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

Landau の O-記号・o-記号

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

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

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

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

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 $O(n \log n \log\log n)$**）**

$\longrightarrow$ **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**
- **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)**

二次篩法の原理

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

$\longrightarrow y^2 - z^2 = (y-z)(y+z)$ **が** N **で割切れる**

$y \equiv \pm z \pmod{N}$ **でない限り、**

$\gcd(y \pm z, N)$ **が** N **の非自明な約数 !!**

注：最大公約数は高速に計算可能（互除法）

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？

$\longrightarrow x$ を沢山取って、

x^2 を N で割った余り r を沢山作る

$$x^2 \equiv r \pmod N$$

$\longrightarrow r = z^2$ なら $y = x$ で OK

そんなにうまくいくのか？

- x を $\sqrt{N}$ の近くにとって、
  $x^2 - N$ 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

例：$N = 18281, \qquad \left[\sqrt{18281}\right] = 135$

$$
\begin{array}{rcccccccc}
145^2 & \equiv & 2744 & = & 2^3 & & & \cdot & 7^3 \\
149^2 & \equiv & 3920 & = & 2^4 & \cdot & 5^1 & \cdot & 7^2 \\
159^2 & \equiv & 7000 & = & 2^3 & \cdot & 5^3 & \cdot & 7^1 \\
\hline
(145 \cdot 149 \cdot 159)^2 & \equiv & & & 2^{10} & \cdot & 5^4 & \cdot & 7^6 \\
 & \equiv & & (& 2^5 & \cdot & 5^2 & \cdot & 7^3 \;)^2
\end{array}
$$

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

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

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

：$N = 18281$ **の非自明な約数 !!**

二次篩法の原理

うまく組み合わさるには、

x^2 を N で割った余りの
素因数の重なりが多いと良い

- **始めに小さい素数を幾つか決めておく（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、

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

も p で割れることが予め判る !!

$\longrightarrow$ **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

$\longrightarrow$ **「篩法」**

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

$\longrightarrow$ **詳しくは暗号理論などの授業で**

例：巨大な指数の冪の計算

RSA 暗号では、次の計算が必要になる：

$$C \equiv M^e \pmod{N}$$

ここで、 C, M, e, N **はどれも数百桁程度**

(500〜1000 **bit)**

単純に M **を** e **回掛けるのでは、**

数百桁回の乗算（と $\bmod N$ **の計算）が必要**

$\longrightarrow$ **事実上不可能（指数時間）**

計算を実行するには高速化の工夫が必要

例：巨大な指数の冪の計算（M^e の高速計算）

$$e = e_0 + e_1 \cdot 2 + e_2 \cdot 2^2 + \cdots + e_{n-1} \cdot 2^{n-1} + e_n \cdot 2^n$$

：e の二進法表示（二進 n 桁、$e_i = 0, 1$）

M^{2^k} **は、次の漸化式で、k 回の乗算で計算できる**

- $M^{2^0} = M$
- $M^{2^{k+1}} = (M^{2^k})^2$

M^e **は、**

$$M^e = \prod_{k=0}^{n} (M^{2^k})^{e_k} = \prod_{k\,:\,e_k=1} M^{2^k}$$

により、$O(n)$ 回の乗算で計算できる（多項式時間）

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 $\dfrac{n(n-1)}{2}$ 通り

$\longrightarrow$ 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：$\dfrac{n(n-1)}{2}$ **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**
- **両者を併せる** $\longrightarrow$ **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 $O(n^2)$ にしかならない**
- **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