信息安全数学基础
实验指导书

教师：韩琦教授

实验地点：格物楼 213

实验时间：2023.11.4

一、背景介绍

本次实验要求同学们利用 Java 编程语言对经典的有关信息安全的算法进行实践，包括：欧几里得算法、快速幂算法、素数判断算法、中国剩余定理，实现每一个算法时应该贴合实际需求以支持大数运算。

本次实验的目的包括：

1. 理解并熟悉信息安全算法的基本概念和原理。
2. 掌握实现和应用信息安全算法的方法。
3. 通过实验验证信息安全算法的安全性和有效性。
4. 比较不同信息安全算法的性能和特点。
5. 发现并分析信息安全算法的潜在漏洞和风险。

二、实验环境和预备知识

2.1 实验环境

- 操作系统：Windows7、Windows10、Windows11
- 编程语言：Java
- 代码编辑器：Eclipse

Ps：同学可以选择自己常用的代码编辑器。

2.2 编辑使用步骤和示例代码

本实验要求利用 Eclipse 进行实验，首先打开桌面上的 Eclipse 编辑器，选择实验进行的目录，如下图 2-1 所示，选择之后点击 Launch：

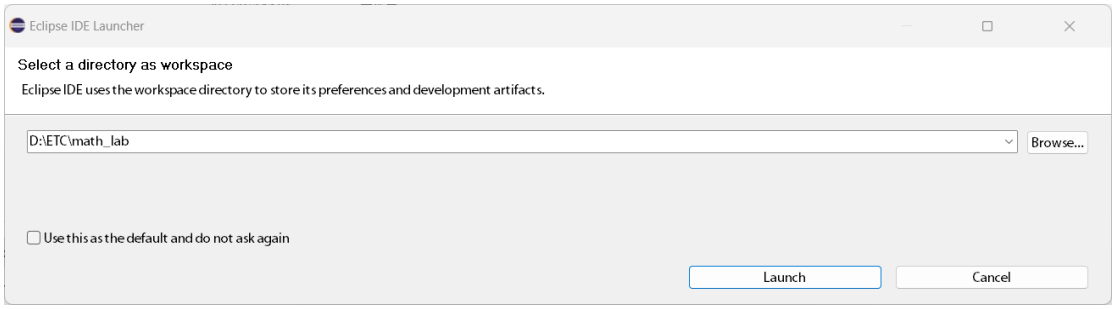

图 2-1 选择实验进行的目录

进入 Eclipse 主界面（图 2-2）之后依次点击：File→New→Java Project.

