实 验 报 告

实 验 题 目	班　　级	姓　　名	学　　号	指导教师
欧几里得算法与素数检测	计科 2303 班	林烨洋	202318140126	程昊苏

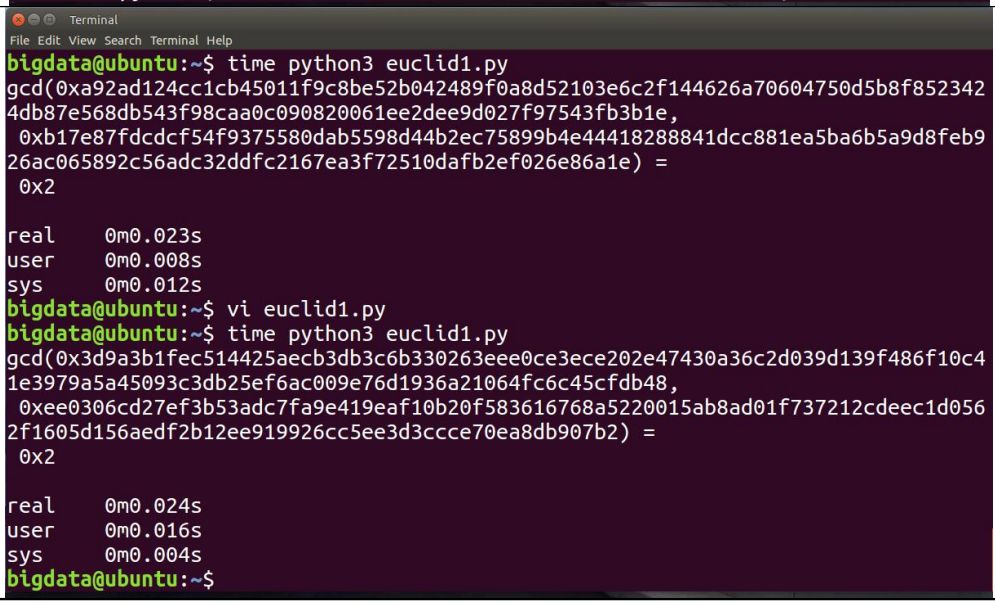
算法概述：

实验 1： 欧几里得算法-求最大公约数

欧几里得算法（Euclidean Algorithm）是一种用于求两个整数最大公约数（GCD）的高效方法。其基本思想是利用带余除法原理，通过不断用较小数去除较大数并取余，递归或循环地将问题规模缩小，直到余数为零为止，此时除数即为最大公约数。

1. 递归做法

➢ 代码及运行结果计时情况如下所示



➢ 运行结果说明

gcd(0x3d9a3b1fec514425aecb3db3c6b330263eee0ce3ece202e47430a36c2d039d139f486f10c41e3979a5a45093c3db25ef6ac009e76d1936a21064fc6c45cfdb48,

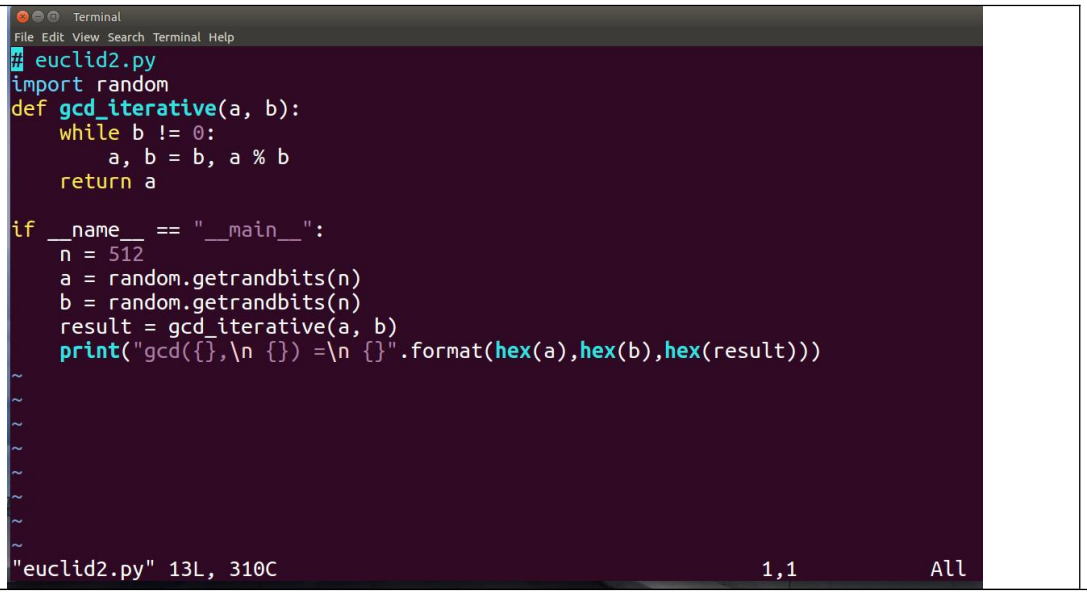
0xee0306cd27ef3b53adc7fa9e419eaf10b20f583616768a5220015ab8ad01f737212cdeec1d0562f1605d156aedf2b12ee919926cc5ee3d3ccce70ea8db907b2) =
0x2

