摘要

信息安全数学基础是信息安全方向的一门专业基础课，内容以数学理论基础为主，内容较抽象，需要与编程实践结合加深理解。因此本文对信息安全数学基础中初等数论部分中的一些定理与方法进行python编程实现，包括辗转相除法，素数的应用，同余判断，中国剩余定理的应用等。

在每一节中本文首先对会涉及的数论基础知识进行介绍，之后对将要实现的定理与算法进行讲解，最后是使用python3.7编程实现并展示对应的结果。将课程内容进行编程实现可以提高对课本内容的理解，并提高编程实践能力，学会理论与实践的结合。

关键字： 信息安全数学基础；初等数论；辗转相除法；埃拉托斯特筛法；中国剩余定理；python；

Abstract

Mathematical foundation of information security is a professional basic course of information security direction. The content is mainly based on mathematical theory, and the content is abstract, which needs to be combined with programming practice to deepen understanding. Therefore, some theorems and methods in the elementary number theory part of the mathematical foundation of information security are implemented by Python programming, including the division of tossing and turning, the application of prime numbers, congruence judgment, and the application of The Chinese remainder theorem.

In each section, this paper first introduces the basic knowledge of number theory that will be involved, then explains the theorems and algorithms to be implemented, and finally uses python3.7 programming implementation and shows the corresponding results. Programming the course content can improve the understanding of the textbook content, improve the ability of programming practice, learn the combination of theory and practice.

Keyword： Mathematical Foundations in Information Security;Elementary Number Theory; Division Algorithm；Sieve of Eratosthenes;Chinese Remainder Theorem; Python

目录

0 引言

随着信息技术的高速发展，人们对信息安全的关注越来越深入。十八大以来，以习近平同志为总书记的党中央高度重视网络安全和信息化工作。习近平在中央网络安全和信息化领导小组第一次会议上强调，网络安全和信息化是事关国家安全和国家发展、事关广大人民群众工作生活的重大战略问题。信息安全的概念在 20 世纪提出后历经了漫长的发展，20 世纪末才被人们深入研究。信息安全作为一门交叉性学科，具有很强的综合性，其涉及诸多学科，如数学、密码学、计算机科学、信息学等，其目的是保障信息在存储和传输过程中的保密性、完整性、不可否认性等特性[1]。

而信息安全数学基础是信息安全方向的一门专业基础课，以密码学为核心内容，建立在数学理论基础之上并涉及多个数学分支，对信息安全学习中的相关数学理论进行系统的介绍。该课程涉及初等数论、近世代数、有限域、离散数学和计算复杂度的内容，其教学内容的最大特点是知识点具有很强的逻辑性和抽象性，内容比较晦涩难懂，课程内容较多较难且与实际联系不大，难以清楚所学内容的实际应用[2, 3]。尤其是初等数论部分，涉及的数学原理偏向单纯的理论，需要通过一定的编程实践来加深对数学原理及具体算法的理解。因此，本文将对《信息安全数学基础》的第一章整数的因子分解，第二章同余式中的部分定理与方法进行编程实现。

Python 由 Guido van Rossum 于 1989 年底发明，第一个公开发行版发行于 1991 年,是一种解释型、面向对象、动态数据类型的高级程序设计语言，支持多种编程范例，包括面向对象、命令式、函数时和过程式，拥有一个大型的标准库。以前的大多数信息安全编程实践都采用C、C++或者JAVA实现，但随着Python这一简洁易懂且具有强大科学计算能力的计算机语言的盛行，尤其是大量开发软件包的出现，python在信息安全领域也有了一席之地，PyCryptodome库也为python在信息安全领域的应用提供了帮助。因此，本文中的编程实现都选用python语言，使用python3.7版本。

1 辗转相除法

1.1 数论基础

(1) **带余除法**：设a和b为整数，$b>0$,则存在唯一的整数q和r使得

$$a=qb+r,\quad 0\leqslant r<b \tag{1.1}$$

式(1.1)称为带余除法，或称为欧几里得除法。q被称为a被b除得出的**不完全商**，r称为**余数**，余数都是非负整数。

(2) **整除**：当(1.1)中$r=0$，即

$$a=qb$$

时，称b能整除a,记为$b|a$,其中“$|$”为整除符号。这时称b是a的**因子**，a是b的**倍数**。

(3) **公因子**：设a,b是两个非零整数，d为正整数，若

$$d|a,a|d,$$

则称d为a和b的公因子。

(4) **最大公因子**：对a和b的公因子d，有$d\leqslant|a|,d\leqslant|b|$,因此$a$和$b$仅可能有有限个公因子，其中最大的一个称为$a$和$b$的**最大公因子**，记为$\boldsymbol{(a,b)}$