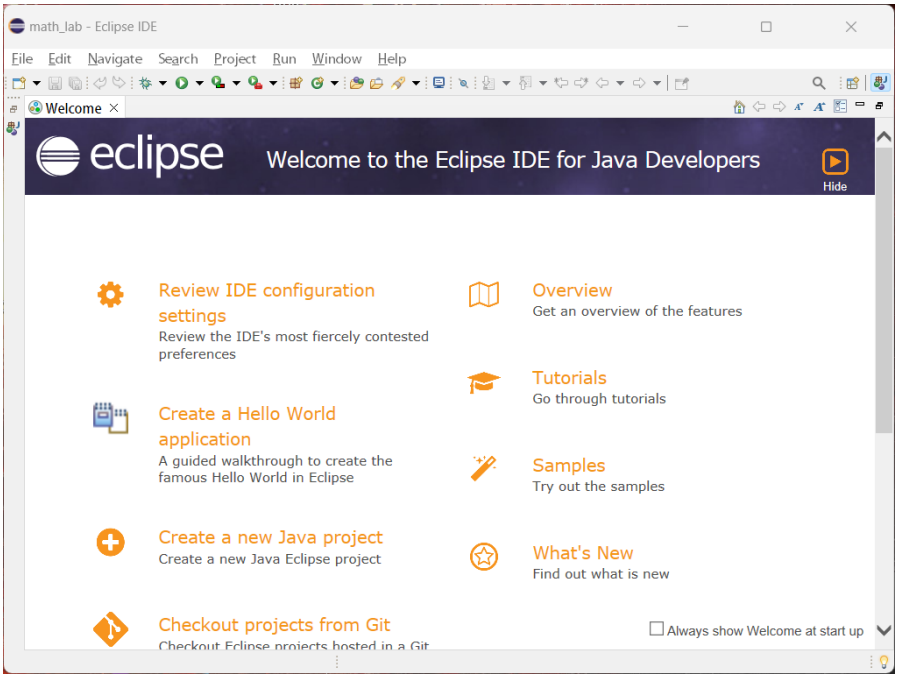

图 2-2 主界面

输入实验项目的命名，示例的项目的命名为 test，如下图 2-3，其它内容默认即可：

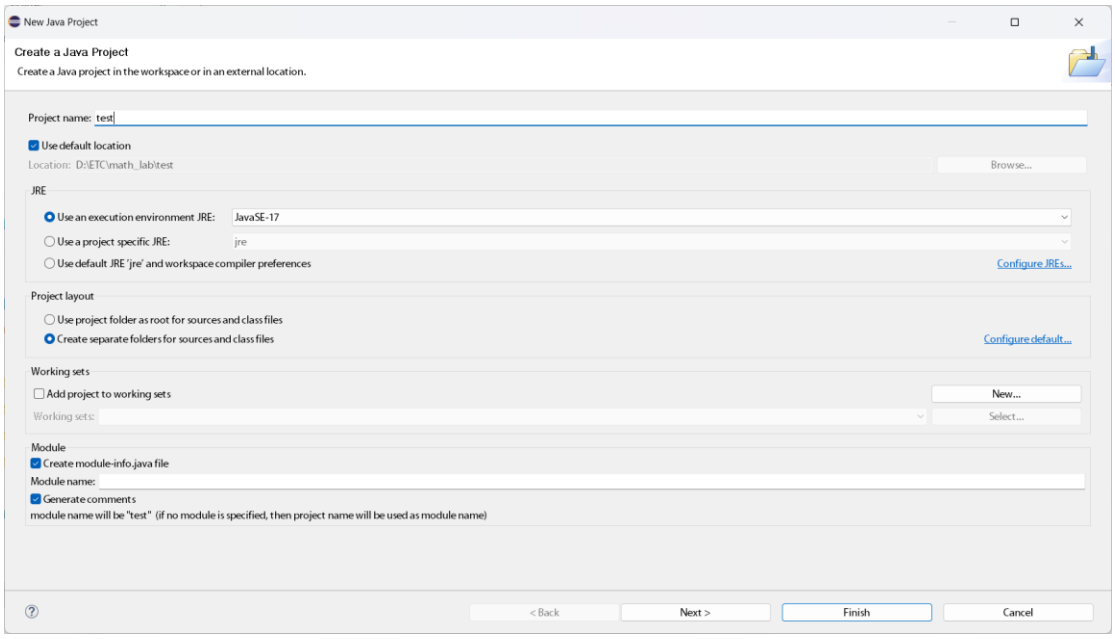

图 2-3 命名项目

点击屏幕左侧的双长方形的按钮如下图 2-4 所示，进入代码编辑主界面如图 2-5：

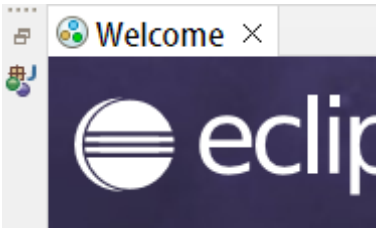

图 2-4 按钮 1

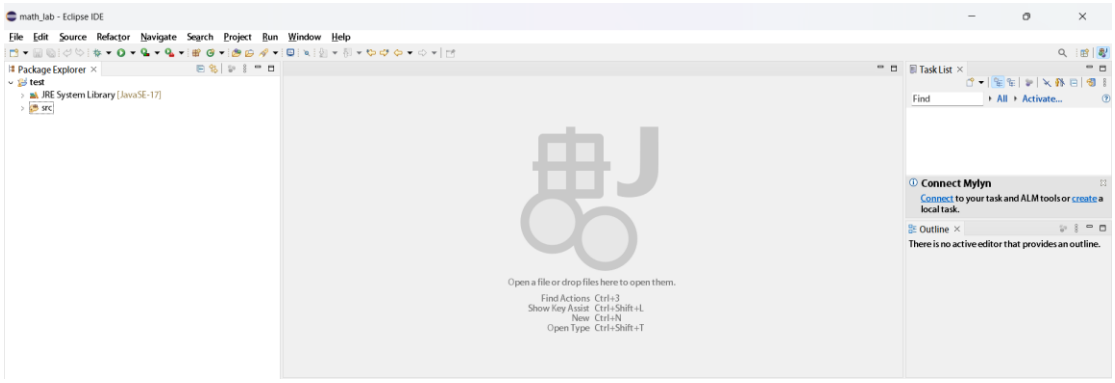

图 2-5 代码编辑主界面

右键点击 src，选择 new，再选择 class，来创建项目需要的各种类，在本示例中选择创建 main 类来运行代码，在“name”的输入栏中输入“main”，如下图 2-6 所示：

