

## 計算量 (complexity)

- **時間計算量**: 計算に必要な手間  
(基本となる演算の回数)
- **空間計算量**: 計算に必要なメモリ量  
(必要な数表の大きさ・覚えておくべき途中結果)

理論上の厳密な定式化には  
計算モデルとしてチューリングマシンを用いるが、  
通常は、決まった桁数の四則演算 1 回を  
1 ステップと数えることが多い

入力データ長  $n$  に対する

増加のオーダー (Landau の  $O$ -記号) で表す

## Landau の $O$ -記号・ $o$ -記号

$f, g : \mathbb{N} \longrightarrow \mathbb{R}_{>0}$  に対し、

$$f = O(g) \iff \exists N > 0, \exists C > 0 : \forall n : \\ (n \geq N \implies f(n) \leq Cg(n))$$

$$f = o(g) \iff \frac{f(n)}{g(n)} \longrightarrow 0 \quad (n \rightarrow \infty) \\ \iff \forall \varepsilon > 0 : \exists N > 0 : \forall n : \\ (n \geq N \implies f(n) \leq \varepsilon g(n))$$

## 計算量 (complexity)

問題を解くアルゴリズムによって決まる

… アルゴリズムの計算量

→ アルゴリズムの効率 の評価

問題の計算量:

その問題を解くアルゴリズムの計算量の下限

最も効率良く解くと、どれ位で解けるか

= どうしてもどれ位必要か

= どれ位難しい問題か

→ 問題の難しさ の評価

## 基本的な例

- 加法:  $O(n)$
  
- 乗法:  $O(n^2)$  かと思いきや  $O(n \log n \log \log n)$   
(高速フーリエ変換 (FFT))

## 例: 互除法

- 入力: 正整数  $x, y$   
入力データ長:

$$n = \lceil \log_2 x \rceil + \lceil \log_2 y \rceil \sim \max\{\log x, \log y\}$$

- 出力: 最大公約数  $d = \gcd(x, y)$

### 計算量の評価:

- 割算の回数:  $O(n)$
- 1回の割算: 素朴な方法でも  $O(n^2)$   
(FFT を使えば  $O(n \log n \log \log n)$ )

→ 併せて  $O(n^3)$  (FFT で  $O(n^2 \log n \log \log n)$ )

… 十分に高速なアルゴリズム

## 重要な難しさのクラス

多項式時間 P ...  $\exists k : O(n^k)$

- “事実上計算可能” な難しさ
- 計算モデルの変更に関して頑健な  
計算量のクラス

「しらみつぶし」が入ると  
大体  $O(2^n)$  程度以上になる (指数時間 EXP)  
“事実上計算不可能”

## 例: 素数判定 (PRIMES)

$n = \log_2 N$  :  $N$  の二進桁数

試行除算 (小さい方から割っていく) だと  
 $O(n^k 2^{n/2})$  くらい掛かりそう

実は多項式時間で解ける!!

Agrawal-Kayal-Saxena

“PRIMES is in P” (2002)

(出版は Ann. of Math. 160(2) (2004), 781-793.)

## 例: 素数判定 (PRIMES)

$n = \log_2 N$  :  $N$  の二進桁数

試行除算 (小さい方から割っていく) だと  
 $O(n^k 2^{n/2})$  くらい掛かりそう

実は多項式時間で解ける!!

**Agrawal-Kayal-Saxena**

**“PRIMES is in P” (2002)**

(出版は **Ann. of Math. 160(2) (2004), 781-793.**)



## 素数判定に関する Fermat テスト

### Fermat の小定理

- $p$  : 素数  $\implies (\forall a \in \mathbb{Z} : a^p \equiv a \pmod{p})$

対偶を取って、

- $(\exists a \in \mathbb{Z} : a^N \not\equiv a \pmod{N}) \implies N$  : 合成数

多くの  $a \in \mathbb{Z}$  に対し、 $a^N \equiv a \pmod{N}$  ならば、  
 $N$  が素数である可能性は高いだろう

しかし実は、

$$\forall a \in \mathbb{Z} : a^N \equiv a \pmod{N}$$

となる合成数  $N$  も存在 (擬素数・Carmichael 数)

## 素数判定に関する Fermat テスト

### Fermat の小定理

- $p$  : 素数  $\implies (\forall a \in \mathbb{Z} : a^p \equiv a \pmod{p})$

対偶を取って、

- $(\exists a \in \mathbb{Z} : a^N \not\equiv a \pmod{N}) \implies N$  : 合成数

多くの  $a \in \mathbb{Z}$  に対し、 $a^N \equiv a \pmod{N}$  ならば、  
 $N$  が素数である可能性は高いだろう

