

公開鍵暗号の例: RSA 暗号

Rivest, Shamir, Adleman (1977)

- 大きな素数 p, q を選び、積 $N = pq$ を作る
- N を用いて、公開鍵 e ・ 秘密鍵 d の対を作る
- 暗号化の計算は N と公開鍵 e とから可能
- 復号は秘密鍵 d を用いる
- N と公開鍵 e とから秘密鍵 d を求めるには、 N の素因子分解 $N = pq$ が必要
- しかしそれは困難 (膨大な計算時間が掛かる)

RSA 暗号 (Rivest-Shamir-Adleman)

p, q : 相異なる大きな素数

$N = pq$, $m = \text{lcm}(p - 1, q - 1)$, $ed \equiv 1 \pmod{m}$

- (N, e) : 公開鍵 (暗号化鍵)
- d : 秘密鍵 (復号鍵)

平文・暗号文は $0, 1, \dots, N - 1$ に符号化
($\text{mod} N$ で考える)

平文 M の暗号化 : $C = E(M) \equiv M^e \pmod{N}$

暗号文 C の復号 : $M = D(C) \equiv C^d \pmod{N}$

冪乗の高速計算

p, q : 相異なる大きな素数 (実際には 1024 bit 以上)

$N = pq$, $m = \text{lcm}(p - 1, q - 1)$, $ed \equiv 1 \pmod{m}$

- 平文 M の暗号化 : $C = E(M) \equiv M^e \pmod{N}$
- 暗号文 C の復号 : $M = D(C) \equiv C^d \pmod{N}$

暗号化して復号すると元に戻ることは判った。

しかし、高速に計算できるのか？ (e, d は巨大)

→ 冪乗の高速計算法が必要

RSA 暗号 (Rivest-Shamir-Adleman)

公開鍵 (N, e) から秘密鍵 d が計算できるか？

- N の素因数分解 $N = pq$ を知っていれば容易
- 事実上 N の素因数分解と同程度の困難さ

「困難さ」… 計算時間が掛かる

RSA 暗号の安全性 \iff 素因数分解の困難さ

“計算量的安全性 (computational secrecy)”

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

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

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、

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

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

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

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

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

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))

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

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”

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”

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”