SHANDONG UNIVERSITY

密码工程第五次实验报告

姓名: 谢子洋
学院: 网络空间安全学院 (研究院)
专业: 网络空间安全
学号:202100460116

2023 年 11 月 29 日

目录

1 实验原理

1.1 Mul.

表 1: 乘法

算法	时间复杂度
normal	$\mathscr{O}(n^2)$
Karatsuba Algorithm	$\mathscr{O}(n^{\log_2^3}) \approx n^{1.585}$
Fürer’s algorithm	$\mathscr{O}(n \log n\, 2^{O(\log^* n)})$
David Harvey	$\mathscr{O}(n \log n)$
FFT multiplication algorithm	$\mathscr{O}(n \log n \log\log n)$

1.1.1 Karatsuba Algorithm.

给定十进制数 X 和 Y, 其 10 进制长度分别为 a 和 b.

设 m 为 a,b 中最小值, 以此为乘数的划分长度, 使用分治算法将 X 和 Y 都分为两段. X 高位部分记作 A,X 低位部分记作 B. 类似的,Y 的高位部分和低位部分分别记作 C 和 D.

那么有

$$\begin{aligned} X &= A \cdot 10^m + B \\ Y &= C \cdot 10^m + D \\ XY &= AC \cdot 10^{2m} + (AD + BC)10^m + BD \end{aligned} \tag{1.1}$$

计算 XY 可以先计算 AC,BD 和 $(AD+BC)$, 将问题划分为三个子问题.

首先直接计算 AC 和 BD. 利用子问题之间的依赖关系, 可将计算 $(AD+BC)$ 转化为:

$$AD + BC = (A+B)\cdot(D+C) - AC - BD$$

伪代码如下:

Algorithm 1 KaratsubaMul

Input: X,Y
Output: (XY)

```
m := min{a, b}
A, B := split(X, m)
C, D := split(Y, m)
Z2 := KaratsubaMul(A, C)
Z1 := KaratsubaMul(A + B, C + D)
Z0 := KaratsubaMul(B, D)
return Z2 · 10^(2m) + (Z1 − Z2 − Z0) · 10^m + Z0
```

1.2 Mod

1.2.1 Barrett Reduction.

计算 $a\ mod\ n$. 若 $s=\frac{1}{n}$, 则 $a\ mod\ n = a-\lfloor as\rfloor n$.

Barrett 模算法使用 $\frac{m}{2^k}$ 逼近 $\frac{1}{n}$. 给定 k, 令

$$\frac{m}{2^k}=\frac{1}{n}\Leftrightarrow m=\frac{2^k}{n}$$

因此 $m=\lfloor\frac{2^k}{n}\rfloor$ 最为准确. 据此我们可以计算

$$a\ mod\ n = a-\lfloor a\cdot\frac{m}{2^k}\rfloor n$$

算法伪代码如下:

Algorithm 2 Barrett Reduction

Input: a,n,k
Output: a

```
m ← ⌊2^k/n⌋
q ← (a × m) >> k
a ← a − q × n
if a ≥ n then
    a ← a − n
end if
```

真实值和估计值之间的误差 $e=\frac{1}{n}-\frac{m}{2^k}$.

要求 $a\cdot e<1$, 即

$$a<\frac{n\cdot 2^k}{2^k-n\cdot\lfloor 2^k/n\rfloor}$$

Barrett 模算法将复杂的运算转化为简单的移位运算, 因此比之一般模运算能提升一定效率.

1.3 ModMul.

1.3.1 Montgomery Reduction Algorithm.

算法详细过程.

1) 转换为蒙哥马利形式
 找到一个与 N 互素的模数 R, 可知:

$$\begin{aligned} aR+bR&=(a+b)R\ mod\ N.\\ aR-bR&=(a-b)R\ mod\ N.\end{aligned}\tag{1.2}$$

可计算

$$\overline{a} \leftarrow aR \; modN. \qquad \overline{b} \leftarrow bR \; modN. \tag{1.3}$$

2) 计算蒙哥马利形式下乘积

计算 R^{-1} 满足 $RR^{-1} \equiv 1 \; mod \; N$.

计算 $R^{'}$ 满足 $RR^{'} \equiv -1 \; mod \; N$.