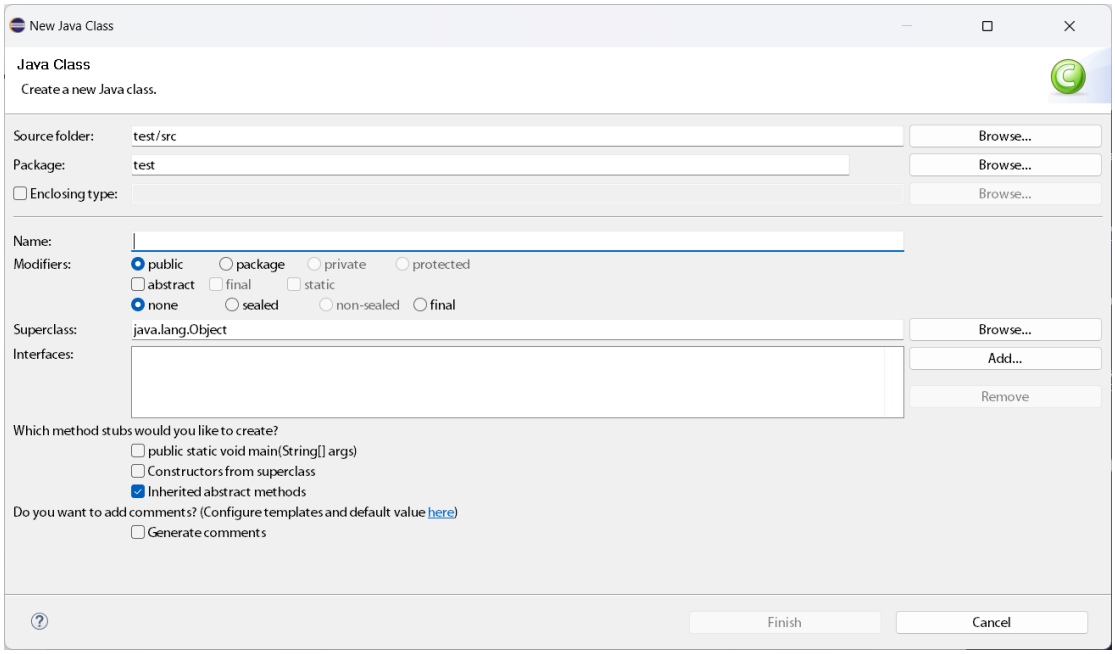

图 2-6 输入类名

点击 main.java 输入代码，如下图 2-7 所示：

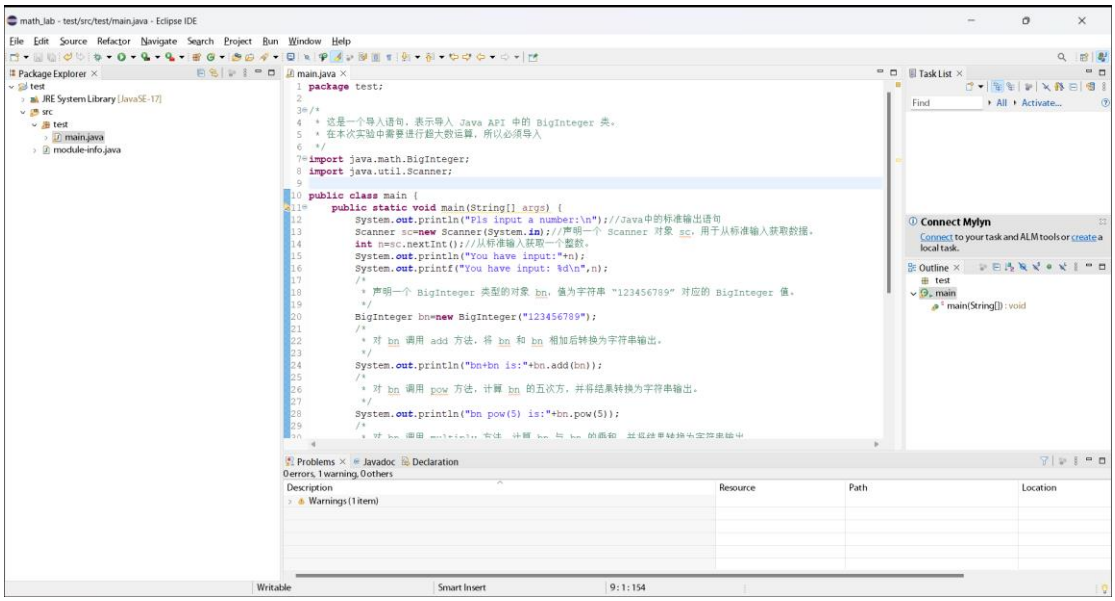

图 2-7 编辑代码

下面是实例的具体代码，同学们需要阅读下面代码和注释，以便对 Java 有初步的了解：

```
1. package test;
2.
3. /*
4.  * 这是一个导入语句，表示导入 Java API 中的 BigInteger 类。
5.  * 在本次实验中需要进行超大数运算，所以必须导入
6.  */
7. import java.math.BigInteger;
8. import java.util.Scanner;
```

```
9.
10. public class main {
11.     public static void main(String[] args) {
12.         System.out.println("Pls input a number:\n");//Java 中的标准输出语句
13.         Scanner sc=new Scanner(System.in);//声明一个 Scanner 对象 sc，用于从标准输入获取数据。
14.         int n=sc.nextInt();//从标准输入获取一个整数。
15.         System.out.println("You have input:"+n);
16.         System.out.printf("You have input: %d\n",n);
17.         /*
18.          * 声明一个 BigInteger 类型的对象 bn，值为字符串 “123456789” 对应的 BigInteger 值。
19.          */
20.         BigInteger bn=new BigInteger("123456789");
21.         /*
22.          * 对 bn 调用 add 方法，将 bn 和 bn 相加后转换为字符串输出。
23.          */
24.         System.out.println("bn+bn is:"+bn.add(bn));
25.         /*
26.          * 对 bn 调用 pow 方法，计算 bn 的五次方，并将结果转换为字符串输出。
27.          */
28.         System.out.println("bn pow(5) is:"+bn.pow(5));
29.         /*
30.          * 对 bn 调用 multiply 方法，计算 bn 与 bn 的乘积，并将结果转换为字符串输出。
31.          */
