

本講義最後の話題は、

計算量

について

問題の難しさを如何に計るか？

Church-Turing の提唱 (再掲)

「全てのアルゴリズム (計算手順) は、
チューリングマシンで実装できる」

(アルゴリズムと呼べるのは
チューリングマシンで実装できるものだけ)

… 「アルゴリズム」の定式化

計算量 (complexity)

- **時間計算量** : 計算に掛かるステップ数
(TM での計算の遷移の回数)
- **空間計算量** : 計算に必要なメモリ量
(TM での計算で使うテープの区画数)

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 $\dots \exists k : O(n^k)$

- “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
(複数テープ TM などに変更しても不変)

「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.**)

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

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)

計算困難な問題の数理技術としての利用

素因数分解の困難さを利用した暗号方式

… **RSA 暗号** (Rivest-Shamir-Adleman)

鍵となる整数 n の素因数分解を

知っていれば解読できるが、
知らないとは解読できない

→ 詳しくは暗号理論などの授業で

計算困難な問題の数理技術としての利用

素因数分解の困難さを利用した暗号方式

… **RSA 暗号** (Rivest-Shamir-Adleman)

鍵となる整数 n の素因数分解を

知っていれば解読できるが、
知らないとは解読できない

→ 詳しくは暗号理論などの授業で

例：並べ替え (sorting)

多くの数値データを大きさの順に並べ替える操作

n 個のデータの比較は $\frac{n(n-1)}{2}$ 通り

→ 全ての組合せを比較しても $O(n^2)$ で済む筈

- 具体的なアルゴリズムは？
- もっと早くなる？ ($o(n^2)$ になる？)

例：並べ替え (sorting)

多くの数値データを大きさの順に並べ替える操作

n 個のデータの比較は $\frac{n(n-1)}{2}$ 通り

→ 全ての組合せを比較しても $O(n^2)$ で済む筈

- 具体的なアルゴリズムは？
- もっと早くなる？ ($o(n^2)$ になる？)

例：並べ替え (sorting)

多くの数値データを大きさの順に並べ替える操作

n 個のデータの比較は $\frac{n(n-1)}{2}$ 通り

→ 全ての組合せを比較しても $O(n^2)$ で済む筈

- 具体的なアルゴリズムは？
- もっと早くなる？ ($o(n^2)$ になる？)

並べ替えの例：バブルソート

- 端から順に、隣と比べて逆順なら入れ換える
(末尾が決まる)
- (末尾を除いて) これを繰り返す

比較回数： $\frac{n(n-1)}{2}$ 回 \longrightarrow 計算量 $O(n^2)$

並べ替えの例：バブルソート

- 端から順に、隣と比べて逆順なら入れ換える
(末尾が決まる)
- (末尾を除いて) これを繰り返す

比較回数： $\frac{n(n-1)}{2}$ 回 \longrightarrow 計算量 $O(n^2)$

並べ替えの例：マージソート

- 半分に分ける
- それぞれをソートする (分割統治)
- 両者を併せる

「それぞれをソート」の部分は再帰を用いる

計算量は？

並べ替えの例：マージソート

- 半分に分ける
- それぞれをソートする (分割統治)
- 両者を併せる

「それぞれをソート」の部分は再帰を用いる

計算量は？

並べ替えの例：マージソート

- 半分に分ける
- それぞれをソートする (分割統治)
- 両者を併せる

「それぞれをソート」の部分は再帰を用いる

計算量は？

並べ替えの例：マージソート

- 半分に分ける
- それぞれをソートする
- 両者を併せる → 計算量は $O(n)$

計算量を $f(n)$ とすると、

$$f(n) = 2f\left(\frac{n}{2}\right) + O(n)$$

$$\longrightarrow f(n) = O(n \log n)$$

並べ替えの例：マージソート

- 半分に分ける
- それぞれをソートする
- 両者を併せる → 計算量は $O(n)$

計算量を $f(n)$ とすると、

$$f(n) = 2f\left(\frac{n}{2}\right) + O(n)$$

$$\longrightarrow f(n) = O(n \log n)$$

並べ替えの例：マージソート

- 半分に分ける
- それぞれをソートする
- 両者を併せる → 計算量は $O(n)$

計算量を $f(n)$ とすると、

$$f(n) = 2f\left(\frac{n}{2}\right) + O(n)$$

$$\longrightarrow f(n) = O(n \log n)$$

最悪計算量と平均計算量

計算量の理論では、入力データに対して

「どんな場合でも (最悪でも) これだけで出来る」

というのが計算量の定義 (最悪計算量) だが、

実際に計算するには、ランダムなデータに対して

「平均的にはこれだけで出来る」

というのも重要である (平均計算量)

並べ替えの例：クイックソート

- 基準値 (pivot) を選ぶ
- それより大きい値と小さい値とに分ける
- それぞれをソートする (分割統治)

計算量は

- 最悪では $O(n^2)$ にしかない
- しかし平均では $O(n \log n)$ で、
多くの場合、実際にはその中でもかなり速い

並べ替えの例：クイックソート

- 基準値 (pivot) を選ぶ
- それより大きい値と小さい値とに分ける
- それぞれをソートする (分割統治)

計算量は

- 最悪では $O(n^2)$ にしかない
- しかし平均では $O(n \log n)$ で、
多くの場合、実際にはその中でもかなり速い

並べ替えの例：挿入ソート

実際に扱うデータはランダムとは限らない

ソート済みデータに変更があった場合など、
殆どソートされているデータに対して速い方法

- それまでのデータをソートしておく
- 次のデータを適切な場所に挿入する

最悪計算量は $O(n^2)$ だが、場合によっては使える

並べ替えの例：挿入ソート

実際に扱うデータはランダムとは限らない

ソート済みデータに変更があった場合など、
殆どソートされているデータに対して速い方法

- それまでのデータをソートしておく
- 次のデータを適切な場所に挿入する

最悪計算量は $O(n^2)$ だが、場合によっては使える

並べ替えの例：挿入ソート

実際に扱うデータはランダムとは限らない

ソート済みデータに変更があった場合など、
殆どソートされているデータに対して速い方法

- それまでのデータをソートしておく
- 次のデータを適切な場所に挿入する

最悪計算量は $O(n^2)$ だが、場合によっては使える