しかし実は、

$$\forall a \in \mathbb{Z} : a^N \equiv a \pmod{N}$$

となる合成数  $N$  も存在 (擬素数・Carmichael 数)

## 素数判定に関する Fermat テスト

### Fermat の小定理

- $p$  : 素数  $\implies (\forall a \in \mathbb{Z} : a^p \equiv a \pmod{p})$

対偶を取って、

- $(\exists a \in \mathbb{Z} : a^N \not\equiv a \pmod{N}) \implies N$  : 合成数

多くの  $a \in \mathbb{Z}$  に対し、 $a^N \equiv a \pmod{N}$  ならば、  
 $N$  が素数である可能性は高いだろう

しかし実は、

$$\forall a \in \mathbb{Z} : a^N \equiv a \pmod{N}$$

となる合成数  $N$  も存在 (擬素数・Carmichael 数)

## 素数判定に関する Fermat テスト

### Fermat の小定理

- $p$  : 素数  $\implies (\forall a \in \mathbb{Z} : a^p \equiv a \pmod{p})$

対偶を取って、

- $(\exists a \in \mathbb{Z} : a^N \not\equiv a \pmod{N}) \implies N$  : 合成数

多くの  $a \in \mathbb{Z}$  に対し、 $a^N \equiv a \pmod{N}$  ならば、  
 $N$  が素数である可能性は高いだろう

しかし実は、

$$\forall a \in \mathbb{Z} : a^N \equiv a \pmod{N}$$

となる合成数  $N$  も存在 (擬素数・Carmichael 数)

## Agrawal-Kayal-Saxena の多項式時間素数判定

Fermat テストを多項式版にしたもの

$$\forall a \in \mathbb{Z} : (X + a)^N \equiv X + a \pmod{N} \quad ?$$

- 正しく素数を判定できる
- 多項式時間で判定できる
  - ★ 比較的少ない個数の  $a$  で確かめれば良い
  - ★ 比較的次数の低い多項式を法とした計算で良い

## Agrawal-Kayal-Saxena の多項式時間素数判定

Fermat テストを多項式版にしたもの

$$\forall a \in \mathbb{Z} : (X + a)^N \equiv X + a \pmod{N} \quad ?$$

- 正しく素数を判定できる
- 多項式時間で判定できる
  - ★ 比較的少ない個数の  $a$  で確かめれば良い
  - ★ 比較的次数の低い多項式を法とした計算で良い

## 素数判定と素因数分解との違い

このような効率の良い素数判定は、  
具体的に素因数を見付けている訳ではない

素因数分解は  $P$  であるかどうか未解決  
(多項式時間アルゴリズムが知られていない)

現状で知られているのは、  
“準指数時間”  $L_N[u, v]$  ( $0 < u < 1$ )  
のアルゴリズム  
(現時点で最高速なのは  $u = 1/3$ )

## 素数判定と素因数分解との違い

このような効率の良い素数判定は、  
具体的に素因数を見付けている訳ではない

素因数分解は  $P$  であるかどうか未解決  
(多項式時間アルゴリズムが知られていない)

現状で知られているのは、  
“準指数時間”  $L_N[u, v]$  ( $0 < u < 1$ )  
のアルゴリズム  
(現時点で最高速なのは  $u = 1/3$ )



素因数分解アルゴリズム等の計算量を表すのに

$$L_N[u, v] := \exp(v(\log N)^u (\log \log N)^{1-u})$$

が良く用いられる

$n = \log N$  ( $N$  の桁数) とおくと、

- $L_N[0, v] = e^{v \log \log N} = n^v$  : 多項式時間
- $L_N[1, v] = e^{v \log N} = e^{vn}$  : 指数時間

主パラメタは  $u$  で、多項式時間と指数時間とを補間

## 代表的な素因数分解法

- $(p - 1)$ -法
- 楕円曲線法 (Elliptic Curve Method)
- 二次篩法 (Quadratic Sieve)
- 数体篩法 (Number Field Sieve)

## 代表的な素因数分解法

- $(p - 1)$ -法
- 楕円曲線法 (Elliptic Curve Method)
- 二次篩法 (Quadratic Sieve)
- 数体篩法 (Number Field Sieve)

## 二次篩法の原理

$y^2 \equiv z^2 \pmod{N}$  となる  $y, z$  を探す

$$\longrightarrow N \mid (y^2 - z^2) = (y - z)(y + z)$$

$y \equiv \pm z \pmod{N}$  でない限り、  
 $\gcd(y - z, N)$  が  $N$  の非自明な約数 !!

