密码学实验报告 2

张天辰 17377321

2019 年 3 月 12 日

1 素性检验算法

1.1 Miller-Rabin 算法

$n \geqslant 3$ 的奇数都可以表示为 $n-1 = 2^k q$。因此，有如下算法：若 n 为素数，则剩余类数列 $a^q, a^{2q}, \cdots, a^{2^{k-1}q}, a^{2^k q}$ 中，要么第一个数模 n 余 1，要么数列中某个数模 n 余 -1，要么 n 为合数。然而就算条件满足也不能确保 n 为素数，因此通过取随机数 a 可以大大减小 n 为合数的概率。

1.2 Fermat 算法

此算法基于 Fermat 小定理。随机选择整数 $b \in (0, n)$，求 $gcd(b, n)$。如果 $d > 1$ 则 n 不是素数；如果 $n = 1$ 则判断 $b^{n-1} \pmod n$ 是否成立，如果不成立则为合数，否则为素数概率小于 $\frac{1}{2}$。重复 t 次则 n 为素数概率为 $1 - \frac{1}{2^t}$。

1.3 Solovay-Stassen 算法

此算法基于 Jacobi 符号和欧拉准则设计。随机选择整数 $b \in [2, n-2]$，计算 $r = b^{\frac{n-1}{2}} \pmod n$。如果 $r \neq 1$ 且 $r \neq -1$ 则根据 Legendre 符号，n 是合数。再计算 Jacobi 符号 $s = J(b, n)$。如果 $r \neq s$ 则不符合欧拉准则，n 为合数。上述过程重复 t 次可以缩小 n 为合数的概率。

1.4 测试样例

```
Input the number to be tested: 2305843009213693951
Input the test times: 100
MillerRabin:
True
Runtime is 0.00843s.
Fermat:
True
Runtime is 0.00730s.
SolovayStassen:
True
Runtime is 0.00839s.
```

图 1: 素性检验

1.5 算法实现

Algorithm 1 Miller-Rabin 算法

输入: 待测试数 n，测试次数 $times$

输出: n 为合数，或者很可能为素数

```
function MillerRabin(n, times)
    power ← 0
    rest ← n − 1
    while rest%2 == 0 do
        rest ← rest >> 1
        power ← power + 1
    end while
    for each i ∈ [1, times] do
        rand ← RandomInt ∈ [2, n − 2]
        if rand^rest (mod n) ∈ 1, n − 1 then
            continue
        end if
        actPower ← rest
        for each j ∈ [0, power − 2] do
            actPower ← actPower ∗ 2
            if rand^actPower (mod n) == n − 1 then
                break
            end if
        end for
        if loop does not break then
            return False
        end if
    end for
    return True
end function
```

Algorithm 2 Fermat 算法

输入: 待测试数 n，测试次数 $times$

输出: n 为合数，或者很可能为素数

```
function Fermat(n, times)
    for each i ∈ [1, times] do
        base ← RandomInt ∈ [1, n − 1]
        gcd ← gcd(n, base)
        if gcd > 1 then
            return False
```

```
        end if
        if base^{n-1} (mod n) ≠ 1 then
            return False
        end if
    end for
    return True
end function
```

Algorithm 3 Solovay-Stassen 算法

输入: 待测试数 n，测试次数 $times$

输出: n 为合数，或者很可能为素数

```
function JACOBI(a, b)
    result = 1
    a = a%b
    if b == 1 then
        return result
    end if
    while a ≠ 0 do
        while a%2 == 0 do
            a ← a/2
            if b%8 ∈ 3, 5 then
                result ← −result
            end if
        end while
        a, b ← b, a
        if a%4 == b%4 == 3 then
            result ← −result
        end if
        a ← a%b
    end while
    return result
end function
function SOLOVAYSTASSEN(n, times)
    for each i ∈ [1, times] do
        base ← RandomInt ∈ [2, n − 2]
        test1 ← base^{(n-1)/2} (mod n)
        if test1not ≡ ±1 then
            return False
        end if
```

$test2 \leftarrow$ JACOBI$(base, n)$
if $test1 \neq test2$ **then**
return $False$
end if
end for
return $True$
end function

1.6 算法比较

RM 算法时间复杂度为 $O(t \log \log n)$，Fermat 算法时间复杂度为 $O(t \log n)$，SS 算法时间复杂度为 $O(t \log n + t \log n) = O(t \log n)$。在 t 很大时，三个算法的错误几率都极小。

