigInteger();

BigInteger p=input.nextBigInteger();

FA.extendedEuclidean(a, p);

System.***out***.println("a^(-1)="+FA.getU().mod(p).abs());

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

System.***out***.print("整数的快速模幂算法g^A mod N,请输入g,A,N:");

BigInteger g=input.nextBigInteger();

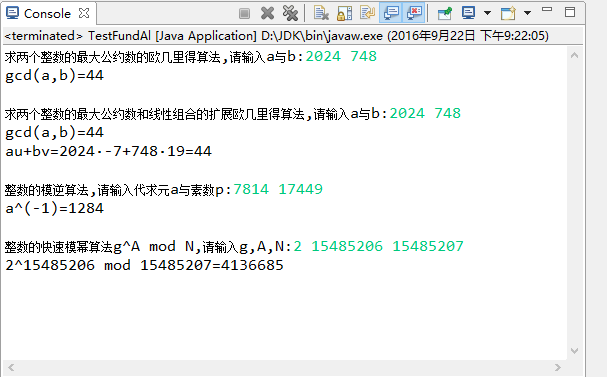
BigInteger A=input.nextBigInteger();

BigInteger N=input.nextBigInteger();

System.***out***.println(g+"^"+A+" mod "+N+"="+FA.fastPowering(g,A,N));

}

}



**五、实验体会**

**（请认真填写自己的真实体会）**

1.学会了密码学中的几个基础算法。欧几里得基本算法和拓展欧几里得算法都用到了递归的思想，实现起来比较容易。

2.对于求一个元素的模逆元素，可以用两种方法，一是使用拓展欧几里得，但需要找到一个与待求元互素的数；当mod的是一个素数时，利用费马小定理得到，可以借助快速模幂算法实现。

3.快速模幂运算将指数分解，相比传统运算可以大大减少运算量。同时快速模幂运算和费马小定理提供了一种判断素数的新的方法，对任意给定的待判断数p，只需另找一个不能被p整除的整数a，运用快速模幂算法求判断其是否等于1，若为1，则p是素数，否则p不是素数。

4.欧几里得算法中：

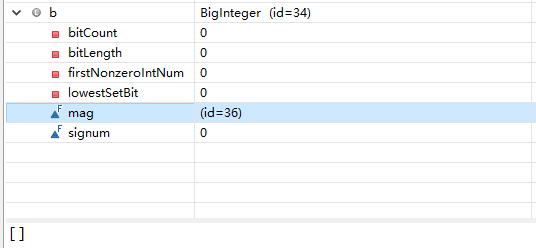
**public** BigInteger euclidean(BigInteger a,BigInteger b) {

**if** (b.equals(0)) **return** a;

**return** euclidean(b,a.mod(b));

}

采用如上代码，则终止条件上出现问题。在最后一步，b理应为0，但是实际上此时的b显示为空[],见下图：



从另一个角度来讲，[boolean equals(Object x)](http://www.yiibai.com/java/math/biginteger_equals.html)此方法比较此BigInteger与指定对象是否相等。

b.equals(0)中的0应为int型，与BigInteger型的b并不相等。需改为值的比较：[int compareTo(BigInteger val)](http://www.yiibai.com/java/math/biginteger_compareto.html) 此方法比较此BigInteger与指定的BigInteger。此方法返回-1，0或1，分类为BigInteger在数字上小于，等于，或大于值val。

**六、参考文献**

1. 主讲课教材（数学密码学导论）第一章

**2.（如有其它参考文献，请列出）**