期末試験のお知らせ

7月27日(月) 15:15 ～ 16:15
(60分試験)

9-255 教室 (ここじゃない)

- 今日の講義内容まで
- 学生証必携

レポート提出について

期日: 8月7日(金)20時頃まで

内容:

配布プリントのレポート問題の例のような内容及び授業に関連する内容で、授業内容の理解または発展的な取組みをアピールできるようなもの

提出方法:

- **授業時に手渡し**
- **4-574 室扉のレポートポスト**
- **電子メイル**

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

「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

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

$$f = O(g) \overset{\leftarrow}{\Longleftrightarrow} \exists N > 0, \exists C > 0 : \forall n : \\ (n \geq N \Longrightarrow f(n) \leq Cg(n))$$

$$f = o(g) \overset{\leftarrow}{\Longleftrightarrow} \frac{f(n)}{g(n)} \longrightarrow 0 \ (n \to \infty) \\ \Longleftrightarrow \forall \varepsilon > 0 : \exists N > 0 : \forall n : \\ (n \geq N \Longrightarrow f(n) \leq \varepsilon g(n))$$

Landau の O-記号・o-記号

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

$$f = O(g) \overset{\leftarrow}{\Longleftrightarrow} \exists N > 0, \exists C > 0 : \forall n : \\ (n \geq N \Longrightarrow f(n) \leq Cg(n))$$

$$f = o(g) \overset{\leftarrow}{\Longleftrightarrow} \frac{f(n)}{g(n)} \longrightarrow 0 \ (n \to \infty) \\ \Longleftrightarrow \forall \varepsilon > 0 : \exists N > 0 : \forall n : \\ (n \geq N \Longrightarrow f(n) \leq \varepsilon g(n))$$

計算量 (complexity)

問題を解くアルゴリズムによって決まる

$\cdots$ アルゴリズムの計算量

$\longrightarrow$ 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

$\longrightarrow$ 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

$\cdots$ アルゴリズムの計算量

$\longrightarrow$ 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

$\longrightarrow$ 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

$\cdots$ アルゴリズムの計算量

$\longrightarrow$ 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

$\longrightarrow$ 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

$\cdots$ アルゴリズムの計算量

$\longrightarrow$ 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

$\longrightarrow$ 問題の難しさ の評価

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:

- **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 $O(n \log n \log\log n)$**)**

$\longrightarrow$ **併せて** $O(n^3)$ **(FFT で** $O(n^2 \log n \log\log n)$**)**
$\cdots$ **充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 $\cdots \exists k : O(n^k)$

“事実上計算可能な難しさ”

「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 $\cdots \exists k : O(n^k)$

“事実上計算可能な難しさ”

「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.)

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 $u=1/3$)

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

$\cdots$ 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、
知らないと解読できない

$\longrightarrow$ 来年春期の「応用数学 I ／情報数学特論」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

$\cdots$ 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、
知らないと解読できない

$\longrightarrow$ 来年春期の「応用数学 I ／情報数学特論」で

計算量にも “非決定性” の概念がある

あてずっぽうを許して、
うまくいけばどの位で解けるか
＝ 答を知って、その検証にどの位かかるか

非決定性多項式時間 (NP):
非決定性の計算モデルで多項式時間で解ける

例: 素因数分解は NP
$\cdots$ 素因数を知っていれば割算 1 回だけ

計算量にも “非決定性” の概念がある

あてずっぽうを許して、
うまくいけばどの位で解けるか
＝ 答を知って、その検証にどの位かかるか

非決定性多項式時間 (NP):
非決定性の計算モデルで多項式時間で解ける

例: 素因数分解は NP
… 素因数を知っていれば割算 1 回だけ

計算量にも “非決定性” の概念がある

あてずっぽうを許して、
うまくいけばどの位で解けるか
＝ 答を知って、その検証にどの位かかるか

非決定性多項式時間 (NP):
非決定性の計算モデルで多項式時間で解ける

例: 素因数分解は NP
… 素因数を知っていれば割算 1 回だけ

未解決問題 (P vs NP Problem)

$$\mathbf{P} = \mathbf{NP}$$

であるか否か ？

“The Millennium Problems”
の 7 つの問題のうちの 1 つ
(賞金 $1M)

未解決問題 (P vs NP Problem)

$$\mathbf{P} = \mathbf{NP}$$

であるか否か ？

“The Millennium Problems”
の 7 つの問題のうちの 1 つ
(賞金 $1M)

おしまい