2 *GF* 域上的四则运算

2.1 $GF(2^8)$

给定 $GF(2)$ 上的 8 次不可约多项式 $x^8 + x^4 + x^3 + x + 1$，可求得其生成元为 $x + 1$。于是在 $GF(2^8)$ 上，加法（减法）利用按位异或，乘法、除法、取逆都是使用打表的方式更加快捷，且不会消耗很多空间（3 个长度为 256 的整数表）。本算法同时给出了移位异或的乘法方案，可以看出代码的繁琐。

Algorithm 4 $GF(2^8)$ 四则运算

输入: 二进制代表的两个多项式 $num1, num2$
输出: $num1, num2$ 四则运算结果
function ADD$(num1, num2)$
$num1, num2 \rightarrow int$
return $num1 \oplus num2 \rightarrow str$
end function
function MULTIPLY$(num1, num2)$
$num2 \leftarrow num2$ 反转
$result \leftarrow 0$
$temp \leftarrow int(num1)$
for each $i \in [1, 8]$ **do**
if $num2[i] == 1$ **then**
$result \leftarrow result \oplus temp$
end if
if temp $[0] == 1$ **then**
$temp \leftarrow (temp << 1)[1:] \oplus 0x1B$
else
$temp \leftarrow (temp << 1)[1:]$
end if

```
    end for
    return result → str
end function
multi ← [1]
for each i ∈ [1, 254] do
    multi[i] ← multi[i − 1] << 1 ⊕ multi[i − 1]
    if multi[i]&0x100 then
        multi[i] ← multi[i] ⊕ 0x11B
    end if
end for
arcMulti ← multi 的反表
reverse ← [0]
for each i ∈ [1, 255] do
    reverse[i] ← multi[(255 − arcMulti[i])%255]
end for
function tableMultiply(num1, num2)
    return multi[(arcMulti[num1] + arcMulti[num2])% 255]
end function
function tableRev(num)
    return reverse[num]
end function
function divide(num1, num2)
    temp ← tableRev(num2)
    return tableMultiply(elm1, temp)
end function
```

2.2 $GF(2^8)$ 演示

2.3 测试样例

2.4 $GF(2^4)$

与 $GF(2^4)$ 情况相仿，选取 4 次不可约多项式为 x^4+x^3+1，其生成元为 x^2。同样使用打表的方式获得乘法、除法、逆的结果，也给出了移位乘法的算法。

Algorithm 5 $GF(2^4)$ 四则运算

输入: 二进制代表的两个多项式 $num1, num2$

输出: $num1, num2$ 四则运算结果

```
function add(num1, num2)
    num1, num2 → int
    return num1 ⊕ num2 → str
```

```
10110101
11000010
10110101 + 11000010 = 01110111
10110101 * 11000010 = 10001100
10110101 * 11000010 = 10001100
10110101 / 11000010 = 10101000
10110101 ^ -1 = 01110101
11000010 ^ -1 = 00101111
```

图 2: $GF(2^8)$

end function
function MULTIPLY($num1, num2$)
  $num2 \leftarrow num2$ 反转
  $result \leftarrow 0$
  $temp \leftarrow int(num1)$
  for each $i \in [1, 4]$ **do**
    if $num2[i] == 1$ **then**
      $result \leftarrow result \oplus temp$
    end if
    if temp $[0] == 1$ **then**
      $temp \leftarrow (temp << 1)[1 :] \oplus 0x9$
    else
      $temp \leftarrow (temp << 1)[1 :]$
    end if
  end for
  return $result \rightarrow str$
end function
$multi \leftarrow [1]$
for each $i \in [1, 14]$ **do**
  $multi[i] \leftarrow multi[i-1] << 1$
  if $multi[i]\&0x10$ **then**
    $multi[i] \leftarrow multi[i] \oplus 0x19$
  end if
  $multi[i] \leftarrow multi[i] << 1$
  if $multi[i]\&0x10$ **then**
    $multi[i] \leftarrow multi[i] \oplus 0x19$
  end if
end for

$arcMulti \leftarrow multi$ 的反表
$reverse \leftarrow [0]$
for each $i \in [1, 15]$ **do**
    $reverse[i] \leftarrow multi[(15 - arcMulti[i])\%15]$
end for
function TABLEMULTIPLY($num1, num2$)
    return multi[(arcMulti[num1] + arcMulti[num2])% 15]
end function
function TABLEREV(num)
    return $reverse[num]$
end function
function DIVIDE($num1, num2$)
    $temp \leftarrow$ TABLEREV($num2$)
    return TABLEMULTIPLY($elm1, temp$)
end function

