整数の素因数分解について

山本修身

2016-05-16
アルゴリズム特論

素数判定の復習

* フェルマーの小定理を応用して素数を求めることができる.
* pが素数であれば,

$$a^{p-1} = 1 \pmod{p}$$

フェルマーの定理

* これ対してミラーとラビンは以下のようなこれより厳しい条件を考えた
* pが素数であれば，以下のようにsとdを決定して

$$p - 1 = 2^s d$$

以下のいずれかの条件が成り立つ

pが素数でないのにこのテストを通ってしまう確率は1/4以下(実際はもっとずっと小さい)

$$a^d = 1 \pmod{p} \qquad a^{2^i d} = -1 \pmod{p} \quad (0 \leq \exists i < s)$$

MRアルゴリズムの実現

* ミラーとラビンのアルゴリズムをPythonで実現すると以下のようになる.

```
# The Miller-Rabin Method

import random

def modpow(a, b, p):
    if b == 0: return 1
    else:
        mm = modpow(a, b / 2, p)
        res = (mm * mm) % p
        if b % 2 == 1: res = (res * a) % p
        return res

def MRTest1(a, p):
    d = p - 1
    s = 0
    while d % 2 == 0:
        s += 1
        d /= 2
    k = modpow(a, d, p)
    if k == 1: return True
    ss = 0
    while ss <= s:
        if (k + 1) % p == 0: return True
        k = (k * k) % p
        ss += 1
    return False
```

```
def MRTest(p):
    r = random.Random()
    for i in range(20):
        a = r.randint(2, p - 1)
        if not MRTest1(a, p): return False
    return True

def find_prime(pp):
    while not MRTest(pp):
        pp += 2
    return pp

a = find_prime(92429849809837973)
b = find_prime(98943752524593701)
p = a * b
print a, b, p
print MRTest(p)
print modpow(3, p - 1, p)
```

```
92429849809837999 98943752524593761
9145356185469980673640298696124239
False
6853420506586464334624612615563061
```

誕生日の問題

問題：今，名古屋市の地下鉄のある車両に30人乗っている．この30人の人たちの誕生日がすべて異なる確率を求めよ．

$$p = \frac{365}{365} \cdot \frac{364}{365} \cdots \frac{365-29}{365} = 0.293683757281$$

すなわち，70%以上の確率で同じ誕生日の人がいることになる

スターリングの公式

* nの階乗 n! の漸近展開が知られている．

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

* この公式はnが大きくなればなるほど精度が高くなっていく．ここではこの公式を用いて解析を行う．

誕生日の問題の一般化 (1)

問題：要素がNの集合があるとする．この集合からランダムに要素を選択してからその要素を戻すという作業をm回繰り返したとき，すべての要素が異なる確率はいくつか．

$$p = \frac{N}{N}\frac{N-1}{N}\cdots\frac{N-m+1}{N} = \frac{1}{N^m}\frac{N!}{(N-m)!}$$

$$= \frac{1}{N^m}\frac{\sqrt{2\pi N}\left(\frac{N}{e}\right)^N}{\sqrt{2\pi(N-m)}\left(\frac{N-m}{e}\right)^{N-m}} = \frac{1}{e^m}\left(\frac{N}{N-m}\right)^{N-m+1/2}$$

pの対数を計算すれば,

$$\log p = -m + \left(N - m + \frac{1}{2}\right)\log\left(\frac{N}{N-m}\right)$$

誕生日の問題の一般化 (2)

これより，Nがmに比べて十分に大きいとすれば,

$$\begin{aligned}\log p &= -m - \left(N - m + \frac{1}{2}\right)\log\left(1 - \frac{m}{N}\right) \\ &\sim -m + \left(N - m + \frac{1}{2}\right)\frac{m}{N} \\ &= -m + m - \frac{m^2}{N} + \frac{m}{2N} = -\frac{m^2}{N} + \frac{m}{2N}\end{aligned}$$

ある定数cを用いて，$\log p < -c$とすれば,

$$\left(m - \frac{1}{4}\right)^2 > cN + \frac{1}{16}$$

となり，これより,

誕生日の問題の一般化 (3)

* mが√Nくらいまで大きくなれば，ある程度の確率が小さくなることがわかる．

$$m > \sqrt{cN + \frac{1}{16}} + \frac{1}{4}$$

* 逆に言えば重複した要素が存在する確率が高くなる．

ρ methodによる因数分解 (1)

