离散对数求解 实验报告

学号：SA20225172 姓名：郭俊勇

实现目的

- 掌握与密码学相关的基础数论知识；
- 利用中间相遇攻击来编程实现离散对数的求解。

编程语言

- Python

实现内容

- 计算模素数p的离散对数
- 令p是一个素数，g是有限乘法群Z_p^*上的一个原根，然后给定一个Z_p^*上的h满足$h = g^x$，其中$1 \le x \le 240$，目的是找到x。也就是说，编写的程序以p, g, h作为输入，然后输出x。
- 该问题最直接的算法就是对x的2^{40}个可能的值逐个进行尝试，直到找到正确的一个，即直到找到一个x在Z_p上满足$h = g^x$。这需要2^{40}次乘法运算。在本次实验中，需要实现一个算法，该算法使用中间相遇攻击，时间代价约为$\sqrt{2^{40}} = 2^{20}$。
- 令$B = 2^{20}$。因为x是小于B^2，我们可以将未知的x写作$x = x_0B + x_1$，其中$x0, x_1 \in [0, B-1]$。然后，$h = g^x = g^{x_0B+x_1} = (g^B)^{x_0} \times g_{x_1}$（在$Z_p$上）。两边同时除以$g_{x_1}$，可得到$h/g^{x_1} = (g^B)^{x_0}$（在$Z_p$上）。
- 上面等式中的变量是x_0和x_1，其他都是已知的：g和h是给定的，$b = 2^{20}$。由于x_0和x_1在等式的两边，所以我们可以使用中间相遇攻击来找到一个解：
  - 为等式左边h/g^{x_1}的所有可能值创建一个哈希表，其中$x_1 = 0, 1, \cdots, 2^{20}$。
  - 对于每一个$x_0 = 0, 1, \cdots, 2^{20}$，检查$(g^B)^{x_0}$是否在哈希表中，如果是，便找到了解$(x0, x1)$，即$x = x_0B + x_1$。
  - 总体工作大约是2^{20}次乘法来构建表，另外2^{20}次查找在此表中。
  - 当完成求解程序之后，请以附件test.txt中的p, g, h为输入，求解出x。

实验原理分析

- 利用中间相遇的方式减小运算的时间复杂度。
- 基础版：按照实验要求设置x=x0*B+x1构造。
- 改进版：将x=x0*B+x1改变成x=x0*B-x1,这样可以节省计算1/g^x1的时间，大大提高算法的运行时间。在本质上两个表达式无差别。

Python代码实现（源码）

基础版

```
import gmpy2
import time

# 初始参数
p =
134078079299425970995740249982058461274793658205923933777235614437217640300735469768018742981669034276900318581864860508537538828119465699464336490060840171
```

```
g =
11717829880366207009516117596335367088558084999998952205599979459063929499736583
746670572176471460312928594829675428279466566527115212748467589894601965568
h =
32394751040504504435652643787280657886490975209524495278347924529719819761432925
58073856937958553180532878928001494706097394108577585732452307673444020333
B = 2 ** 20  # 1048576

# 创建空字典
hashmap = {}

# 记录开始的时间
start = time.process_time()

# 处理下x1
# h mod p
m0 = gmpy2.f_mod(h, p)
for x1 in range(B):
    # g^x1 mod p
    m1 = gmpy2.powmod(g, x1, p)
    # 1/g^x1
    m2 = gmpy2.invert(m1, p)
    # h*1/g^x1 mod p
    m3 = gmpy2.f_mod(m0 * m2, p)
    # 记录x1 和 h*1/g^x1 mod p
    hashmap[m3] = x1

# g^B mod p
right = gmpy2.powmod(g, B, p)

for x0 in range(B):
    # g^B^x0 mod p
    c = gmpy2.powmod(right, x0, p)
    # 判断是否在右边的hashmap中
    if c in hashmap:
        # x = x0*B - x1
        x = gmpy2.mul(B, x0) - hashmap[c]
        print("x0=", x0)
        print("x1=", hashmap[c])
        print("x= ", x)
        break
# 记录结束时间
end = time.process_time()
print("The running time totals is", end - start, "s")
```

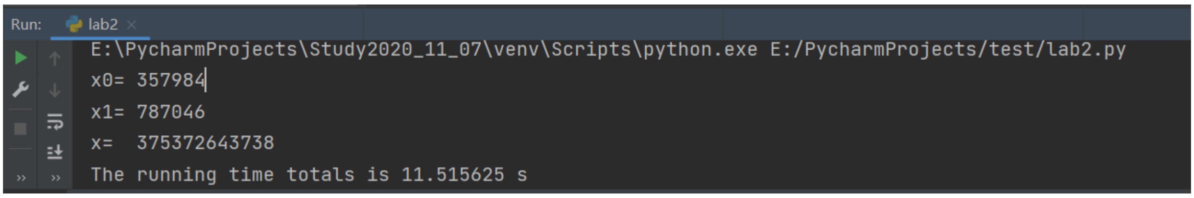
改进版

```
import gmpy2
import time

# 初始参数
p =
13407807929942597099574024998205846127479365820592393377723561443721764030073546
976801874298166903427690031858186486050853753882811946569946433649006084171
g =
11717829880366207009516117596335367088558084999998952205599979459063929499736583
746670572176471460312928594829675428279466566527115212748467589894601965568
```

```python
h =
32394751040504504435652643787280657886490975209524495278347924529719819761432925
5807385693795855318053287892800149470609739410857758573245230767344020333
B = 2 ** 20  # 1048576

# 创建空字典
hashMap = {}

# 优化后，令x=Bx0-x1
start = time.process_time()
s0 = gmpy2.f_mod(h, p)
for x1 in range(1048577):
    # g^x1 mod p
    s1 = gmpy2.powmod(g, x1, p)
    # h*g^x1 mod p
    s2 = gmpy2.f_mod(s1 * s0, p)
    hashMap[s2] = x1

r = gmpy2.powmod(g, B, p)
for x0 in range(len(hashMap)):
    c = gmpy2.powmod(r, x0, p)
    if c in hashMap:
        x = gmpy2.mul(B, x0) - hashMap[c]
        print("x0=", x0)
        print("x1=", hashMap[c])
        print("x=", x)
        break
end = time.process_time()
print("The running time totals is", end - start, "s")
```

运行结果

基础版



改进版