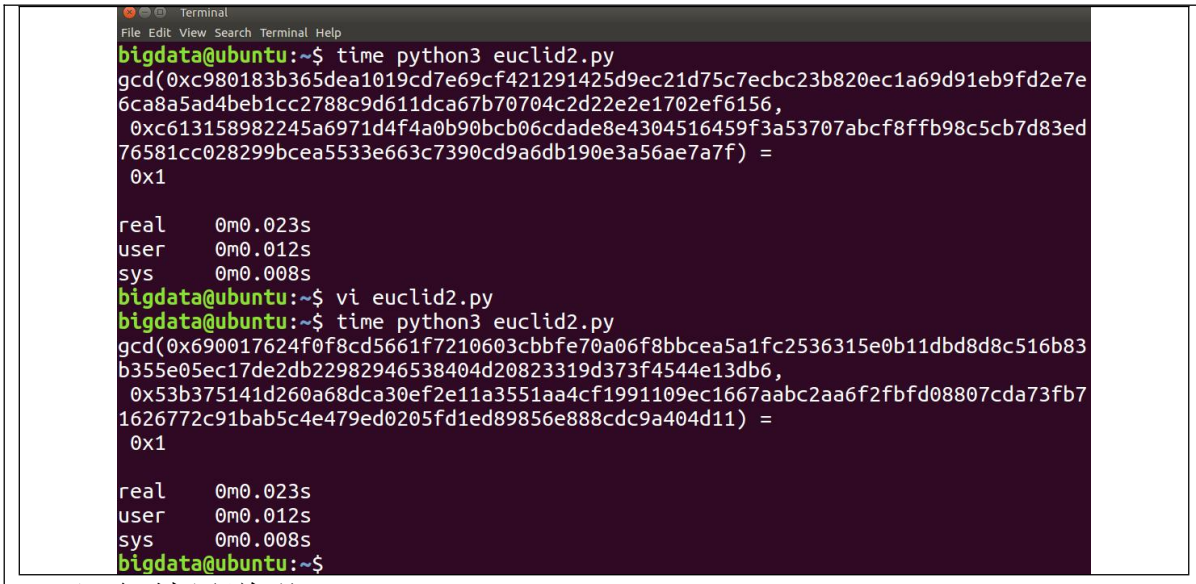
real 0m0.024s
user 0m0.016s
sys 0m0.004s

代表这两个 512bit 的数的最大公因数的 16 进制表示为 **0x2**
且运行时间分别为
real（实际时间） 0m0.024s
user（用户态时间） 0m0.016s
sys（内核时间） 0m0.004s

2. 非递归做法

➢ 代码及运行结果计时情况如下所示





➢ 运行结果说明

gcd(0x690017624f0f8cd5661f7210603cbbfe70a06f8bbcea5a1fc2536315e0b11dbd8d8c516b83b355e05ec17de2db22982946538404d20823319d373f4544e13db6,