(最大公約数は互除法により高速に計算可能)

## 二次篩法の原理

$y^2 \equiv z^2 \pmod{N}$  なる  $y, z$  の組を見付けるには？  
 $x$  を沢山取り、 $x^2$  を  $N$  で割った余り  $r$  を沢山作る  
$$x^2 \equiv r \pmod{N}$$

$r = z^2$  なら  $y = x$  で **OK**

そんなにうまくいくのか？

- $x \approx \sqrt{kN}$  にとって、  
 $r = x^2 - kN$  が小さくなるようにする
- それを組み合わせて (掛け合わせて)  
うまく作れることがある

## 二次篩法の原理

$y^2 \equiv z^2 \pmod{N}$  なる  $y, z$  の組を見付けるには？  
 $x$  を沢山取り、 $x^2$  を  $N$  で割った余り  $r$  を沢山作る  
$$x^2 \equiv r \pmod{N}$$

$r = z^2$  なら  $y = x$  で **OK**

そんなにうまくいくのか？

- $x \approx \sqrt{kN}$  にとって、  
 $r = x^2 - kN$  が小さくなるようにする
- それを組み合わせる (掛け合わせて)  
うまく作れることがある

## 二次篩法の原理

$y^2 \equiv z^2 \pmod{N}$  なる  $y, z$  の組を見付けるには？  
 $x$  を沢山取り、 $x^2$  を  $N$  で割った余り  $r$  を沢山作る  
$$x^2 \equiv r \pmod{N}$$

$r = z^2$  なら  $y = x$  で **OK**

そんなにうまくいくのか？

- $x \doteq \sqrt{kN}$  にとって、  
 $r = x^2 - kN$  が小さくなるようにする
- それを組み合わせせて (掛け合わせて)  
うまく作れることがある

例:  $N = 18281$

$$145^2 \equiv 2744 = 2^3 \cdot 7^3$$

$$149^2 \equiv 3920 = 2^4 \cdot 5^1 \cdot 7^2$$

$$159^2 \equiv 7000 = 2^3 \cdot 5^3 \cdot 7^1$$

---

$$\begin{aligned} (145 \cdot 149 \cdot 159)^2 &\equiv 2^{10} \cdot 5^4 \cdot 7^6 \\ &\equiv (2^5 \cdot 5^2 \cdot 7^3)^2 \end{aligned}$$

$$145 \cdot 149 \cdot 159 \equiv 16648 =: y$$

$$2^5 \cdot 5^2 \cdot 7^3 \equiv 185 =: z$$

$$\gcd(y - z, N) = 101$$

:  $N = 18281$  の非自明な約数!!



## 二次篩法の原理

うまく組み合わせるには、

$x^2$  を  $N$  で割った余りの

素因数の重なりが多いと良い

- 始めに幾つかの小さい素数を決めておく (因子底)
- $x^2$  を  $N$  で割った余り  $r$  で、  
素因数が因子底の素数のみから成るものを貯める
- 必要なだけ貯ったら、  
平方数を作る組合せを探す計算に移行

## 二次篩法の原理

うまく組み合わせるには、

$x^2$  を  $N$  で割った余りの

素因数の重なりが多いと良い

- 始めに幾つかの小さい素数を決めておく (因子底)
- $x^2$  を  $N$  で割った余り  $r$  で、  
素因数が因子底の素数のみから成るものを貯める
- 必要なだけ貯ったら、  
平方数を作る組合せを探す計算に移行

## 何が「篩」か？

候補の  $x$  を沢山計算するときに、  
採用され易そうな候補  $x$  だけ選んで計算

$x^2 - N$  が因子底の幾つかで割れることが  
予め判っているものを選ぼう

或る  $x^2 - N$  が  $p$  で割れていたら、

$$(x \pm p)^2 - N, (x \pm 2p)^2 - N, \dots$$

も  $p$  で割れることが予め判る !!

→ 因子底の素数毎に、

候補に残り易そうな  $x$  が 等間隔に並ぶ !!

## 素因数分解法

- 一般的な整数に対する方法
- 特殊な形の整数に対する方法

一般的な整数に対する方法で

現在最強と言われている方法

… 数体篩法 (**number field sieve method**)

二次篩法のアイデア + 代数的整数論の知見