第4章　計算の複雑さ入門

4.1. 計算の複雑さの理論概観

「計算可能か？」➔「どの程度の計算コストで計算可能か？」

計算の複雑さの理論 (Computational Complexity Theory)

(1) 計算量の上限に関する研究

(2) 計算量の下限に関する研究

(3) 計算の難しさについての構造的研究

(1) 計算量の上限に関する研究

効率のよいアルゴリズムの設計（アルゴリズム理論）

ある問題 X に対して，それを解くアルゴリズム A があり，サイズ n のどんな問題例に対しても A の時間計算量が $T(n)$ 以内であるとき，アルゴリズム A の時間計算量の上限は $T(n)$

（最悪時の漸近的時間計算量）

(2) **計算量の下限に関する研究**

問題 X に対する**どんなアルゴリズム**も最悪の場合には $T(n)$ 時間だけ必ずかかってしまうとき，問題 X の時間計算量の下限は $T(n)$.

- P $\neq$ NP予想
- 暗号システムの頑健さ

(3) **計算の難しさについての構造的研究**

“xx程度の難しさ”がもつ特徴について調べること.

難しさの程度による階層構造.

4.2. 計算時間の計り方

4.2.1. 標準形プログラム再考

定義4.1.（計算時間の定義）

***A*: *k*入力標準形プログラム**

$x_1, x_2, \ldots, x_k$: *A*への入力

- 全体は while ループ
- 各行は
  - 1つの if 文+pcへの代入
  - 基本命令1つ+pcへの代入

*A*のwhileループ１回り分の実行を*A*での**1ステップ**という.
入力$x_1, x_2, \ldots, x_k$に対して*A*が停止するまでに回るwhileループの回数を**<u>*A*の$x_1, x_2, \ldots, x_k$に対する計算時間</u>**（略して$A(x_1, x_2, \ldots, x_k)$の計算時間）という. ただし, 停止しないとき, 計算時間は無限大.

$$time\_A(x_1, x_2, \ldots, x_k) \equiv A(x_1, x_2, \ldots, x_k)\text{の計算時間}$$

$$time\_A(l) \triangleq \max\{time\_A(x_1, x_2, \ldots, x_k) : \sum_{1\le i\le k} |x_i| = l\}$$

標準形プログラム

```
prog プログラム名(input ...);
var pc: Σ*; ... ; Σ; ... ; Σ*;
 begin
    pc:=1;
    while pc ≠ 0 do
        case pc of
        1: (文);
        2: (文);
        3: (文);
        ...........
        K: (文);
        end-case
    end-while;
    halt(Σ*型の変数);
end.
```

各(文)の形は

- if 比較文 then pc:=k_1 else pc:=k_2 end-if
- 代入文; pc:=k;

のいずれか.

・各文が高々定数時間で実行できるための制約

u, u': Σ型の変数, v, v': Σ^*型の変数

c: Σ型の定数, s: Σ^*型の定数

（代入文）(1) $u := c$; (2) $u := u'$;

(3) $u :=$ head(v); (4) $u :=$ tail(v);

(5) $v := s$; ~~(6) $v := v'$;~~ ??

(7) $v :=$ right(v); (8) $v :=$ left(v);

(9) $v := u \# v$; (10) $v :=$ v $\# u$;

（比較文）(11) $u = c$ (12) $v = s$

・$v = v'$の形の比較は禁止されている.

4.2.2. プログラムの時間計算量

プログラムの時間計算量を**入力サイズ**の関数として表現
（入力文字列の長さ）

妥当なコード化：
元の対象のサイズに定数倍の範囲内で忠実なコード化

例4.5: 1進表記と2進表記
「数のサイズはその桁数」との立場では
2進表記は妥当なコード化であるが，
1進表記は冗長なコード化

定義4.3: 自然数上の関数 f, g に対し,

$\exists c,d>0, \forall n\ [f(n) \leqq c\,g(n)+d]$

となるとき, f は<u>オーダーgである</u>といい, $f=O(g)$ と記述する.

★定数c, dはnと無関係に定まることが必要.