32.         System.out.println("bn*bn is:"+bn.multiply(bn));
33.
34.         sc.close();
35.         /*
36.          * 声明一个值为 0 的 BigInteger 类型变量 ans，
37.          * 以及一个值为 1 的 BigInteger 类型变量 temp。
38.          */
39.         BigInteger ans = new BigInteger("0"), temp=new BigInteger("1");
40.         for (int i=1;i<=n;i++) {
41.             //计算 i 的阶乘，并将结果保存在 temp 中。
42.             temp=temp.multiply(BigInteger.valueOf(i));
43.             //计算前 i 个数的阶乘之和，并将结果保存在 ans 中。
44.             ans=ans.add(temp);
45.         }
```

```
46.         System.out.println("The answer is :"+ans);
47.     }
48.
49. }
```

编辑完代码之后点击下图 2-8 所示的运行按钮，运行代码：

图 2-8 运行按钮

在编辑器下方出现代码运行结果，如下图 2-10 所示，用户可以在这里进行输入，示例中输入的 123：

```
Pls input a number:

123
You have input:123
You have input: 123
bn+bn is:246913578
bn pow(5) is:28679718602997181072337614380936720482949
bn*bn is:1524157875019052l
The answer is :122458709846074208432659661496943456132638002656295929580755149558615989953699967782139459444357519119903972237835929087782422
```

图 2-9 运行结果

2.3 java.math.BigInteger 的 API 说明

java.math.BigInteger 是 Java 中的一个类，用于表示任意精度的整数。该类提供了一系列方法来执行基本的算术操作、比较和位操作。以下是 BigInteger 类的一些常用的 API 说明：

1. 创建 BigInteger 对象：

- BigInteger(String val)：使用指定的字符串值创建一个 BigInteger 对象。
- BigInteger(int signum, byte[] magnitude)：使用指定的符号和字节数组创建一个 BigInteger 对象。