(5) **定理1.1**：设a,b,c为三个正整数，且

$$a = bq + c,$$

其中q为整数,则$\boldsymbol{(a,b)=(b,c)}$

1.2 辗转相除法

辗转相除法又称欧几里得算法，是一种计算两个正整数a和b的最大公因子的算法，古希腊数学家欧几里得在其著作《The Elements》中最早描述了这种算法,所以被命名为欧几里得算法。其计算公式为$\gcd(a,b)=\gcd(b,a \bmod b)$

辗转相除法利用了带余除法以及定理1.1，其计算过程为：

设

$$a = q_0 b + r_0, \quad 0 \leqslant r_0 < b,$$

如果$r_0 \neq 0$，设

$$b = q_1 r_0 + r_1, \quad 0 \leqslant r_1 < r_0,$$

如果$r_1 \neq 0$，设

$$r_0 = q_2 r_1 + r_2, 0 \leqslant r_2 < r_1,$$

以此类推，设

$$r_{i-2} = q_i r_{i-1} + r_i, \quad 0 \leqslant r_i < r_{i-1}, \quad i = 3,4,\cdots$$

因为$r_0 > r_1 > r_2 > \cdots \geq 0$,故到某一步必有$r_n = 0$，这时$r_P n - 2 = q_n r_{n-1}$,即$r_{n-1} | r_{n-2}$。
由定理1.1可得

$$(a,b) = (b,r_0) = (r_0,r_1) = \cdots = (r_{i-1},r_i) = \cdots = (r_{n-2},r_{n-1}) = r_{n-1}$$

即r_{n-1}是a和b的最大公因子。

1.3 代码实现

现在使用python编程实现用辗转相除法求两个数的最大公因子。

由于辗转相除法使用的是求两个正整数的最大公因子的情况，因此要实现求任意两个整数的最大公因子，要考虑非正整数的情况。

(1) 由于0可以被任何整数整除，所以任意整数a与0的最大公因子就是a,两个0的最大公因子定义为0。

(2) 设d是a的因子，则存在整数q使得$a = dq$,因此$-a = d(-q)$,d也是$-a$的因子，由此可得$(-a,b) = (a,b)$。因此，对于负整数与整数的最大公因子的求解可以化为其相反数与整数的最大公因子求解。

下面依据辗转相除法的算法原理给出完整的求最大公因子的程序代码：

```
def division_algorithm(m,n):
  while m*n!=0:
      m=m%n
      if(m==0):
          return n
      else:
          n=n%m
          if(n==0):
```

```
            return m

a=int(input("请输入第一个整数:"))
b=int(input("请输入第二个整数:"))
gcd=0
if(a<0):
    a=-a
if(b<0):
    b=-b
if(a==0 and b==0):
    gcd=0
elif(a==0 or b==0):
    gcd=a+b
else:
    gcd=division_algorithm(a,b)
print("最大公因子为: ")
print(gcd)
```

上述代码依据原理一步步建立。由于python具有高效简洁的特性，我们可以采用更简洁的代码编写辗转相除法的函数：

```
def division_algorithm(m,n):
  while(n%m!=0):
      m,n=n%m,m
  return m
```

采用简洁的辗转相除法函数的完整代码请见附录。

1.4 结果展示

下面是不同情况下程序结果的展示：

```
请输入第一个整数:8
请输入第二个整数:4
最大公因子为:
4

Process finished with exit code 0
```

```
请输入第一个整数:4
请输入第二个整数:-8
最大公因子为:
4

Process finished with exit code 0
```

```
请输入第一个整数:16
请输入第二个整数:0
最大公因子为:
16

Process finished with exit code 0
```

```
请输入第一个整数:0
请输入第二个整数:0
最大公因子为:
0

Process finished with exit code 0
```