计算 $m \leftarrow (\overline{T} \; mod \; R)N^{'} \; mod \; R$.

则可计算 $\overline{c} \leftarrow (\overline{T} + mN)/R$

3) 转换回原形式

$$c = \overline{c}R^{-1} \; mod \; N. \tag{1.4}$$

算法伪代码如下:

Algorithm 3 Montgomery

Input: a, b, N, R
Output: c

1: $\overline{a} \leftarrow aR \; mod \; N.\overline{b} \leftarrow bR \; mod \; N.$
2: $R^{-1} \leftarrow xgcd(R, N, 1)$
3: $R^{'} \leftarrow xgcd(R, N, N-1)$
4: $N^{-1} \leftarrow xgcd(N, R, 1)$
5: $N^{'} \leftarrow xgcd(N, R, R-1)$
6: $\overline{T} \leftarrow \overline{x} \cdot \overline{y}$
7: $m \leftarrow (\overline{T} \; mod \; R)N^{'} \; mod \; R$
8: $\overline{c} \leftarrow (\overline{T} + mN)/R$
9: **if** $\overline{c} \geq N$ **then**
10: $\quad \overline{c} \leftarrow \overline{c} - N$
11: **end if**
12: $c \leftarrow \overline{c}R^{-1} \; mod \; N.$

2 实验过程

2.1 广义欧几里得算法及模逆运算

1. 欧几里得算法

辗转相除法，又称欧几里得算法, 是求最大公约数的算法。给定两数 a 和 b 求其最大公约数, 两数不停相减直到其中一个数变为 0, 此时另一个数即为最大公约数.

```python
def gcd(a,b):
  while(True):
      if(a>b):  a=a%b
      elif(a<b):b=b%a
      if(a==0 or b==0):break
  return a if b==0 else b
```

2. 拓展欧几里得算法

扩展欧几里得算法是欧几里得算法的扩展。在已知整数 a、b 情况下, 扩展欧几里得算法可以在求得 a、b 的最大公约数的同时, 找到整数 x、y（其中一个可能是负数），使它们满足等式.

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

实现上可以使用矩阵法

$$\begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \end{pmatrix} \xrightarrow{\text{行变换}} \begin{pmatrix} x & y & gcd(a,b) \\ * & * & 0 \end{pmatrix}$$

```
def xgcd(a,b,muti=1):
  tempa=a
  tempb=b
  mat=[[1,0],[0,1]]
  loop=1
  while(loop):
      if(a>b):
          k=a//b
          a=a%b
          mat[0][0]-=k*mat[1][0]
          mat[0][1]-=k*mat[1][1]
      elif(a<b):
          k=b//a
          b=b%a
          mat[1][0]-=k*mat[0][0]
          mat[1][1]-=k*mat[0][1]
      if(b==0 ):
          loop=0
          result=[a,mat[0][0],mat[0][1]]
      elif(a==0):
          loop=0
          result=[b,mat[1][0],mat[1][1]]
  #print(f"{result[0]}={result[1]} * {tempa}+{result[2]} * {tempb}")
  if(muti!=1 and muti%result[0]==0):
      Times=muti//result[0]
      result=[item*Times for item in result]
      #print(f"{result[0]}={result[1]} * {tempa}+{result[2]} * {tempb}")
  return result
```

3. 模逆运算

模逆运算可转换为拓展欧几里得算法: 若 p 在模 N 下存在逆 $p^{-1}\ mod\ N$, 则有

$$p^{-1}\cdot p+k\cdot N=1$$

可使用拓展欧几里得算法计算 p^{-1}.

```
def inverse_mod(a,n):
  if(gcd(a,n)!=1):
      return None
  a_inv=xgcd(a,n)[1]
```

```
return a_inv
```

2.2 乘运算 Karatsuba 算法

给定十进制数 X 和 Y, 使用分治算法将 X 和 Y 都分为两段.

$$\begin{aligned} X &= A \cdot 10^m + B \\ Y &= C \cdot 10^m + D \\ XY &= AC \cdot 10^{2m} + (AD+BC)10^m + BD \end{aligned} \tag{2.1}$$