2. 基本操作：

- add(BigInteger val)：将 val 与当前对象相加，返回新的 BigInteger 对象。
- subtract(BigInteger val)：从当前对象中减去 val，返回新的 BigInteger 对象。
- multiply(BigInteger val)：将当前对象乘以 val，返回新的 BigInteger 对象。
- divide(BigInteger val)：将当前对象除以 val，返回商的 BigInteger 对象。
- remainder(BigInteger val)：返回当前对象除以 val 的余数，作为 BigInteger 对象。
- pow(int exponent)：将当前对象的指数幂返回为一个新的 BigInteger 对象。

3. 比较操作：

- equals(Object x)：将当前对象与 x 进行比较，返回一个布尔值，表示两个对象是否相等。

- compareTo(BigInteger val)：将当前对象与 val 进行比较，返回一个整数值：-1 表示当前对象小于 val；0 表示当前对象等于 val；1 表示当前对象大于 val。

4. 位操作：

- and(BigInteger val)：将当前对象与 val 进行按位与操作，返回新的 BigInteger 对象。
- or(BigInteger val)：将当前对象与 val 进行按位或操作，返回新的 BigInteger 对象。
- xor(BigInteger val)：将当前对象与 val 进行按位异或操作，返回新的 BigInteger 对象。
- shiftLeft(int n)：将当前对象左移 n 位，返回新的 BigInteger 对象。
- shiftRight(int n)：将当前对象右移 n 位，返回新的 BigInteger 对象。

这只是 BigInteger 类的一部分 API 说明，还有其他方法可用于更高级的操作，可以参考 Java 官方文档以获取更详细的信息。

三、实验内容

3.1 欧几里得算法和拓展欧几里得算法

欧几里得算法又称辗转相除法。古希腊数学家欧几里得在其著作 The Element 中最早描述了这种算法，所以该算法又叫欧几里得算法。算法利用公式$gcd(a,b) = gcd\ (b, a\ mod\ b)$，求两个非负整数$a$和$b$的最大公约数。欧几里得算法具体步骤如下：