定理4.1: 自然数上の任意の関数 f, g, h に対し次の関係が成立。

(1) $\forall n[f(n) \leqq g(n)] \rightarrow f=O(g)$

(2) $\exists c>0, \overset{\infty}{\forall} n[f(n) \leqq cg(n)] \rightarrow f=O(g)$

(3) $[f=O(g)$ かつ $g=O(h)] \rightarrow f=O(h)$

4.2.3. 問題の時間計算量

定義4.4. Φ を計算問題とし，t を自然数上の関数とする．
いま Φ を計算するプログラム A と定数 $c, d > 0$が存在して，

$$\forall l\ [\mathrm{time\_}A(l) \leqq ct(l) + d]$$

ならば，Φは$O(t)$時間計算可能，あるいはΦの時間計算量は$O(t)$であるという．

注意：ここでは計算問題として，集合の認識問題を想定している．

直観的には「問題Φは t 時間以下で計算可能」という意味。

(注1) A の時間計算量は t より低いかもしれない．
(注2) A よりも速くΦを計算するプログラムがあるかもしれない．

例4.7. 素数判定問題の時間計算量

素数判定問題(PRIME)
入力：自然数 n（ただし，2進表記）
質問：n は素数か？
PRIME $\equiv \{ \ulcorner n \urcorner : n$は素数$\}$

スターリングの公式：

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

```
prog Naive(input n);
begin
  for each i := 1 < i < n do
    if n mod i = 0 then reject end-if
  end-for;
  accept
end.
```

2 〜 n-1の数で割ってみる

$\log n \cdot \log i$ 時間

余談：
2002年に
$O(l^6)$
のアルゴリズム
が考案された!!

$$time\_\mathrm{Naive}(n) \le \sum_{1<i<n} (c \log n \log i + d)$$
$$= c \log n \log n! + dn = O(n(\log n)^2)$$

n の長さを l とすると，l はほぼ$\log n$だから，time_Naive=$O(l^2 2^l)$
故に，素数判定問題の時間計算量は（高々）$O(l^2 2^l)$

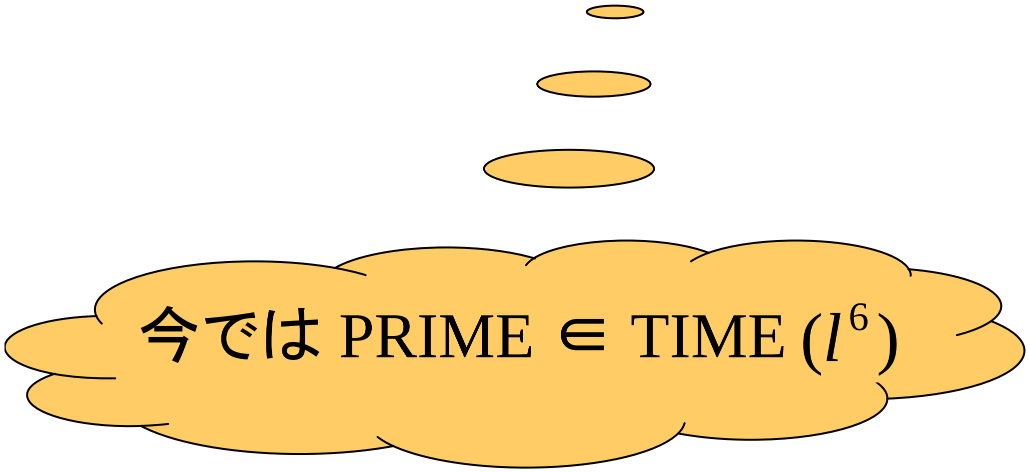
定義4.5.

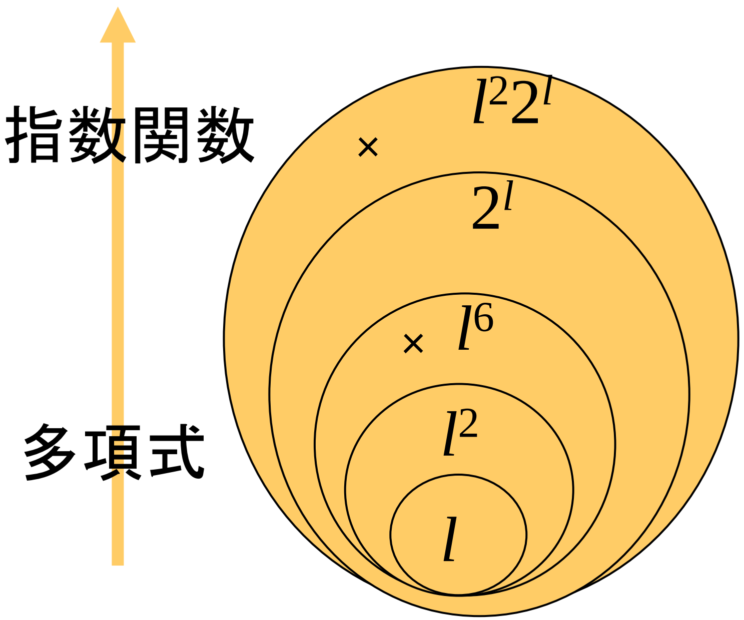
自然数上の関数 t に対し，時間計算量が $O(t)$ となる集合（i.e.,認識問題）の全体を **$O(t)$ 時間計算量クラス**といい，そのクラスを**TIME(t)**と表す．