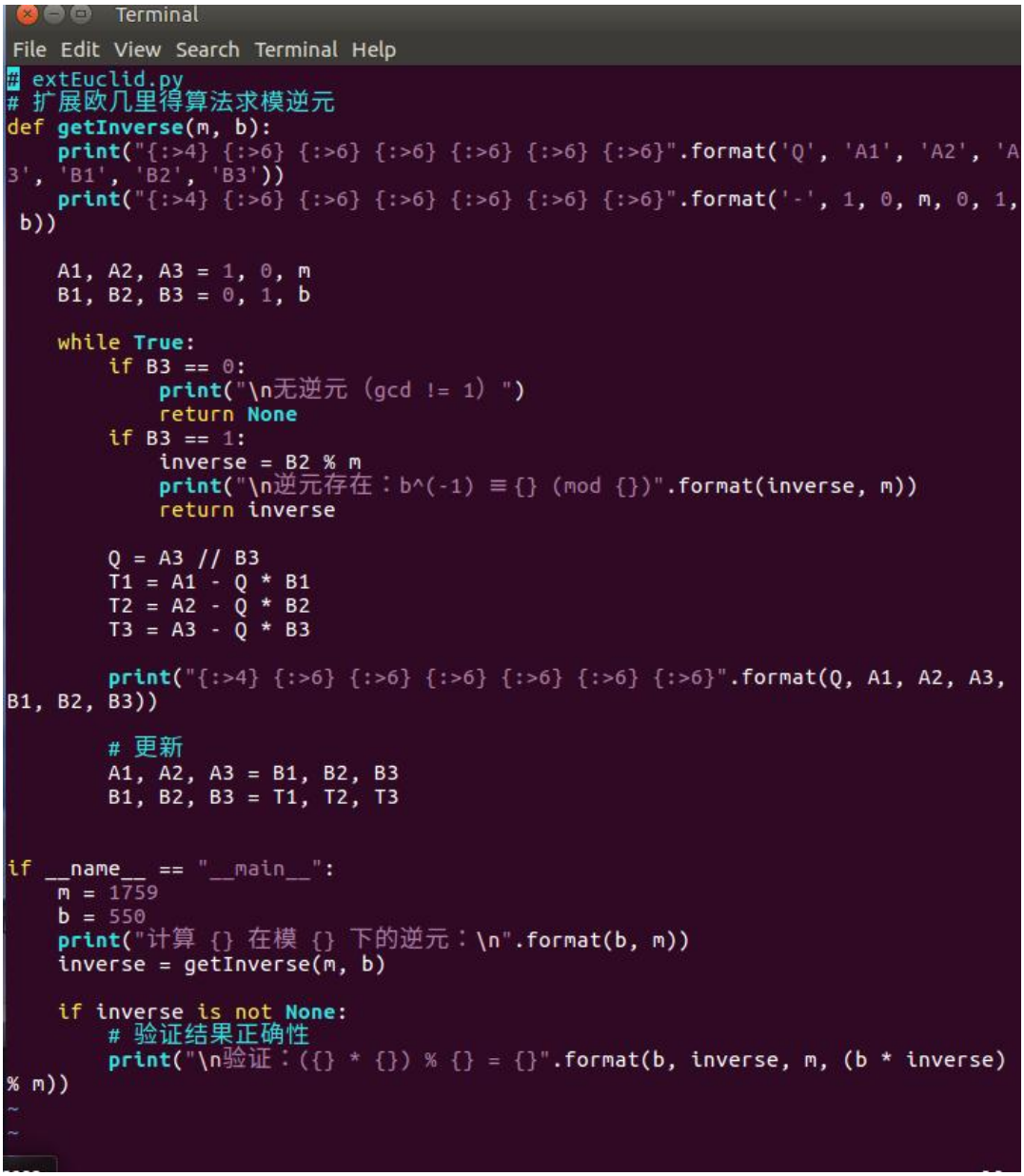
0x53b375141d260a68dca30ef2e11a3551aa4cf1991109ec1667aabc2aa6f2fbfd08807cda73fb71626772c91bab5c4e479ed0205fd1ed89856e888cdc9a404d11) =
0x1

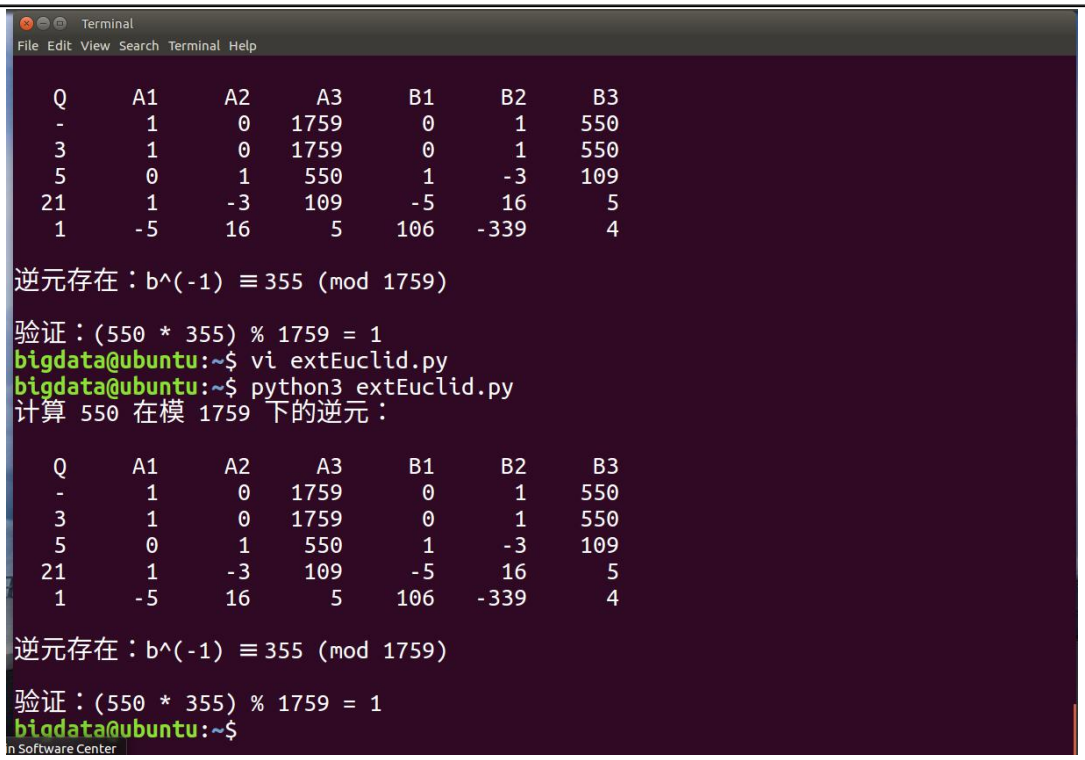
real 0m0.023s
user 0m0.012s
sys 0m0.008s