2 素数

2.1 数论基础

(1) **素数**：一个大于1的正整数p，如果仅以1和自身p作为其因子，则称p为**素数**。大于1的非素数的自然数称为**复合数**

(2) **互素**：如果两个整数a和b的最大公因子等于1，即$(a,b)=1$，则称a和b**互素**。

2.2 互素的判断

根据互素的定义，两个整数互素的判断可以通过求它们的最大公因子实现。

2.2.1 代码实现

判断两个整数是否互素的函数如下

```
def Relatively_prime(a,b):
  if(a<0):
      a=-a
  if(b<0):
      b=-b
  if(a==0 and b==0):
      gcd=0
  elif(a==0 or b==0):
      gcd=a+b
  else:
      gcd=division_algorithm(a,b)
  if(gcd==1):
      return True
  else: return False
```

完整程序代码请见附录。

2.2.2 结果展示

使用上述程序判断两个整数是否互素的结果如下：

```
请输入第一个整数:13
请输入第二个整数:17
13和17互素

Process finished with exit code 0
```

```
请输入第一个整数:32
请输入第二个整数:128
32和128不互素

Process finished with exit code 0
```

2.3 Erarosthenes筛法求素数

2.3.1 定义

埃拉托斯特尼筛法是求不超过自然数N(N¿1)的所有质数的一种方法。

埃拉托斯特尼是一位古希腊数学家,他在寻找整数N以内的素数时,采用了一种与众不同的方法:先将2—N的各数写在纸上:在2的上面画一个圆圈,然后划去2的其他倍数; 第一个既未画圈又没有被划去的数是3,将它

画圈,再划去3的其他倍数;现在既未画圈又没有被划去的第一个数是5,将它画圈, 并划去5的其他倍数……依此类推,一直到所有小于或等于N的各数都画了圈或划去为止。这时,画了圈的以及未划去的那些数正好就是小于N的素数[4]。

2.3.2 代码实现

Erarosthenes筛法寻找素数的代码如下：

```
def sieve_func(n):
    prime=[]
    sieve =[True] *(n+1)
    for i in range(2, n):
        if sieve[i]:
            prime.append(i)
            for j in range(i *i, n, i):
                sieve[j] =False
    return prime

n=input("请输入素数的范围: n")
print(sieve_func(int(n)))
```

2.3.3 结果展示

用Erarosthenes筛法寻找1000以内的素数，结果如下：

```
请输入素数的范围n: 1000
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149,
 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311,
 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487,
 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673,
 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877,
 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]
```

下面我们对程序进行修改，以计算n较大时Erarosthenes筛法的运行速度。修改后的程序代码请见附录。

分别取$n = 100000, n = 10000000, n = 100000000$,得到结果：

```
请输入素数的范围n: 1000000
运行时间: 0.18102097511291504

Process finished with exit code 0
请输入素数的范围n: 10000000
运行时间: 2.057182788848877

Process finished with exit code 0
请输入素数的范围n: 100000000
运行时间: 25.175408601760864

Process finished with exit code 0
```

由结果分析可得，使用Erarosthenes筛法寻找100000000以内的素数的速度是比较快的。

3 同余的应用

十九世纪初，现代数论第一人的数学家高斯，出版了《算术研究》一书，并在其中提出了同余的概念及一次同余式组的解法，极大地丰富了数学的内容，为现代同余理论的发展奠定了牢固的基础.。同余理论在初等数论中占有重要的地位，是研究整数问题的重要手段[5]。

3.1 数论基础

(1) **同余**：设n为自然数，a,b为任意两个整数，若$a-b$能被n除尽，则称a与b模n同余，记为

$$a \equiv b \pmod{n}$$

此时，n除a所得的余数与n除b所得的余数相同。

(2) **同余的基本性质**：

1) 对所有的a，$a \equiv a \ (mod\ n)$；

2) 若$a \equiv b \ (mod\ n)$,则$b \equiv a \ (mod\ n)$

3) 若$a \equiv b \ (mod\ n)$，$b \equiv c \ (mod\ n)$则$a \equiv c \ (mod\ n)$。

(3) **解同余方程**：给定整数a，正整数m，求整数x，使

$$x \equiv a \ (mod\ m)$$

这称为解同余方程问题。此时$x = a + km$(k为任一整数)是同余方程的所有解。

(4) **扩展欧几里得算法**：是辗转相除法的扩展，已知整数a, b，扩展欧几里得算法可以再求得a,b的最大公因子的同时，找到整数x, y，使得：

$$ax + by = \gcd(a, b)$$

在本节中，扩展欧几里得算法将用来在中国剩余定理中计算模反元素(也叫模逆元)。

3.2 同余判断

判断两个整数是否同余是同余的基本应用。其实现依据于同余的定义。

3.2.1 代码实现

判断两个整数是否同余的代码如下：

```
def Congruence_judgment(a,b,n):
  if((a-b)%n==0):
      return True
  else:
      return False

a=int(input("请输入第一个整数:"))
b=int(input("请输入第二个整数:"))
n=int(input("请输入模n:"))

if(Congruence_judgment(a,b,n)):
```

```
12    print("{}与{}模{}同余".format(a,b,n))
13  else:
14    print("{}与{}模{}不同余".format(a,b,n))
```

