

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な研究

(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)

Chap.4 Computational Complexity

4.1. Survey on Theory of Computational Complexity

“Computable?” → “How much cost is required for computation?”

Computational Complexity Theory

- (1) Studies on upper bound of computational cost
- (2) Studies on lower bound of computational cost
- (3) Structural studies on hardness of computation

(1) Studies on upper bound of computational cost

Algorithm Theory: design of efficient algorithms

Suppose we have an algorithm A which solves a problem X in at most time $T(n)$ for any input of size n . Then, an upper bound on the time complexity of the algorithm A is $T(n)$.

(asymptotic worst case time complexity)

(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な研究

“ 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.

(2) Studies on lower bound of computational cost

If any algorithm for a problem X takes time $T(n)$ in the worst case, a lower bound on the time complexity of the problem X is $T(n)$.

- $\mathcal{P} \neq \mathcal{NP}$ conjecture
- Robustness of crypto system

(3) Structural studies on hardness of computation

Studies to characterize hardness in the level of “xx-hardness” hierarchical structure depending on the hardness

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, \dots, 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, \dots, 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 **A の x_1, x_2, \dots, x_k に対する計算時間**(略して $A(x_1, x_2, \dots, x_k)$ 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。

$time_A(x_1, x_2, \dots, x_k) \equiv A(x_1, x_2, \dots, x_k)$ の計算時間

$$time_A(l) \equiv \max \{ time_A(x_1, x_2, \dots, x_k) : \sum_{1 \leq i \leq k} |x_i| \leq l \}$$

4.2 Measuring Computation Time

4.2.1 Revisiting Programs in the Standard form

It consists of one while loop of

- one if + substitute to pc
- one basic states + sub. to pc
in each line

Definition 4.1 (Computation time)

A: program with k inputs in the standard form

x_1, x_2, \dots, x_k : inputs to A

Single execution of while loop in A is “**one step**” in A .

The number of iterations of the while loop required before A halts is called **the computation time of A for inputs x_1, x_2, \dots, x_k** (in short, computation time of $A(x_1, x_2, \dots, x_k)$).

If A does not halt, its computation time is infinite.

$time_A(x_1, x_2, \dots, x_k) \equiv$ computation time of $A(x_1, x_2, \dots, x_k)$

$time_A(l) \equiv \max\{time_A(x_1, x_2, \dots, x_k) : \sum_{1 \leq i \leq k} |x_i| \leq l\}$

標準形プログラム

```
prog プログラム名(input ...);
```

```
var pc:  $\Sigma^*$ ; ... ;  $\Sigma$ ; ... ;  $\Sigma^*$ ;
```

```
begin
```

```
  pc:=1;
```

```
  while pc  $\neq$  0 do
```

```
    case pc of
```

```
      1: (文);
```

各(文)の形は

```
      2: (文);
```

- if 比較文 then pc:= k_1 else pc:= k_2 end-if

```
      3: (文);
```

- 代入文; pc:= k ;

```
      .....
```

```
       $k$ : (文);
```

のいずれか.

```
    end-case
```

```
  end-while;
```

```
  halt( $\Sigma^*$ 型の変数);
```

```
end.
```

Programs in the standard form

```

prog program name (input ...);
var pc:  $\Sigma^*$ ; ... ;  $\Sigma$ ; ... ;  $\Sigma^*$ ;
begin
  pc:=1;
  while pc  $\neq$  0 do
    case pc of
      1: (statement);  Each statement must be either
      2: (statement);  if comparison then pc:= $k_1$  else pc:= $k_2$  end-if
      3: (statement);   or
      .....           substitution; pc:= $k$ ;
       $k$ : (statement);
    end-case
  end-while;
  halt(variable of type  $\Sigma^*$ );
end.

```


・各文が高々定数時間で実行できるための制約

$u, u': \Sigma$ 型の変数, $v, v': \Sigma^*$ 型の変数

$c: \Sigma$ 型の定数, $s: \Sigma^*$ 型の定数

(代入文) (1) $u := c$; (2) $u := u'$;

(3) $u := \text{head}(v)$; (4) $u := \text{tail}(v)$;

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

(7) $v := \text{right}(v)$; (8) $v := \text{left}(v)$;

(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.

- Constraints to execute each statement in constant time

u, u' : variable of type Σ , v, v' : variable of type Σ^*
 c : constant of type Σ , s : constant of type Σ^*

(Substitution)

- (1) $u := c$; (2) $u := u'$;
 (3) $u := \text{head}(v)$; (4) $u := \text{tail}(v)$;
 (5) $v := s$; ~~(6) $v := v'$; ??~~
 (7) $v := \text{right}(v)$; (8) $v := \text{left}(v)$;
 (9) $v := u \# v$; (10) $v := v \# u$;

(Comparison)

- (11) $u = c$ (12) $v = s$

- comparison of the form $v = v'$ is forbidden

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

4.2.2. Time complexity of a program

The time complexity of a program is represented as a *function of input size* (length of an input string)

Valid Encoding :

Encoding into *at most constant times* larger than the original.