利用子问题之间的依赖关系, 将 $(AD+BC)$ 转化为:

$$AD+BC = (A+B)\cdot(D+C) - AC - BD$$

最终计算:

$$XY = AC\cdot 10^{2m} + [(A+B)\cdot(D+C) - AC - BD]10^m + BD$$

```
def MUL_karatsuba(num1,num2):
  def size_base10(num):
      return math.floor(math.log10(num))+1
  def split_at(num,pos):
      if(pos<0):return None
      return (num//(10**pos),num%(10**pos))
  if(num1<10 or num2<10): return num1*num2
  m=min(size_base10(num1),size_base10(num2))
  m2=m//2
  high1,low1=split_at(num1,m2)
  high2,low2=split_at(num2,m2)
  z0=MUL_karatsuba(low1,low2)
  z1=MUL_karatsuba((low1+high1),(low2+high2))
  z2=MUL_karatsuba(high1,high2)
  result = z2       *10**(m2*2)
  result+=(z1-z2-z0)*10**(m2)
  result+= z0
  return result
```

2.3 模运算 Barrett 算法

使用 Barrett 算法需要预先给定参数 k, 参数 k 的选择影响了能够进行模运算的最大值. 欲计算 $a \bmod n$, 参数 a 必须满足

$$a < \frac{n\cdot 2^k}{2^k - n\cdot\lfloor 2^k/n\rfloor}$$

否则,Barrett 算法可能不成立. 因此本文在实现中先检查参数是否满足条件, 其次才进行实际运算.

算法实现如下.

```python
def check(a,n,k):
  exp2_k=2**k
  bound=abs((n*exp2_k)//(exp2_k-n*(exp2_k//n)))
  return False if(a≥bound) else True
def Mod(a,n,k):
  if(not check(a,n,k)): return None
  m=2**k//n
  q=(a*m)>>k
  a-=q*n
  return a-n if a≥n else a
```

2.4 模乘运算 Montgomery 算法

生成参数 R 后, 将乘数转换为蒙哥马利形式. 蒙哥马利形式下, 两元素的乘法在计算上效率更高. 得到蒙哥马利形式的乘积后, 再使用参数 R 转化为一般形式, 得到乘积结果.

```python
def MulMod(a,b,N,k=10000):
  #0. 选取R
  R=number.getPrime(int(math.log2(N)))
  while(R>N or gcd(R,N)!=1):
      R=number.getPrime(int(math.log2(N)))
  #R=2**(int(math.log2(N))-1)
  #1. 转换
  a_=Mod((a*R),N,k)
  b_=Mod((b*R),N,k)
  #2. 计算
  R_inv=inverse_mod(R,N)
  N_d=xgcd(N,R,R-1)[1]%R
  T_=a_*b_
  m=Mod(Mod(T_,R,k)*N_d,R,k)
  c_=(T_+m*N)//R
  if(c_≥N):c_-=N
  #3. 转换
  c=Mod(c_*R_inv,N,k)
  return c
```

3 实验结果

设定大数如下:

x 取值为:

0d409785221820172445520800307998497665029407471704964127335943917845149239101714713859390210118936175035537479020070370695528976936265916588849033572220337998093087303913800076982044441820561523707583970238357708796211918231428211403395982049187398716124337198929190627533118104190251922366545100731803405095376485418179516178291355907751953999438194705962002302991988600954283680359182950637417514423593538991580893572932409469958003809465514898978236030841306432933135362540881529871704158568455487484827611105500839707722562683448359033815263592

4731380759090353784791250208940440992656344482556759866601089415604‌79

y 取值为:

0d112681325193434719424704563932292847871220674312425253906958453895714072‌4
1366027178949892024843331065490011040000546490680744558710859470582080481888889
9914569784426345948408455709607496904870854981901822471432991561322887055356978
9989607857051688353799501239743727392939051630525941620507860951622985059818276
1525569479285635058583880323259073363519274716721116550273765947611720905216797
2340133620294509887180526634707357286243948781780776410169803648918987845110064
8538023064815154417225919832593419354807307583373730533999979822770396404163252
749973747202836031351958673350403845497634605247744234052608326640710‌1

模 N 取值为:

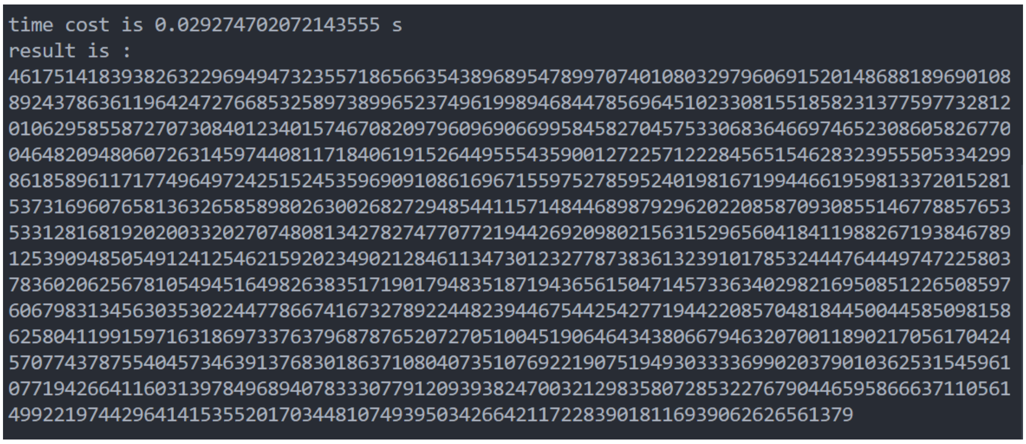
0d1516010406449491065924226074803688313273271066729359959467969706490667648
9273092653888993286457588764810314306835799721254337590993031215661893884819834
5639272056953746696138585489747867302950616954521417460849119786369246378032873
9941291661802399330476606112028613922364303979862482639018908016205638422193108
4010663631735040171061181835036040655615370355543859211859670087779557545039404
8936718165260491345409503736852470395097916754132293411306120631933725198463649
3821486719644147892286231311798016719698839838897385575929535038311579124842374
370083144994670172268726481959813398074702300533685168698315534167‌59

3.1 Karatsuba Mul

使用 Karatsuba 算法计算 $x \cdot y$, 耗时 0.02927s, 得到结果如下:

0d46175141839382632296949473235571865663543896895478997074010803297960691‌52
0148688189690108892437863611964247276685325897389965237496199894684478569645102
3308155185823137759773281201062958558727073084012340157467082097960969066995845
8270457533068364669746523086058267700464820948060726314597440811718406191526449
5554359001272257122284565154628323955505334299861858961171774964972425152453596
9091086169671559752785952401981671994466195981337201528153731696076581363265858
9802630026827294854411571484468987929620220858709308551467788576535331281681920
2003320270748081342782747707721944269209802156315296560418411988267193846789125
3909485054912412546215920234902128461134730123277873836132391017853244476444974
7225803783602062567810549451649826383517190179483518719436561504714573363402982
1695085122650859760679831345630353022447786674167327892244823944675442542771944
2208570481844500445850981586258041199159716318697337637968787652072705100451906
4643438066794632070011890217056170424570774378755404573463913768301863710804073
5107692219075194930333699020379010362531545961077194266411603139784968940783333
0779120939382470032129835807285322767904465958666371105614992219744296414153552
017034481074939503426642117228390181169390626265613‌79

经检验为正确结果.



```
time cost is 0.029274702072143555 s
result is :
46175141839382632296949473235571865663543896895478997074010803297960691520148688189690108
89243786361196424727668532589738996523749619989468447856964510233081551858231377597732812
```

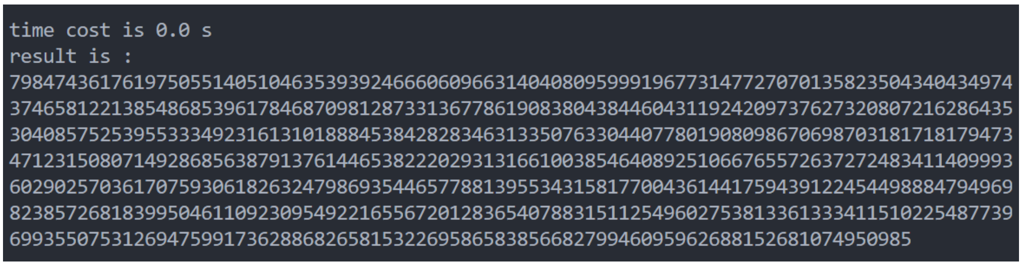
图 1: 乘法

3.2 Barrett Mod

令 $c = x \cdot y$, 使用 Barrett 算法计算 $c \ mod \ N$, 耗时 0s(极短时间). 得到结果为:

