

さて、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？

さて、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

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

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、

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

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

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
(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”

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”

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
(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.)

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.**)