2.5 $GF(2^4)$ 演示

2.6 测试样例

```
1001
0101
1001 + 0101 = 1100
1001 * 0101 = 0110
1001 * 0101 = 0110
1001 / 0101 = 1010
1001 ^ -1 = 1101
0101 ^ -1 = 1111
```

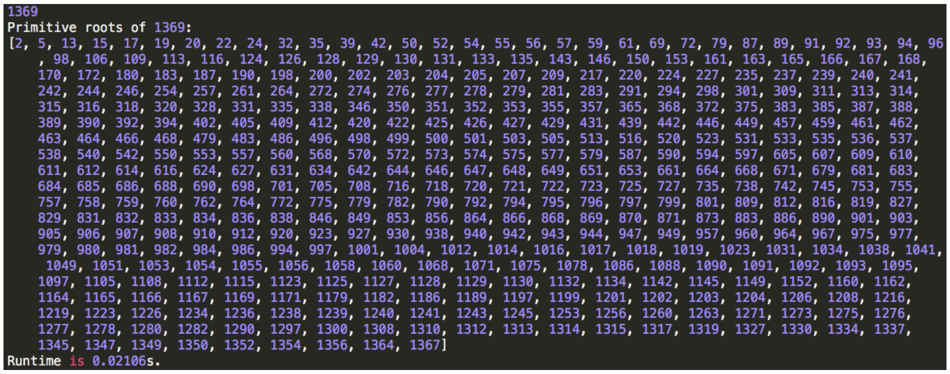
图 3: $GF(2^4)$

3 原根生成算法

3.1 算法简介

先求得 Euler 函数 $\phi(n)$，再对 n 进行素因子分解。对每个即将测试的 g，只要所有的 $g^{\frac{\phi(n)}{p}} \neq 1 \pmod n$，且 $g^{\phi(n)} \equiv 1 \pmod n$ 就说明 g 是原根。

3.2 测试样例



```
1369
Primitive roots of 1369:
[2, 5, 13, 15, 17, 19, 20, 22, 24, 32, 35, 39, 42, 50, 52, 54, 55, 56, 57, 59, 61, 69, 72, 79, 87, 89, 91, 92, 93, 94, 96
    , 98, 106, 109, 113, 116, 124, 126, 128, 129, 130, 131, 133, 135, 143, 146, 150, 153, 161, 163, 165, 166, 167, 168,
    170, 172, 180, 183, 187, 190, 198, 200, 202, 203, 204, 205, 207, 209, 217, 220, 224, 227, 235, 237, 239, 240, 241,
    242, 244, 246, 254, 257, 261, 264, 272, 274, 276, 277, 278, 279, 281, 283, 291, 294, 298, 301, 309, 311, 313, 314,
    315, 316, 318, 320, 328, 331, 335, 338, 346, 350, 351, 352, 353, 355, 357, 365, 368, 372, 375, 383, 385, 387, 388,
    389, 390, 392, 394, 402, 405, 409, 412, 420, 422, 425, 426, 427, 429, 431, 439, 442, 446, 449, 457, 459, 461, 462,
    463, 464, 466, 468, 479, 483, 486, 496, 498, 499, 500, 501, 503, 505, 513, 516, 520, 523, 531, 533, 535, 536, 537,
    538, 540, 542, 550, 553, 557, 560, 568, 570, 572, 573, 574, 575, 577, 579, 587, 590, 594, 597, 605, 607, 609, 610,
    611, 612, 614, 616, 624, 627, 631, 634, 642, 644, 646, 647, 648, 649, 651, 653, 661, 664, 668, 671, 679, 681, 683,
    684, 685, 686, 688, 690, 698, 701, 705, 708, 716, 718, 720, 721, 722, 723, 725, 727, 735, 738, 742, 745, 753, 755,
    757, 758, 759, 760, 762, 764, 772, 775, 779, 782, 790, 792, 794, 795, 796, 797, 799, 801, 809, 812, 816, 819, 827,
    829, 831, 832, 833, 834, 836, 838, 846, 849, 853, 856, 864, 866, 868, 869, 870, 871, 873, 883, 886, 890, 901, 903,
    905, 906, 907, 908, 910, 912, 920, 923, 927, 930, 938, 940, 942, 943, 944, 947, 949, 957, 960, 964, 967, 975, 977,
    979, 980, 981, 982, 984, 986, 994, 997, 1001, 1004, 1012, 1014, 1016, 1017, 1018, 1019, 1023, 1031, 1034, 1038, 1041,
     1049, 1051, 1053, 1054, 1055, 1056, 1058, 1060, 1068, 1071, 1075, 1078, 1086, 1088, 1090, 1091, 1092, 1093, 1095,
    1097, 1105, 1108, 1112, 1115, 1123, 1125, 1127, 1128, 1129, 1130, 1132, 1134, 1142, 1145, 1149, 1152, 1160, 1162,
    1164, 1165, 1166, 1167, 1169, 1171, 1179, 1182, 1186, 1189, 1197, 1199, 1201, 1202, 1203, 1204, 1206, 1208, 1216,
    1219, 1223, 1226, 1234, 1236, 1238, 1239, 1240, 1241, 1243, 1245, 1253, 1256, 1260, 1263, 1271, 1273, 1275, 1276,
    1277, 1278, 1280, 1282, 1290, 1297, 1300, 1308, 1310, 1312, 1313, 1314, 1315, 1317, 1319, 1327, 1330, 1334, 1337,
    1345, 1347, 1349, 1350, 1352, 1354, 1356, 1364, 1367]
Runtime is 0.02106s.
```