誕生日の問題とその解を用いることにより，整数を因数分解するアルゴリズムを構成することができる．

ある巨大な合成数Nがあり，N = pqと因数分解できるとする．今，適当な関数fが存在して，

$$x_{n+1} = f(x_n)$$

によって，0,..., N - 1の整数がランダムに生成されるとする．前述の誕生日の問題からこの系列がランダムであるとすれば 系列 $\{x_n \bmod p\}$，$\sqrt{p}$ 個程度このランダム列をとってくれば，そのなかに同じ値が存在することになる．さらに，

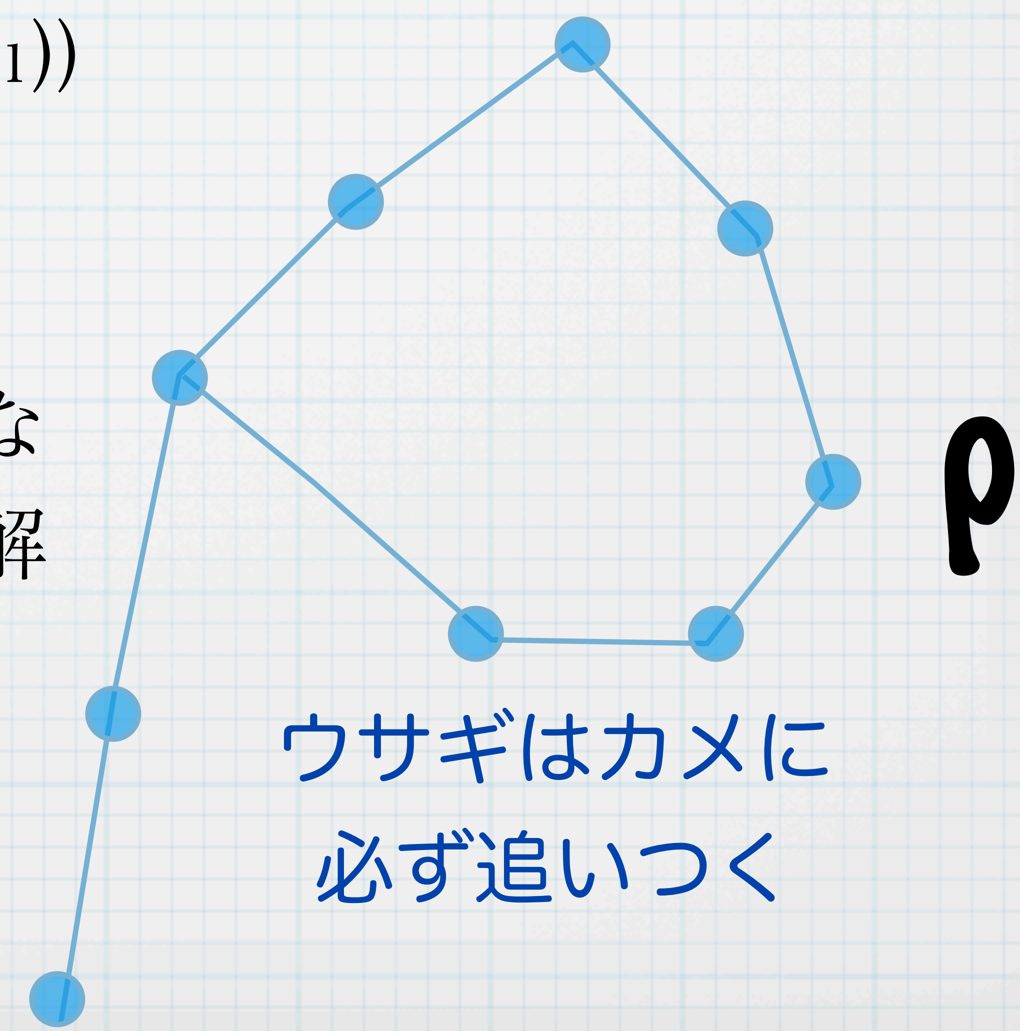
$$x_{n+1} \bmod p = f(x_n) \bmod p = f(x_n \bmod p) \bmod p$$

なので，系列はf(x) mod p = g(x) によって，初期の要素から$\{x_n \bmod p\}$の系列を順次作り出していくことができる．

ρ methodによる因数分解 (2)