3.2.2 结果展示

以下展示使用判断同余程序判断两个整数是否模一个整数同余：

```
请输入第一个整数:121
请输入第二个整数:13
请输入模n:2
121与13模2同余

请输入第一个整数:10
请输入第二个整数:3
请输入模n:5
10与3模5不同余
```

3.3 中国剩余定理

3.3.1 定义

中国剩余定理也称为孙子定理，是中国古代求解一次同余式组的方法，是数论中一个重要定理，来源于我国古代的经典数学著作《孙子算经》中的算术题：“今有物不知其数，三三数之剩二，五五数之剩三，七七数之剩二，问物几何？”

其给出的解法为：“三、三数之剩二，置一百四十；五、五数之剩三，置六十三；七、七数之剩二，置三十，并之，得二百三十三。以二百一十减之，即得。凡三、三数之剩一，则置七十；五、五数之剩一，则置二十一；七、七数之剩一，则置十五。一百六以上，一百五减之，即得。”它的算法即为：

$$2 \times 70 + 3 \times 21 + 2 \times 15 = 233,$$

再减去105，直至得到23，即为所求结果。

我国古代数学家秦九韶将“孙子问题”的解答进行了推广形成了中国剩余定理，其解法运用了同余的性质。将其抽象为现在的数学问题，我们可以得到现代的中国剩余定理为：

设$m_1, m_2, ..., m_r$是两两互素的自然数，令$m = m_1 m_2 \dots m_r = m_i M_i$，即$M_i = m_1 \dots m_{i-1} m_{i+1} \dots m_r, i = 1, 2, \dots, r$, 则方程组

$$\begin{cases} x \equiv b_1 \pmod{m_1} \\ x \equiv b_2 \pmod{m_2} \\ \dots \\ x \equiv b_r \pmod{m_r} \end{cases} \tag{1}$$

的解为

$$x \equiv M_1' M_1 b_1 + M_2' M_2 b_2 + \dots + M_r' M_r b_r \pmod{m},$$

其中M_i'是整数，使

$$M_i'M_i \equiv 1 \pmod{m_i}, i = 1, 2, \cdots, r$$

该方程有且仅有一个小于m的非负整数解。

3.3.2 代码实现

中国剩余定理的主函数的代码如下：

```
def CRT(b_list,m_list):
  M=1
  for mi in m_list:
      M=M*mi
  #计算Mi
  Mi_list=[]
  for mi in m_list:
      Mi_list.append(M//mi)
  #计算的模逆元Mi
  Mi_inverse=[]
  for i in range(len(Mi_list)):
      Mi_inverse.append(get_inversr(Mi_list[i],m_list[i]))
  #计算解x
  x=0
  for i in range(len(b_list)):
      x+=Mi_list[i]*Mi_inverse[i]*b_list[i]
      x%=M
  return x
```

通过中国剩余定理求同余方程组的解的完整代码请见附录

3.3.3 结果展示

使用程序求解同余方程组

$$\begin{cases} x \equiv 2 \pmod{3} \\ x \equiv 3 \pmod{5} \\ x \equiv 2 \pmod{7} \end{cases}$$

的结果:

```
请输入bi，以空格分隔:2 3 2
请输入mi，以空格分隔:3 5 7
同余式方程组的解为23
```

异常输入展示：

```
请输入bi，以空格分隔:2 4 5
请输入mi，以空格分隔:2 3 5 7
mi的个数和bi的个数不相同，请重新输入

请输入bi，以空格分隔:2 5 7
请输入mi，以空格分隔:3 6 9
输入的mi并不是两两互质的，请重新输入mi
```

参考文献

[1] 牛淑芬,于斐,杨平平,方丽芝.交叉学科背景下信息安全数学基础理论与实践教学方法研究[J].计算机教育,2021(02):149-152.DOI:10.16512/j.cnki.jsjjy.2021.02.035.

