**西安电子科技大学网信院**

**信息安全基础与密码学**

**综合实验**

**实 验 报 告（二）**

**中国剩余定理**

**班级：2118021**

**姓名：**

**学号：**

**日期：2023年10月29日**

一、实验目的

1. 实验环境

macOS Sonoma、Python 3.9

1. 实验目标

判断给定同余方程组能否使用中国剩余定理求解，若能，则通过中国剩余定理，解同余方程组。

二、方案设计

1. 背景

《孙子算经》中给出求解一次同余方程组的定理，即孙子定理的初步思想，经过学界认定，为首次提及该定理，因此这种求解同余方程组的方法也被称为中国剩余定理。通过中国剩余定理，可以较快地求出满足一定条件的依次同余方程组的解。

1. 原理

设正整数两两互素，则对于任意个整数，同余方程组

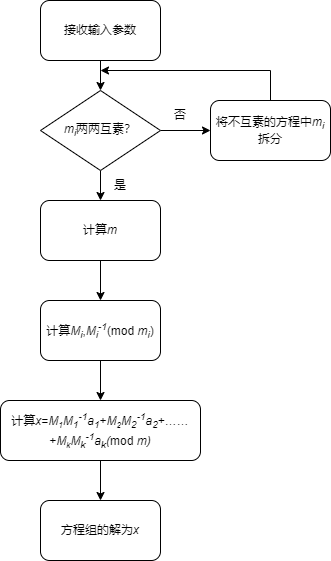
必有解，且模下解唯一。该解为：

当x的系数不为1时，必须通过模逆运算将系数化为1，才能使用中国剩余定理。

当不满足两两互素时，可以将及方程拆分。若拆分后出现冲突的方程，则该一次同余方程组无法用中国剩余定理求解。

三、方案实现

1. 算法流程图



1. 主要函数的介绍

* **class Equation**

用来表示方程组中每一个方程的与。

* **gcd(a, b)**

用欧几里得算法计算最大公因数

* **is\_relatively\_prime(eq: [Equation]) -> [int, int]**

判断方程组中是否两两互素

返回[-1,-1]表示两两互素，否则返回第一对发现的不互素的数据的下标

* **chinese\_remainder\_theorem(eq) -> [int, int]**

用中国剩余定理求解，返回解与模

1. 算法实现的主要代码
2. from rsa.common import inverse
3. class Equation:
4. a = 0
5. m = 0
6. def \_\_init\_\_(self, a, m):
7. self.a = a
8. self.m = m
9. def gcd(a, b):
10. """
11. 用欧几里得算法计算最大公因数
12. """
13. if a < b:
14. temp = a
15. a = b
16. b = temp
17. if a % b == 0:
18. return b
19. return gcd(b, a % b)
20. def is\_relatively\_prime(eq : [Equation]) -> [int, int]:
21. for i in range(len(eq)):
22. for j in range(i + 1, len(eq)):
23. if gcd(eq[i].m, eq[j].m) != 1:
24. return [i, j]
25. return [-1, -1]
26. def chinese\_remainder\_theorem(eq) -> [int, int]:
27. """
28. 中国剩余定理
29. return [x, m]
30. """
31. # 当mi不互素时，尝试分解使其互素
32. not\_relatively\_prime\_num = is\_relatively\_prime(eq)
33. while not\_relatively\_prime\_num != [-1, -1]:
34. gcd\_num = gcd(eq[not\_relatively\_prime\_num[0]].m, eq[not\_relatively\_prime\_num[1]].m)
35. # 存在不互素的mi时，先判断是否成倍数，成倍数则删去较小数，否则分解
36. if gcd\_num == eq[not\_relatively\_prime\_num[0]].m:
37. eq.remove(eq[not\_relatively\_prime\_num[0]])
38. elif gcd\_num == eq[not\_relatively\_prime\_num[1]].m:
39. eq.remove(eq[not\_relatively\_prime\_num[1]])
40. else:
41. # 中国剩余定理无法求解
42. if eq[not\_relatively\_prime\_num[0]].a % gcd\_num != eq[not\_relatively\_prime\_num[1]].a % gcd\_num:
43. return [-1, -1]
44. eq[not\_relatively\_prime\_num[0]].m = eq[not\_relatively\_prime\_num[0]].m / gcd\_num
45. eq[not\_relatively\_prime\_num[0]].a = eq[not\_relatively\_prime\_num[0]].a % eq[not\_relatively\_prime\_num[0]].m
46. eq[not\_relatively\_prime\_num[1]].m = eq[not\_relatively\_prime\_num[1]].m / gcd\_num
47. eq[not\_relatively\_prime\_num[1]].a = eq[not\_relatively\_prime\_num[1]].a % eq[not\_relatively\_prime\_num[1]].m
48. eq.append(Equation(eq[not\_relatively\_prime\_num[0]].a % gcd\_num, gcd\_num))
49. not\_relatively\_prime\_num = is\_relatively\_prime(eq)
50. M = 1
51. x = 0
52. for i in range(len(eq)):
53. M = M \* eq[i].m
54. for i in range(len(eq)):
55. mi = 1
56. for j in range(len(eq)):
57. if j != i:
58. mi = mi \* eq[j].m