0d7984743617619750551405104635393924666060966314040809599919677314772707013
5823504340434974374658122138548685396178468709812873313677861908380438446043119
2420973762732080721628643530408575253955333492316131018884538428283463133507633
0440778019080986706987031817181794734712315080714928685638791376144653822202931
3166100385464089251066765572637272483411409993602902570361707593061826324798693
5446577881395534315817700436144175943912245449888479496982385726818399504611092
3095492216556720128365407883151125496027538133613334115102254877396993550753126
9475991736288682658153226958658385668279946095962688152681074950985

经检验为正确结果.



```
time cost is 0.0 s
result is :
79847436176197505514051046353939246660609663140408095999196773147727070135823504340434974
37465812213854868539617846870981287331367786190838043844604311924209737627320807216286435
30408575253955333492316131018884538428283463133507633044077801908098670698703181718179473
47123150807149286856387913761446538222029313166100385464089251066765572637272483411409993
60290257036170759306182632479869354465778813955343158177004361441759439122454498884794969
82385726818399504611092309549221655672012836540788315112549602753813361333411510225487739
69935507531269475991736288682658153226958658385668279946095962688152681074950985
```

图 2: 模运算

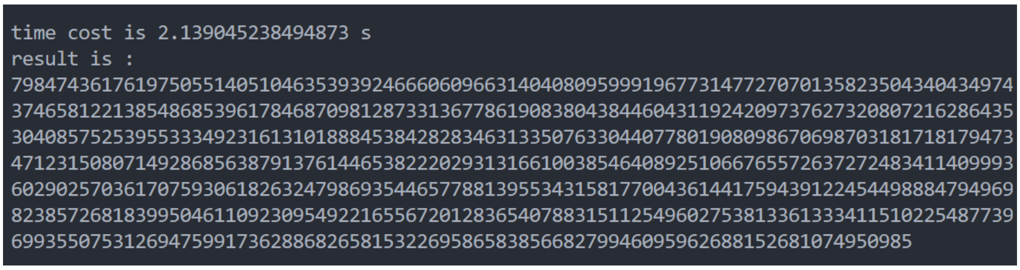
3.3 Montgomery MulMod

使用 Montgomery 算法计算 $x \times y \ mod \ N$, 耗时 2.13 秒, 得到结果为:

0d7984743617619750551405104635393924666060966314040809599919677314772707013
5823504340434974374658122138548685396178468709812873313677861908380438446043119
2420973762732080721628643530408575253955333492316131018884538428283463133507633
0440778019080986706987031817181794734712315080714928685638791376144653822202931
3166100385464089251066765572637272483411409993602902570361707593061826324798693

5446577881395534315817700436144175943912245449888479496982385726818399504611092
3095492216556720128365407883151125496027538133613334115102254877396993550753126
9475991736288682658153226958658385668279946095962688152681074950985

经检验为正确结果.



```
time cost is 2.139045238494873 s
result is :
79847436176197505514051046353939246660609663140408095999196773147727070135823504340434974
37465812213854868539617846870981287331367786190838043844604311924209737627320807216286435
30408575253955333492316131018884538428283463133507633044077801908098670698703181718179473
47123150807149286856387913761446538222029313166100385464089251066765572637272483411409993
60290257036170759306182632479869354465778813955343158177004361441759439122454498884794969
82385726818399504611092309549221655672012836540788315112549602753813361333411510225487739
69935507531269475991736288682658153226958658385668279946095962688152681074950985
```

图 3: 模乘

参考文献

[1] https://en.wikipedia.org/wiki/Karatsuba_algorithm.