また，t のような関数を制限時間と呼ぶ．

たとえば，$O(l^2 2^l)$ 時間で認識可能な集合を集めたクラスが TIME($l^2 2^l$)であり，集合 PRIME はその一要素．

$$\mathrm{PRIME} \in \mathrm{TIME}(l^2 2^l)$$





制限時間 t にふさわしい関数の条件

<u>自然な</u>制限時間(の定義)

(a) $\forall n\ [\ n \leq t(n)\ ]$
(b) $\forall n_1, n_2\ [n_1 < n_2 \rightarrow t(n_1) \leq t(n_2)]$
(c) 与えられた入力 x に対し，$\overline{t(|x|)}$（$t(|x|)$の1進表記）を求める計算が $O(t)$ 時間で可能

(a)の条件：入力を読むだけで n 時間かかってしまうから.
(c)の条件：時間が $t(n)$ になったら計算を打ち切るようにするためのタイマーが実現できるようにするため.
（1進表記を作って，1桁ずつ短くしていく）.

例4.8: 集合D = {<a,b>: aはbで割り切れる}の時間計算量

集合Dを認識するプログラムとして，下のプログラムを考える.

```
prog D(input x);
begin
   a:=get(x, 1̄); b:=get(x, 2̄);
   %   x = <a,b>の形でないときは，この時点でreject
   if a mod b = 0 then accept else reject end-if
end.
```

$O(|a||b|)$時間で計算可能

time_D(x) = $O(|x|^2)$

入力がx=<a,b>の形のときは$O(|x|^2)$時間かかるが，そうでない場合にはmod計算の前にrejectするので，$O(|x|)$時間で終ってしまう. これは自然な制限時間の条件(b) に反するが，Dの最悪時の効率を議論するには，制限時間としてn^2を使い，time_D(l) = $O(l^2)$と評価しても十分である.

例4.9. 制限時間 n^2 が条件(c)を満たすこと

入力列 x ➔ $O(|x|^2)$時間以内で出力$0^{|x|^2}$を出力.
以下に基本的なアイディアを示す(プログラムsq)

```
w1:=x # 0; y:= ε;  xは入力変数, yは出力変数
while w1≠ε do
  w1:=right(w1); w2:=x;
  while w2 ≠ε do                 このループで
    w2:=right(w2); y:=y # 0;     |y| ←|y|+|x|となる
  end-while
end-while;
```

入力の長さをlとすると,

内側のwhileループはl回　（1回あたり3ステップ）

外側のwhileループは$l+1$回

全体で　$2+(l+1)(3+3l) = 3l^2 + 6l + 5$

$$\mathrm{time\_sq}(l) = O(l^2)$$

定理4.2: 関数t_1, t_2を任意の自然な制限時間とする．このとき，
$t_1 + t_2$, $t_1 \times t_2$, $t_1 \circ t_2$ も自然な制限時間．

定理4.3: すべての制限時間t_1, t_2に対し，
$t_1 = O(t_2) \rightarrow \mathrm{TIME}(t_1) \subseteq \mathrm{TIME}(t_2)$.

証明:
すべてのLで，$L \in \mathrm{TIME}(t_1) \rightarrow L \in \mathrm{TIME}(t_2)$を示せばよい．
定義より，Lを$O(t_1)$で認識するプログラムAが存在．
つまり，time_A = $O(t_1)$.
$t_1 = O(t_2)$だから，
time_A = $O(t_2)$.
よって，Lの時間計算量は$O(t_2)$.
すなわち，$L \in \mathrm{TIME}(t_2)$.　　　　証明終