代表这两个 512bit 的数的最大公因数的 16 进制表示为 **0x1**
且运行时间分别为
real（实际时间） 0m0.023s
user（用户态时间） 0m0.012s
sys（内核时间） 0m0.008s

实验 2：欧几里得扩展算法 模运算求逆

扩展欧几里得算法是在普通欧几里得算法基础上发展而来，不仅能求出两个整数的最大公约数，还能同时求得满足 $sa + tb = \gcd(a,b)$的整数 s、t。当 a 与 b 互素时，该算法可用于求解模逆元 $a^{-1} \bmod m$。其核心思想是通过不断的带余除法与线性组合迭代更新，逐步得到逆元的系数，算法高效且实现简单。


Terminal
File Edit View Search Terminal Help
# extEuclid.py
# 扩展欧几里得算法求模逆元
def getInverse(m, b):
print("{:>4} {:>6} {:>6} {:>6} {:>6} {:>6} {:>6}".format('Q', 'A1', 'A2', 'A3', 'B1', 'B2', 'B3'))
print("{:>4} {:>6} {:>6} {:>6} {:>6} {:>6} {:>6}".format('-', 1, 0, m, 0, 1, b))
A1, A2, A3 = 1, 0, m
B1, B2, B3 = 0, 1, b
while True:
if B3 == 0:
print("\n无逆元 (gcd != 1) ")
return None
if B3 == 1:
inverse = B2 % m
print("\n逆元存在：b^(-1) ≡ {} (mod {})".format(inverse, m))
return inverse
Q = A3 // B3
T1 = A1 - Q * B1
T2 = A2 - Q * B2
T3 = A3 - Q * B3
print("{:>4} {:>6} {:>6} {:>6} {:>6} {:>6} {:>6}".format(Q, A1, A2, A3, B1, B2, B3))
# 更新
A1, A2, A3 = B1, B2, B3
B1, B2, B3 = T1, T2, T3
if __name__ == "__main__":
m = 1759
b = 550
print("计算 {} 在模 {} 下的逆元：\n".format(b, m))
inverse = getInverse(m, b)
if inverse is not None:
# 验证结果正确性
print("\n验证：({} * {}) % {} = {}".format(b, inverse, m, (b * inverse) % m))