そこで，前述の誕生日の問題の一般化から，$\{x_n \bmod p\}$の系列の周期が$\sqrt{p}$程度である確率はある程度高い．すなわち，$\sqrt{p}$回くらい繰り返せば同じ値に戻る可能性が高いことになる．

この繰り返しを（フロイドによる）ρ methodと呼ばれる方法で見つけ出す．一般にこの構造は同じ数が出てくればそこから先は繰り返しになる．そこで，$x_n = g(x_{n-1})$と$y_n = g(g(y_{n-1}))$という２つの列で一致したものが出現したときループに入ったことが確認できる．この場合，$\mathrm{GCD}(|x_n - y_n|, N)$を計算して1でない値が得られれば，Nの自明でない因数分解が得られたことになる．



ρ methodによる因数分解 (3)

```
n = 9145356185469980673640298696124239

def gcd(x, y):
    if x == 0:
        return y
    elif y == 0:
        return x
    else:
        return gcd(y, x % y)

def g(x):
    return (x * x + 1) % n

def gg(x):
    return g(g(x))
```

```
def work():
    x = 1
    y = 1
    count = 0
    while True:
        x = g(x)
        y = gg(y)
        pp = gcd(abs(x - y), n)
        if pp != 1:
            break
        count += 1
        if count % 100000 == 0:
            print count
    print "ans = ", pp

work()
```

```
79200000
79300000
79400000
79500000
79600000
79700000
ans =  98943752524593761
```

プログラムは上のようになる.

フェルマー数

* フェルマーが素数であると主張した整数の系列.

```
def exp(a, n):
  if n == 0: return 1
  else:
    k = exp(a, n / 2)
    if n % 2 == 0:
      return k * k
    else:
      return k * k * a

def print_fermat():
  for i in range(10):
    print i, exp(2, exp(2, i)) + 1

print_fermat()
```

$$F_i = 2^{2^i} + 1$$

F_4までは素数であったが，オイラーによってF_5が素数でないことが示された.

```
0 3
1 5
2 17
3 257
4 65537
5 4294967297
6 18446744073709551617
7 340282366920938463463374607431768211457
8
115792089237316195423570985008687907853269
984665640564039457584007913129639937
9
134078079299425970995740249982058461274793
658205923933777235614437217640300735469768
018742981669034276900318581864860508537538
828119465699464336490060084097
```

ρ methodの改良 (1)

Richard Brentによる改良．いちいちそれぞれのペアーについてgcdを計算するのは無駄である．gcd(a, n) > 1ならばgcd(ab, n) > 1なので，下の例では100回に 1 回だけgcdを計算している．

```
def work():
    x = 1
    y = 1
    count = 0
    count2 = 0
    pp = 1
    while True:
        x = g(x)
        y = gg(y)
        pp = (abs(x - y) * pp) % n
        if count2 % 100 == 0:
            px = gcd(pp, n)
            pp = 1
            if px > 1: break
        count += 1
        count2 += 1
        if count % 100000 == 0:
            print count
    print "ans = ", px
```

```
79000000
79100000
79200000
79300000
79400000
79500000
79600000
79700000
ans =  98943752524593761

real	6m36.551s
user	6m23.213s
sys	0m1.158s
```

ρ methodの改良 (2)

Brentはさらにρ methodの考え方を変えて，y_{2^i}の値とそれ以降の$y_{2^{i+1}}$までの値を比較することによって，同じ値になるものを探す．このようにすることで1ステップあたいほぼ1回だけ関数gの評価すれば良いことになり効率的である．

```
def work():
    count = 0
    width = 1
    start = 0
    x0 = g(3)
    p = 1
    while True:
        width *= 2
        print "width =", width
        x = x0
        for i in range(width - 1):
            x = g(x)
            p = (p * abs(x - x0)) % n
            count += 1
            if count % 100 == 0:
                k = gcd(p, n)
                if k != 1:
                    print k
                    return
                p = 1
            if count % 1000000 == 0:
                print count
        x0 = g(x)
```

```
213000000
214000000
215000000
216000000
217000000
218000000
219000000
220000000
98943752524593761

real  10m7.440s
user  9m3.010s
sys   0m1.466s
```

レポート問題 2

1. フェルマー数のうち5番目から11番目までの数が素数か否かをミラー=ラビンのアルゴリズムによって判定せよ.

2. これらの数の素因数分解を出来る限り行え.

3. 以下の数の素因数分解を試みよ.

914535618546997293219643669199126899

しめきり： 2017年6月18日（月）の講義まで
提出先： 講義の最後で集める