（1） 使用带余除法，b除a得到余数r；

（2） 若$r > 0$，则用b代替a，用r代替b，重复第（1）步；

（3） b 的值就是最大公约数d。

欧几里得算法流程图如图 3-1 所示。

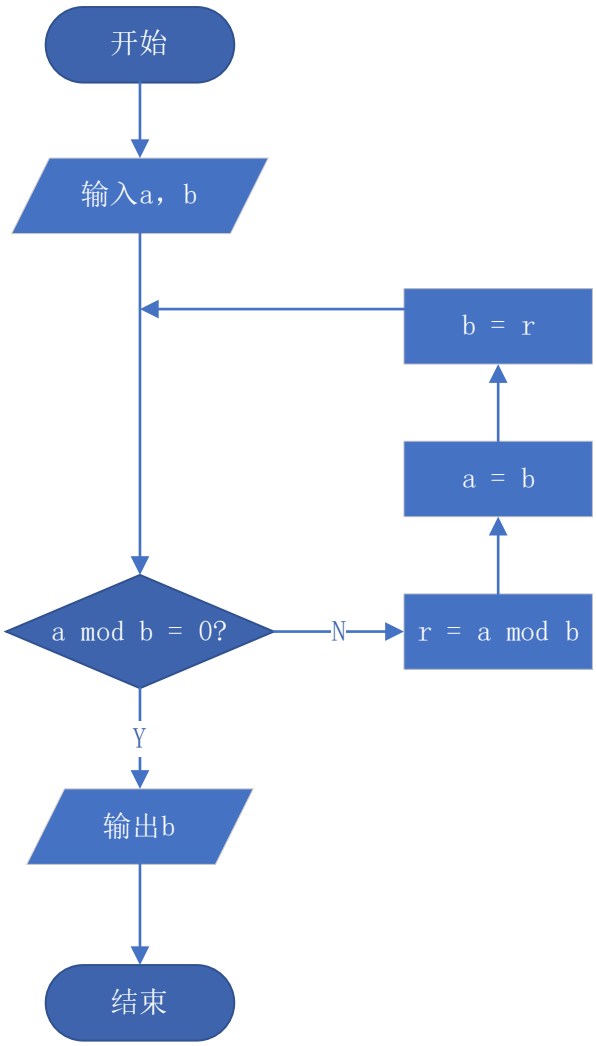

图 3-1 欧几里得算法流程

因为欧几里得算法相当简单，所以在这里给同学们提供了完整的欧几里得

算法的可运行代码作为后续算法代码的模板和范本：

```java
package euclid;

import java.math.BigInteger;
import java.util.Scanner;

public class main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the first number: ");
        BigInteger a = sc.nextBigInteger();

        System.out.print("Enter the second number: ");
        BigInteger b = sc.nextBigInteger();

        BigInteger result = euclid(a, b);
        System.out.println("The greatest common divisor is: " + resul
t);

        sc.close();
    }

    public static BigInteger euclid(BigInteger a, BigInteger b) {
        while (!b.equals(BigInteger.ZERO)) {
            BigInteger temp = a;
            a = b;
            b = temp.mod(b);
        }
        if (b.equals(BigInteger.ZERO)) {
            return a;
        }
        // return null or do appropriate handling based on your requi
rements
        // 根据你的需求返回 null 或根据需求进行适当的处理
        return null;
    }
}
```

拓展欧几里得算法是欧几里得算法的拓展，基于以下定理：对任意两个整数a、b，必存在整数x、y，使得$ax+by=gcd\ (a,b)$成立，求出任意整数解x、y，可以得到$gcd\ (a,b)$。拓展欧几里得算法具体步骤如下：

（1）根据带余除法，并假设每个步骤i都可以找到对应的x_i和y_i，那么可以

列出公式：

$$a = q_1 b + r_1, r_1 = ax_1 + by_1$$
$$b = q_2 r_1 + r_2, r_2 = ax_2 + by_2$$
$$r_1 = q_3 r_2 + r_3, r_3 = ax_3 + by_3$$
$$\cdots$$
$$r_{n-2} = q_n r_{n-1} + r_n, r_n = ax_n + by_n$$
$$r_{n-1} = q_{n+1} r_n + 0$$

（2）通过移项得到：

$$r_i = q_i r_{i-1} + r_{i-2}$$

（3）以此类推，将$r_{i-2} = ax_{i-2} + by_{i-2}$和$r_{i-1} = ax_{i-1} + by_{i-1}$代入$r_i = q_i r_{i-1} + r_{i-2}$得到：

$$r_i = a(x_{i-2} - q_i x_{i-1}) + b(y_{i-2} - q_i y_{i-1})$$

（4）由$r_i = ax_i + by_i$，得到：

$$x_i = x_{i-2} - q_i x_{i-1}, y_i = y_{i-2} - q_i y_{i-1}$$

（5）依次递推，直到$x_0 = 0, y_0 = 1; x_{-1} = 1, y_{-1} = 0$。

（6）将中间公式迭代即可得出x_n和y_n，即所求的x和y：与此同时，可得a和b的最大公约数$d = r_n = ax_n + by_n$。

拓展欧几里得算法的输入为两个正整数a、b，输出为其最大公约数 GCD 和x、y，使得$ax + by = GCD$，下表 3-1 是该算法的测试数据和结果：

表 3-1 测试用例

a	b	x	y	$gcd\ (a,b)$
31	-13	-5	-12	1
123456789012345 67890	14702583691470258 369	-27717093	23273891	90000000009
197541167019754 116701975411670 197541167020010 222431195175345 6258789632145	- 98563214789654123 69874125753159456 25877889952416313 46798246	40799585089 99703306452 79827313751 76471785593 57692364804 6133	81770847976 09264649892 59836955996 99000597338 97854976085 53671559861 840054	1

3.2 快速幂取模算法

若要计算$a^b \bmod p$，可以先计算$a^2 \bmod p$，$a^4 \bmod p$，$a^8 \bmod p$，……，再将b的二进制表示中等于 1 的位对应的a的幂相乘，便可以得到结果，具体步骤如下：

（1） 将b表示成二进制数 0 和 1 的字符串，令初始结果为 1；

（2） 从b的最低位开始，若当前位为 1，则将当前结果乘以a模p；若当前

位为 0，则当前结果不变；

（3） 将a变为a的平方模p；

（4） 重复步骤（2）和步骤（3），直到b的每一位运算完毕，当前结果即为最终结果。

算法的输入为底数$base$，指数$expo$和模数p，输出为$base$的$expo$次方模p的值$result$。下表 3-2 是快速幂取模算法的两个测试用例：