59. x = x + mi \* inverse(mi, eq[i].m) \* eq[i].a
60. x = x % M
61. return [x, M]
62. def main():
63. equations1 = []
64. a1 = []
65. m1 = []
66. with open('1.txt', 'r') as f:
67. count = 0
68. for line in f:
69. count = count + 1
70. if count <= 3:
71. a1.append(int(line))
72. else:
73. m1.append(int(line))
74. f.close()
75. for i in range(len(a1)):
76. equations1.append(Equation(a1[i], m1[i]))
77. result = [chinese\_remainder\_theorem(equations1)]
78. equations2 = []
79. a2 = []
80. m2 = []
81. with open('2.txt', 'r') as f:
82. count = 0
83. for line in f:
84. count = count + 1
85. if count <= 3:
86. a2.append(int(line))
87. else:
88. m2.append(int(line))
89. f.close()
90. for i in range(len(a2)):
91. equations2.append(Equation(a2[i], m2[i]))
92. result.append(chinese\_remainder\_theorem(equations2))
93. equations3 = []
94. a3 = []
95. m3 = []
96. with open('3.txt', 'r') as f:
97. count = 0
98. for line in f:
99. count = count + 1
100. if count <= 3:
101. a3.append(int(line))
102. else:
103. m3.append(int(line))
104. f.close()
105. for i in range(len(a3)):
106. equations3.append(Equation(a3[i], m3[i]))
107. result.append(chinese\_remainder\_theorem(equations3))
108. equations4 = []
109. a4 = []
110. m4 = []
111. with open('4.txt', 'r') as f:
112. count = 0
113. for line in f:
114. count = count + 1
115. if count <= 3:
116. a4.append(int(line))
117. else:
118. m4.append(int(line))
119. f.close()
120. for i in range(len(a4)):
121. equations4.append(Equation(a4[i], m4[i]))
122. result.append(chinese\_remainder\_theorem(equations4))
123. if result[0] == [-1, -1]:
124. print("1.txt中一次同余方程由于mi不互素，无法用中国剩余定理求解")
125. else:
126. print("1.txt中一次同余方程的解为", result[0][0], sep='')
127. if result[1] == [-1, -1]:
128. print("2.txt中一次同余方程由于mi不互素，无法用中国剩余定理求解")
129. else:
130. print("2.txt中一次同余方程的解为", result[1][0], sep='')
131. if result[2] == [-1, -1]:
132. print("3.txt中一次同余方程由于mi不互素，无法用中国剩余定理求解")
133. else:
134. print("3.txt中一次同余方程的解为", result[2][0], sep='')
135. if result[3] == [-1, -1]:
136. print("4.txt中一次同余方程由于mi不互素，无法用中国剩余定理求解")
137. else:
138. print("4.txt中一次同余方程的解为", result[3][0], sep='')
139. if \_\_name\_\_ == '\_\_main\_\_':
140. main()

四、数据分析(包括算法测试数据的分析，运行结果截图等等)

1. 算法测试数据的分析

本次实验对所有测试样例进行测试。

第1个测试样例中的不互素，且前两个方程不能拆分为使得两两不互素的情况，因此第1个测试样例中方程组不能用中国剩余定理求解。

第2个测试样例情况和第1个样例相同。

第3个测试样例中的两两互素，可由中国剩余定理求出唯一解。

第4个测试样例情况和第3个样例相同。

1. 运行结果截图



五、思考与总结

1. 求一次同余方程组的解，若正整数𝒎𝟏，𝒎𝟐，…，𝒎𝒌不是两两互素，是否能直接用中国剩余定理求解？例如方程组，需要如何求解？

不能。中国剩余定理可以使用的条件时必须两两互素，不满足则不能使用其求解。

该方程组由于18与15不互素，可这两个方程中的拆分，且拆分前后最小公倍数需相同，得到

对新方程组求解即可得到原方程组的解。

1. 实验过程中还遇到了什么问题，如何解决的？通过该实验有何收获？

**问题与解决**

由于使用Python进行计算。Python中两整数的除法结果为浮点型数据，不能表示大整数，因此计算时，不能用除法计算，需将包含的数依次相乘，才能正确得到。

在表示方程的与时，若用两个数组表示，则在移动数据时易造成混乱。因此定义一个类表示方程中的系数，会更加不易出错。

从文件中读取数据时，读入的信息为字符型，需类型转换为整型，才能进行后续计算。

**收获**

通过本次实验，我更深入地理解了中国剩余定理的计算过程，从整体上对其有了更清晰的思路。