[2] 李瑞琪,高敏芬,贾春福.信息安全数学基础的“讲一练二考三”改革方案设计[J].计算机教育,2016(11):27-30.DOI:10.16512/j.cnki.jsjjy.2016.11.007.

[3] 裴定一，徐祥，董军武.信息安全数学基础[M].北京:人民邮电出版社，2016

[4] 杜瑞庆,夏方林.埃拉托斯特尼筛法及改进(C++语言)[J].中国科技信息,2006(18):152-153+156.

[5] 张双红,高亚楠.论同余理论在生活中的应用[J].吉林省教育学院学报,2017,33(11):181-183.DOI:10.16083/j.cnki.1671-1580.2017.11.053.

附录

(1) 辗转相乘法求最大公因子完整代码

```
  def division_algorithm(m,n):
      while(n%m!=0):
          m,n=n%m,m
      return m

  a=int(input("请输入第一个整数:"))
  b=int(input("请输入第二个整数:"))
  gcd=0
  if(a<0):
      a=-a
  if(b<0):
      b=-b
  if(a==0 and b==0):
      gcd=0
  elif(a==0 or b==0):
      gcd=a+b
  else:
      gcd=division_algorithm(a,b)
  print("最大公因子为: ")
  print(gcd)
```

(2) 整数互素判断完整代码

```
def division_algorithm(m,n):
  while(n%m!=0):
      m,n=n%m,m
  return m

def Relatively_prime(a,b):
  if(a<0):
      a=-a
  if(b<0):
      b=-b
  if(a==0 and b==0):
      gcd=0
  elif(a==0 or b==0):
      gcd=a+b
  else:
      gcd=division_algorithm(a,b)
  if(gcd==1):
      return True
  else: return False

a=int(input("请输入第一个整数:"))
b=int(input("请输入第二个整数:"))

if(Relatively_prime(a,b)):
  print("{}和{}互素".format(a,b))
else:
  print("{}和{}不互素".format(a,b))
```

(3) Erarosthenes筛法寻找素数运行时间计算代码

```
from time import *
def sieve_func(n):
    prime=[]
    sieve =[True] *(n+1)
    for i in range(2, n):
        if sieve[i]:
            prime.append(i)
            for j in range(i *i, n, i):
                sieve[j] =False
    return prime

n=input("请输入素数的范围: n")
begin_time=time()
sieve_func(int(n))
end_time=time()
run_time=end_time-begin_time
print('运行时间: {}'.format(run_time))
```

(4) 中国剩余定理求解同余方程组完整代码

```
#扩展欧几里得算法的函数
def ext_euclid(a, b):
    if b ==0:
        return 1, 0, a
    else:
        x, y, q =ext_euclid(b, a % b)
        x, y =y, (x -(a //b) * y)
        return x, y, q

#运用扩展欧几里得算法求模逆元
def get_inversr(a,b):
    return ext_euclid(a,b)[0]

#中国剩余定理
def CRT(b_list,m_list):
    M=1
    for mi in m_list:
        M=M*mi
    #计算Mi
    Mi_list=[]
    for mi in m_list:
        Mi_list.append(M//mi)
    #计算的模逆元Mi
    Mi_inverse=[]
    for i in range(len(Mi_list)):
        Mi_inverse.append(get_inversr(Mi_list[i],m_list[i]))
    #计算解x
    x=0
    for i in range(len(b_list)):
        x+=Mi_list[i]*Mi_inverse[i]*b_list[i]
        x%=M
    return x

#判断列表中元素是否两两互素
def ifcoprime(ls):
```

```
  for i in range(len(ls)):
    for j in range(i+1,len(ls)):
      if ext_euclid(ls[i],ls[j])[2]!=1:
        return 0


#输入bi,mi
while True:
    m_list=[]
    b_list=[]

    b_i=input("请输入，以空格分隔bi:")
    b_i=b_i.split( )
    for i in b_i:
      b_list.append(int(i))

    m_i=input("请输入，以空格分隔mi:")
    m_i=m_i.split( )
    for i in m_i:
      m_list.append(int(i))

    if len(m_list)!=len(b_list):
      print("的个数和的个数不相同，请重新输入mibi\n")
    elif ifcoprime(m_list)==0:
      print("输入的并不是两两互质的，请重新输入mimi\n")
    else:
      break

x=CRT(b_list,m_list)
print("同余式方程组的解为{}".format(x))
```