表 3-2 测试用例

底数$base$	指数$expo$	模式p	取模结果
5	10003	31	5
1494462659429290047815067355171411187560751791530	65537	22688387113047243043043961195094167745977232924 74	2099538163720891467842744895846522520832379454230

3.4 Miller-Rabin 素性检测算法

根据费马小定理：设p是素数，a为整数，且$(a,p)=1$，则$a^{p-1}\equiv 1(mod\ p)$，以及二次探测定理：如果p是一个素数，且$0<x<p$，且方程$x^2\equiv 1(mod\ p)$成立，那么$x=1$或$x=p-1$.Miller-Rabin 素性检测算法是基于以上两个定理的随机化算法，用于判断一个数是合数还是素数，判断n是否为素数的具体步骤如下：

（1） 令$n-1=2^k q$，其中$k>0$，q为奇数，随机选取整数a，$1<a<n-1$；

（2） 若$a^q\ mod\ n=1$，则n有可能是素数；

（3） 取整数j，$0\le j<k$，若存在$a^{2^j q}\ mod\ n=n-1$，则n有可能是素数；否则，n为合数。

由以上分析可知，素数一定能通过测试，不通过测试的必为合数，通过测试的很可能就是素数。

算法的输入为正整数n，若n不为素数，则返回$false$，否则返回$true$。该算法的测试数据如下表 3-3：

表 3-3 测试数据

输入数据	输出检测结果
1000023	False
1000033	True
100160063	False
1500450271	True

3.5 厄拉多塞筛选算法

厄拉多塞筛选算法是一种求素数的方法。它的原理是，给定一个数n，从 2 开始依次将$\sqrt{n}$以内的素数的倍数标记为合数，标记完成后，剩余未被标记的数为素数（从 2 开始）。如此可省去检查每个数的步骤，使筛选素数的过程更加简单。厄拉多塞筛选算法具体步骤如下;

（1） 读取输入的数n，将 2 至n的所有整数记录在表中；

（2） 从 2 开始，划去表中所有 2 的倍数；

（3） 由小到大寻找表中下一个未被划去的整数，再划去表中所有该整数的倍数；

（4） 重复步骤（3），直到找到的整数大于$\sqrt{n}$为止；

（5） 表中所有未被划去的整数均为素数。

算法的输入为正整数n，输出为n以内的所有素数。例如：输入 10，输出 2、3、5、7；输入 20，输出 2、3、5、7、11、11、13、17、19。

3.6 中国剩余定理

用现代语言描述，可以给出一元线性同余方程组(S)：

$$x \equiv a_1(mod\ m_1)$$
$$x \equiv a_2(mod\ m_2)$$
$$\cdots\cdots$$
$$x \equiv a_n(mod\ m_n)$$

假设整数m_1、m_2、……、m_n两两互质，则对任意的整数a_1、a_2、……、a_n，方程组有解，并且通解可以用如下的方式构造得到：

（1） 设$M = \prod_{i=1}^{n} m_i$，并设$M_i = \frac{M}{m_i}$，$i \in \{1，2，\ \ldots\ldots，n\}$，是除了$m_i$以外的$n-1$个数的乘积。

（2） 设$t_i = M_i^{-1}$是M_i模m_i的逆元，$t_i M_i \equiv 1(mod\ m_i)$，$i \in \{1，2，\ \ldots\ldots，n\}$

（3） 方程组的(S)的通解形式为：

$$x = kM + \sum_{i=1}^{n} a_i t_i M_i，\ k \in Z$$

则在模M的意义下，方程组(S)只有一个解$x = \sum_{i=1}^{n} a_i t_i M_i$。

算法的输入为余数数组a、模数数组p以及方程个数num，输出为同余方程的结果$result$。下表 3-4 给出了中国测试定理的测试数据：

表 3-4 测试数据

a List	m List	M_i List	t_i List	x
0,0,0	23,28,33	924,759,644	6,-9,2	0(mod 21252)
5,20,34	23,28,33	924,759,644	6,-9,2	19900(mod 21252)
283,102,23	23,28,33	924,759,644	6,-9,2	9230(mod 21252)

四、实验要求

本次实验要求尽量实现上述算法，并且能够运行包括但不限于第三章给出的测试数据。

在实验报告中应该给出算法的运行逻辑、算法的代码和运行的结果。代码和实验报告以打包的方式发送到助教的邮箱。