➢ 运行结果说明

此结果展示扩展欧几里得算法求 550 模 1759 逆元过程。

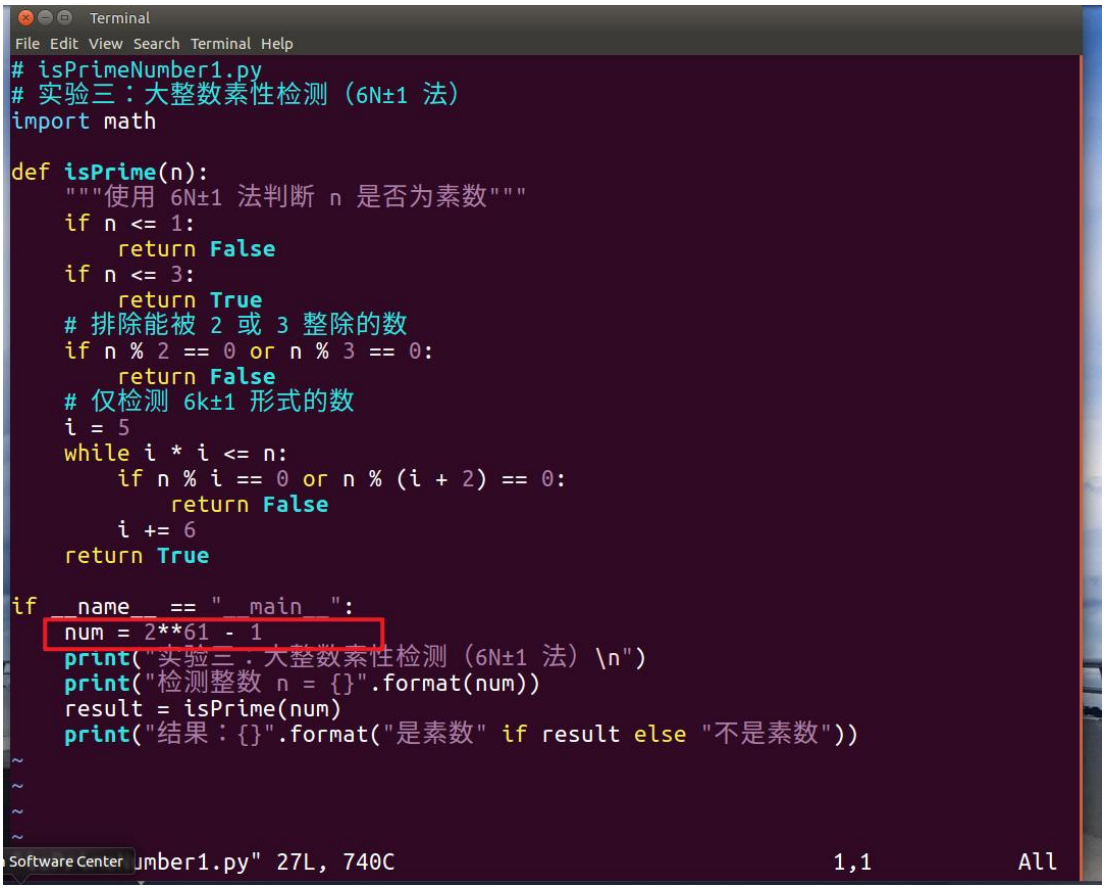
表格列示每步商 Q 及系数 A1-A3、B1-B3，体现迭代计算。

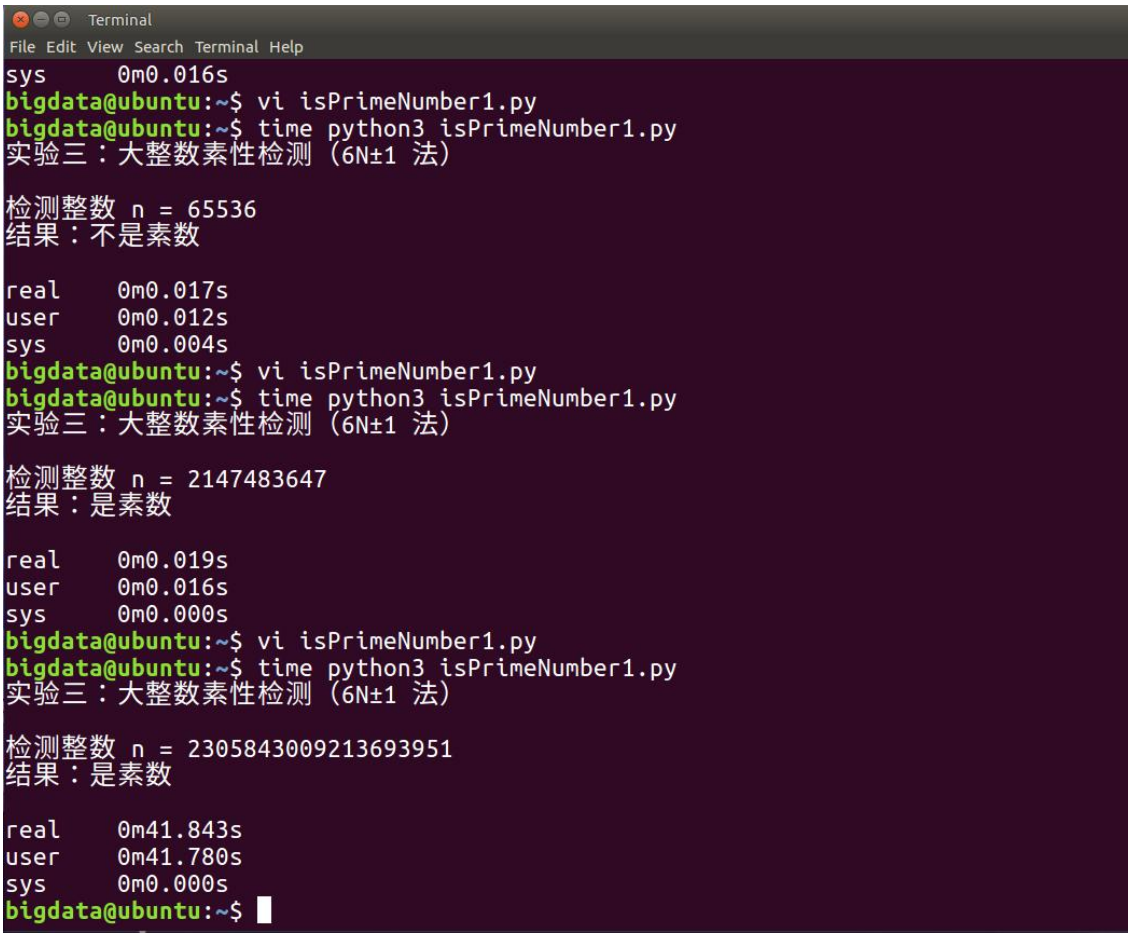
最终得逆元 355，验证 550×355 mod1759=1，符合逆元定义（乘积模 m 为 1）。