A Code

```
from Crypto.Util import number
import math
import gmpy2
import time
def gcd(a,b):
    while(True):
        if(a>b):  a=a%b
        elif(a<b):b=b%a
        if(a==0 or b==0):break
    return a if b==0 else b
def xgcd(a,b,muti=1):
    mat=[[1,0],[0,1]]
    loop=1
    while(loop):
        if(a>b):
            k=a//b
            a=a%b
            mat[0][0]-=k*mat[1][0]
            mat[0][1]-=k*mat[1][1]
        elif(a<b):
            k=b//a
            b=b%a
            mat[1][0]-=k*mat[0][0]
            mat[1][1]-=k*mat[0][1]
        if(b==0 ):
            loop=0
            result=[a,mat[0][0],mat[0][1]]
        elif(a==0):
            loop=0
            result=[b,mat[1][0],mat[1][1]]
    #print(f"{result[0]}={result[1]} * {tempa}+{result[2]} * {tempb}")
    if(muti!=1 and muti%result[0]==0):
        Times=muti//result[0]
        result=[item*Times for item in result]
        #print(f"{result[0]}={result[1]} * {tempa}+{result[2]} * {tempb}")
    return result

def inverse_mod(a,n):
    if(gcd(a,n)!=1):
        return None
    a_inv=xgcd(a,n)[1]
    return a_inv

def MUL_karatsuba(num1,num2):
    def size_base10(num):
        return math.floor(math.log10(num))+1
    def split_at(num,pos):
        if(pos<0):return None
        return (num//(10**pos),num%(10**pos))

    if(num1<10 or num2<10): return num1*num2
    m=min(size_base10(num1),size_base10(num2))
    m2=m//2
    high1,low1=split_at(num1,m2)
    high2,low2=split_at(num2,m2)
```

```

    z0=MUL_karatsuba(low1,low2)
    z1=MUL_karatsuba((low1+high1),(low2+high2))
    z2=MUL_karatsuba(high1,high2)
    result = z2       *10**(m2*2)
    result+=(z1-z2-z0)*10**(m2)
    result+= z0
    return result

def Mod(a,n,k):
    def check(a,n,k):
        exp2_k=2**k
        bound=abs((n*exp2_k)//(exp2_k-n*(exp2_k//n)))
        return False if(a≥bound) else True
    if(not check(a,n,k)): return None
    m=2**k//n
    q=(a*m)>>k
    a-=q*n
    return a-n if a≥n else a

def MulMod(a,b,N,k=10000):
    #0. 选取R
    R=number.getPrime(int(math.log2(N)))
    while(R>N or gcd(R,N)!=1):
        R=number.getPrime(int(math.log2(N)))
    #R=2**(int(math.log2(N))-1)
    #1. 转换
    a_=Mod((a*R),N,k)
    b_=Mod((b*R),N,k)
    #2. 计算
    R_inv=inverse_mod(R,N)
    N_d=xgcd(N,R,R-1)[1]%R
    T_=a_*b_
    m=Mod(Mod(T_,R,k)*N_d,R,k)
    c_=(T_+m*N)//R
    if(c_≥N):c_-=N
    #3. 转换
    c=Mod(c_*R_inv,N,k)
    return c

# ----------- 以下均为函数测试 -------------------
k=189281
x='4,097,852,218,201,724,455,208,003,079,984,976,650,294,074,717,049,641,273,359,439,178,451,492,391,017,147,
y='11,268,132,519,343,471,942,470,456,393,229,284,787,122,067,431,242,525,390,695,845,389,571,407,241,366,027
N='151,601,040,644,949,106,592,422,607,480,368,831,327,327,106,672,935,995,946,796,970,649,066,764,892,730,92

x=int(x.replace(',',''))
y=int(y.replace(',',''))
N=int(N.replace(',',''))
def testMul():
    sT=time.time()
    c=MUL_karatsuba(x,y)
    eT=time.time()
    print(f"\ntime cost is {eT-sT} s")
    print(f"result is :\n{c}\n")
def testMod():
    sT=time.time()
    c=Mod(x*y,N,6000)
    eT=time.time()
```

```
    print(f"\ntime cost is {eT-sT} s")
    print(f"result is :\n{c}\n")

def testMulMod():
    sT=time.time()
    c=MulMod(x,y,N,k)
    eT=time.time()
    print(f"\ntime cost is {eT-sT} s")
    print(f"result is :\n{c}\n")
#testMul()
#testMod()
testMulMod()
```