图 4: 原根

3.3 算法实现

Algorithm 6 计算 Euler 函数

输入: n

输出: n 的 Euler 函数值

function PHI(n)
  $result \leftarrow n$
  $i \leftarrow 2$
  while $i^2 \leqslant n$ **do**
    if $n\%i == 0$ **then**
      while $n\%i == 0$ **do**
        $n \leftarrow n/i$
      end while
      $result \leftarrow result - result/i$
    end if
    $i \leftarrow i + 1$
  end while
  if n > 1 **then**
    $result \leftarrow result - result/n$
  end if
  return $result$
end function

Algorithm 7 原根生成算法

输入: n

输出: n 的原根

function PRIMITIVE(n)

```
    roots, factors ← []
    if n == 2 then
        return 1
    end if
    euler ←PHI(n)
    temp ← euler
    for each i ∈ [2, √n] do
        if n%i == 0 then
            factors.append(i)
            while temp%i == 0 do
                temp ← temp/i
            end while
        end if
    end for
    if n > 1 then
        factors.append(n)
    end if
    for each g ∈ [2, n] do
        for each p ∈ factors do
            if g^(euler/p) (mod n)== 1 or g^euler (mod n)≠ 1 then
                break
            end if
            if loop does not break then
                roots.append(g)
            end if
        end for
    end for
    return roots
end function
```

4 本原多项式的生成

4.1 简介

唯一分解环上系数均互素的多项式为本原多项式。**由于本实验根本没有说明具体在哪个唯一分解环上如何生成次数为几的本原多项式，因此本算法采用整数环为唯一分解环，采用随机数方式生成。**

4.2 算法

Algorithm 8 本原多项式生成

输入： 次数 n

输出: 一个随机生成的 n 次本原多项式

```
function GENERATE(n)
    factors ← []
    factors.append(randint(1, 50))
    gcd ← factors[0]
    for each i ∈ [1, n] do
        while True do
            temp ← randint(1, 50)
            tgcd ←GCD(temp, gcd)
            if tgcd == 1 then
                gcd ← tgcd
                factors.append(temp)
                break
            end if
        end while
    end for
    return factors
end function
```

4.3 测试样例

```
15
50 + 11x^1 + 7x^2 + 13x^3 + 31x^4 + 33x^5 + 49x^6 + 27x^7 + 21x^8
+ 3x^9 + 31x^10 + 41x^11 + 1x^12 + 17x^13 + 43x^14 + 37x^15
```

图 5: 本原多项式

5 感想

我希望实验要求能明确一点。**如果出题者不站在解题者的角度思考问题，则这个要求不可能是完备而清晰的**。对于本原多项式和有限域的四则运算的讲解太不清晰了，导致很多内容需要我花很多时间来弄明白。事实上，关于本原多项式实验的确切要求我直到此刻还没有弄懂。

希望老师能提前给出实验内容，以便于预习。此外，本次实验报告提交时间有些早了。

尽管如此，我还是学会了很多内容，无论是数学、编程还是排版。希望这门课能变得更好。