实验 3：大整数素性检测

6N±1 素性检测法基于素数分布规律：除 2 和 3 外，所有素数均可表示为 6N−1 或 6N+1 的形式。算法首先排除能被 2 和 3 整除的数，然后仅对形如 6N±1 的数进行整除性检测，显著减少计算次数，提高检测效率。该方法简单高效，适用于中等规模整数的素性判断。

➢ **代码及运行结果计时情况如下所示**


Terminal
File Edit View Search Terminal Help
# isPrimeNumber1.py
# 实验三：大整数素性检测（6N±1 法）
import math
def isPrime(n):
"""使用 6N±1 法判断 n 是否为素数"""
if n <= 1:
return False
if n <= 3:
return True
# 排除能被 2 或 3 整除的数
if n % 2 == 0 or n % 3 == 0:
return False
# 仅检测 6k±1 形式的数
i = 5
while i * i <= n:
if n % i == 0 or n % (i + 2) == 0:
return False
i += 6
return True
if __name__ == "__main__":
num = 2**61 - 1
print("实验三：大整数素性检测（6N±1 法）\n")
print("检测整数 n = {}".format(num))
result = isPrime(num)
print("结果：{}".format("是素数" if result else "不是素数"))
Software Center
1,1
All


Terminal
File Edit View Search Terminal Help
sys 0m0.016s
bigdata@ubuntu:~$ vi isPrimeNumber1.py
bigdata@ubuntu:~$ time python3 isPrimeNumber1.py
实验三：大整数素性检测（6N±1 法）
检测整数 n = 65536
结果：不是素数
real 0m0.017s
user 0m0.012s
sys 0m0.004s
bigdata@ubuntu:~$ vi isPrimeNumber1.py
bigdata@ubuntu:~$ time python3 isPrimeNumber1.py
实验三：大整数素性检测（6N±1 法）
检测整数 n = 2147483647
结果：是素数
real 0m0.019s
user 0m0.016s
sys 0m0.000s
bigdata@ubuntu:~$ vi isPrimeNumber1.py
bigdata@ubuntu:~$ time python3 isPrimeNumber1.py
实验三：大整数素性检测（6N±1 法）
检测整数 n = 2305843009213693951
结果：是素数
real 0m41.843s
user 0m41.780s
sys 0m0.000s
bigdata@ubuntu:~$

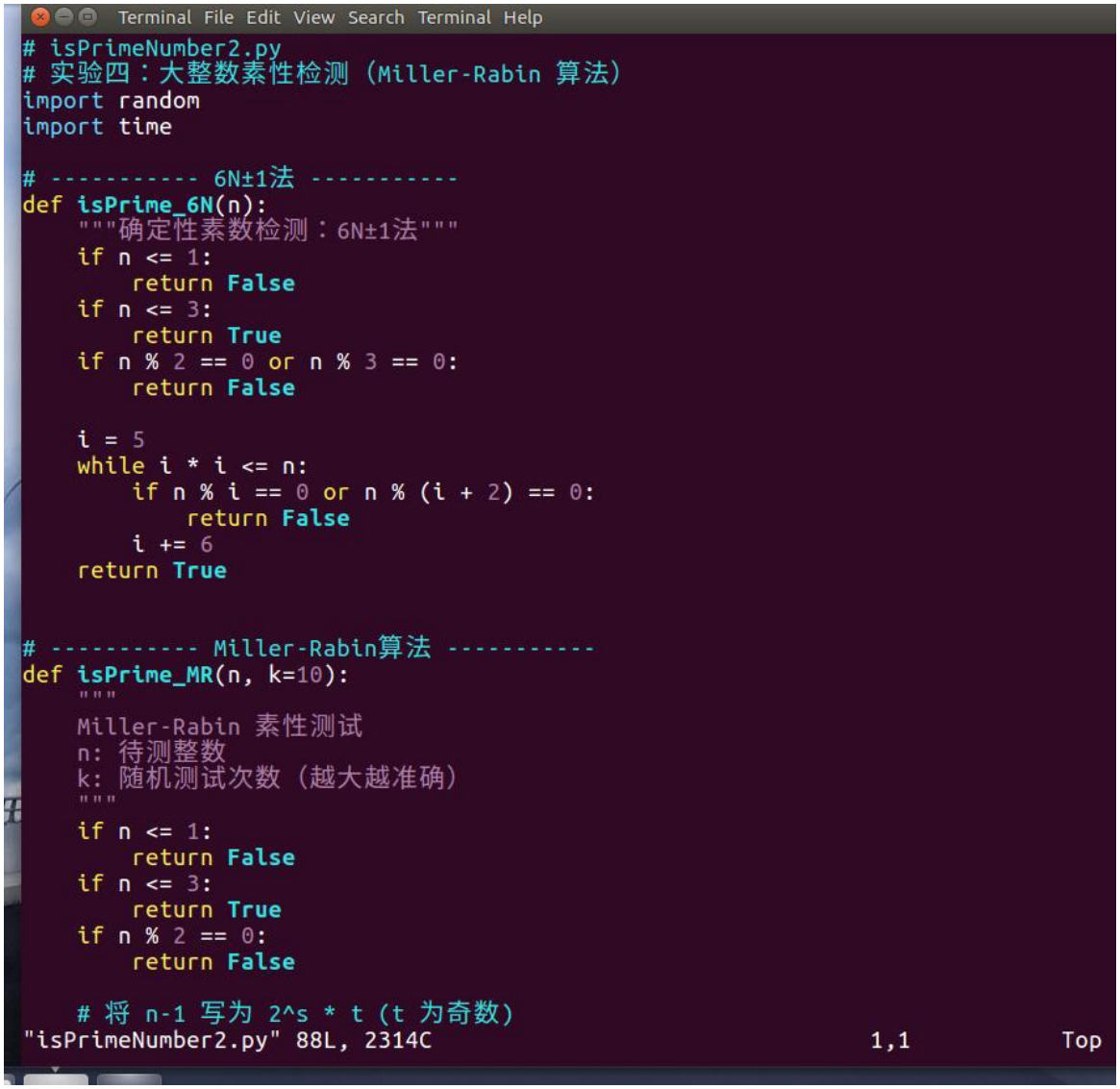
如代码所示，只需要修改 num 的值即可依次看三个数是否是质数并且分别计算三个数的素性检测的运行时间

该结果展示了 6N±1 法对三个大整数的素性检测：2^16（65536）因能被 2 整除判定为非素数，耗时仅 0.017 秒；2^31-1 和 2^61-1 均判定为素数，前者耗时 0.019 秒，后者因数值极大耗时 41.843 秒，体现出大整数检测的时间成本随位数显著增加。

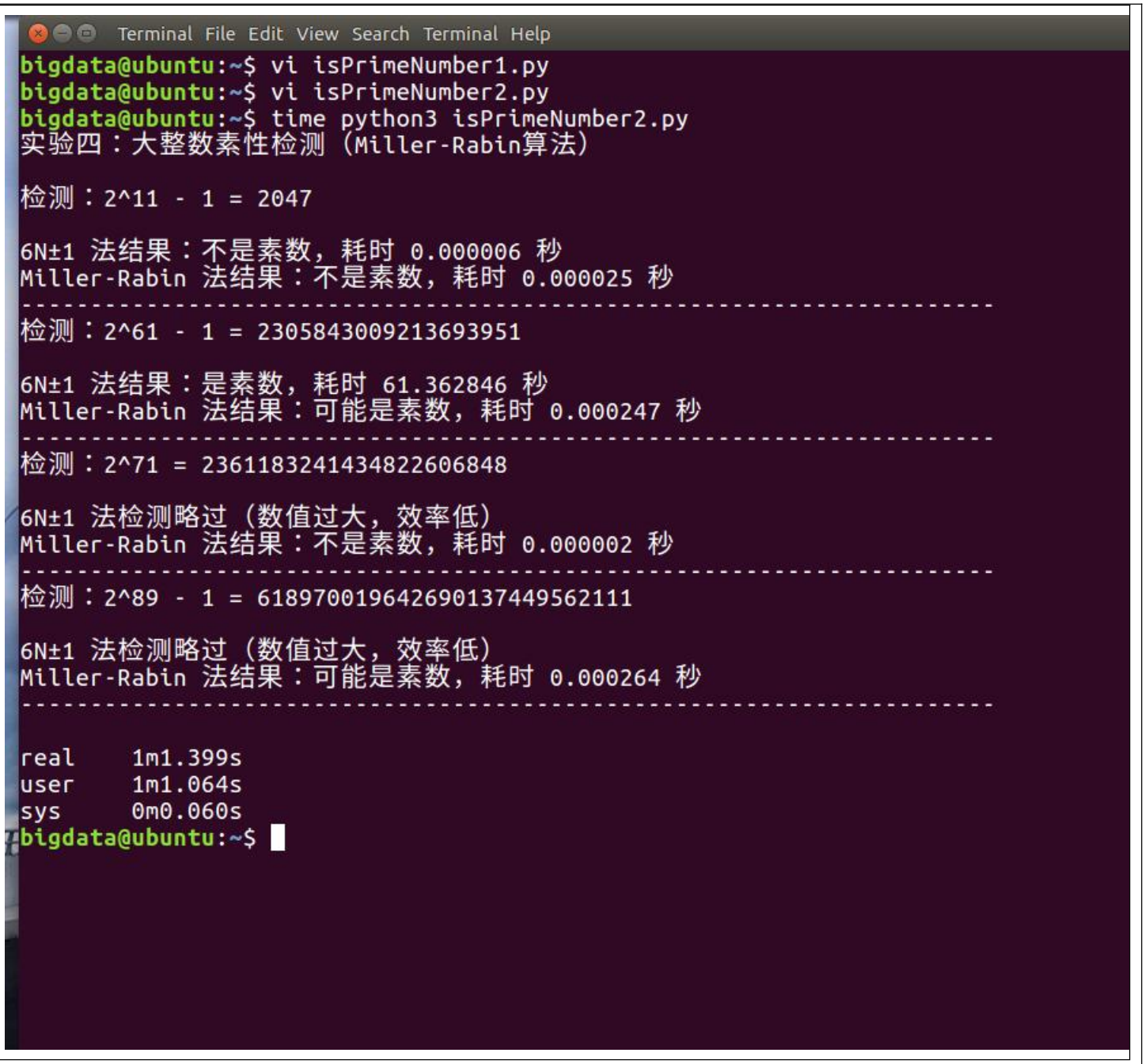
思考题：更高效的素性检测算法

该程序实现两种素性检测算法。6N±1 法为确定性算法，基于素数（除 2、3 外）必形如 6k±1 的特性，通过检验 n 是否能被 5 至$\sqrt{n}$ 间 6k±1 形式的数整除判断素性，适用于小数。Miller-Rabin 算法为概率性测试，先将 n-1 分解为 $2^s t$，再随机选 k 个基底 a，检验 $a^t \bmod n$ 是否为 1 或 n-1，及后续平方结果是否为 n-1，通过多次测试降低误判率，效率远高于 6N±1 法，适合大整数检测。

部分代码展示如下


Terminal File Edit View Search Terminal Help
# isPrimeNumber2.py
# 实验四：大整数素性检测（Miller-Rabin 算法）
import random
import time
# ----------- 6N±1法 -----------
def isPrime_6N(n):
"""确定性素数检测：6N±1法"""
if n <= 1:
return False
if n <= 3:
return True
if n % 2 == 0 or n % 3 == 0:
return False
i = 5
while i * i <= n:
if n % i == 0 or n % (i + 2) == 0:
return False
i += 6
return True
# ----------- Miller-Rabin算法 -----------
def isPrime_MR(n, k=10):
"""
Miller-Rabin 素性测试
n: 待测整数
k: 随机测试次数（越大越准确）
"""
if n <= 1:
return False
if n <= 3:
return True
if n % 2 == 0:
return False
将 n-1 写为 2^s * t (t 为奇数)
"isPrimeNumber2.py" 88L, 2314C
1,1
Top

运行结果如下



备注：

2025 年 10 月 16 日