Ex.4.5: Unary and binary representations

Binary representation is a valid encoding in the standpoint of “size of a number is its number of bits”, but unary one is redundant.

定義4.3: 自然数上の関数 f, g に対し,

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

となるとき, f は オーダー g であるといい, $f = O(g)$ 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) $\forall n [f(n) \leq g(n)] \rightarrow f = O(g)$

(2) $\exists c > 0, \forall n [f(n) \leq c g(n)] \rightarrow f = O(g)$

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

Definition 4.3: For functions f and g on natural numbers, if
$$\exists c, d > 0, \forall n [f(n) \leq c g(n) + d]$$
then we say f is in the order of g and denote it by $f = O(g)$.

Remark: the constants c and d must be determined independently of n .

Theorem 4.1: The followings hold for any functions f , g and h on natural numbers:

1. $\forall n [f(n) \leq g(n)] \rightarrow f = O(g)$
2. $\exists c > 0, \forall n [f(n) \leq c g(n)] \rightarrow f = O(g)$
3. $[f = O(g) \text{ and } g = O(h)] \rightarrow f = O(h)$

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,
$$\forall l [time_A(l) \leq ct(l) + d]$$
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.

4.2.3. Time complexity of a problem

Def.4.4. Let Φ be a computing problem and t be a function over natural numbers. If we have a program A to compute Φ and some constants c and $d > 0$ such that

$$\forall l [time_A(l) \leq ct(l) + d]$$

then we say that Φ is computable in $O(t)$ time, or time complexity of Φ is $O(t)$.

Notice: We assume here that a computing problem is that of recognizing a set.

Intuitively

problem Φ is computable within time t

- time complexity of A may be less than t .
- there may be a faster program to compute Φ than A does.

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 $\equiv \{ \lceil n \rceil : n \text{ は素数} \}$

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);      2 ~ n-1の数で割ってみる
begin
  for each i := 1 < i < n do
    if n mod i = 0 then reject end-if
  end-for;
  accept
end.

```

← $\log n \cdot \log i$ 時間

$$\begin{aligned}
 \text{time_Naive}(n) &\leq \sum_{1 < i < n} (c \log n \log i + d) \\
 &= c \log n \log n! + dn = O(n(\log n)^2)
 \end{aligned}$$

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

余談:
 2002年に
 $O(l^6)$
 のアルゴリズム
 が考案された!!

Ex.4.7. Time complexity of the problem determining primes

Prime-determining problem(PRIME)

Input: a natural number n (binary representation)

Question: Is n prime?

PRIME $\equiv \{ \lceil n \rceil : n \text{ is prime} \}$

Stirling's Formula:

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

```

prog Naive(input n);      try to divide by numbers between 2 – n-1
begin
  for each i:= 1 < i < n do
    if n mod i = 0 then reject end-if
  end-for;
  accept
end.

```

$\log n \cdot \log i$ time

$O(l^6)$ time algorithm has
been proposed in 2002!!

$$\begin{aligned}
 \text{time_Naive}(n) &\leq \sum_{1 < i < n} (c \log n \log i + d) \\
 &= c \log n \log n! + dn = O(n(\log n)^2)
 \end{aligned}$$

When the length of n is l , l is approximately $\log n$. So, $\text{time_Naive} = O(l^2 2^l)$. Thus, time complexity of PRIME is $O(l^2 2^l)$.

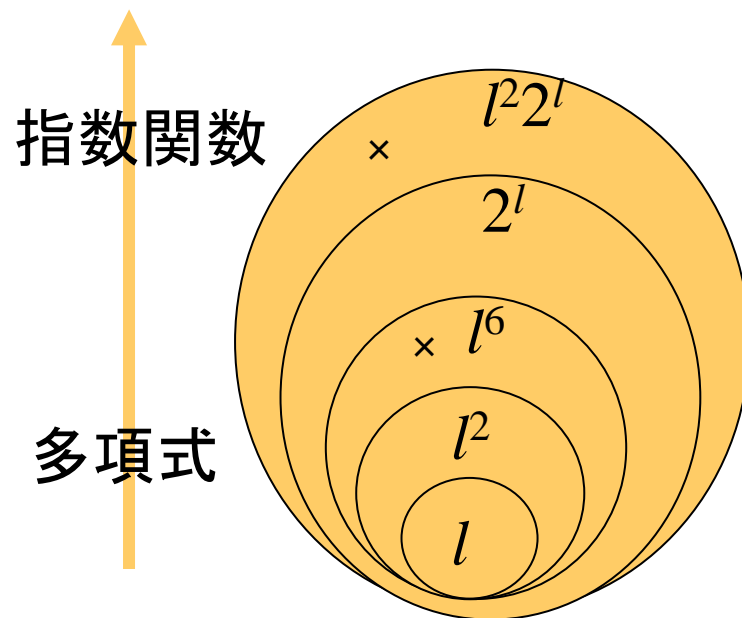
定義4.5.

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.

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

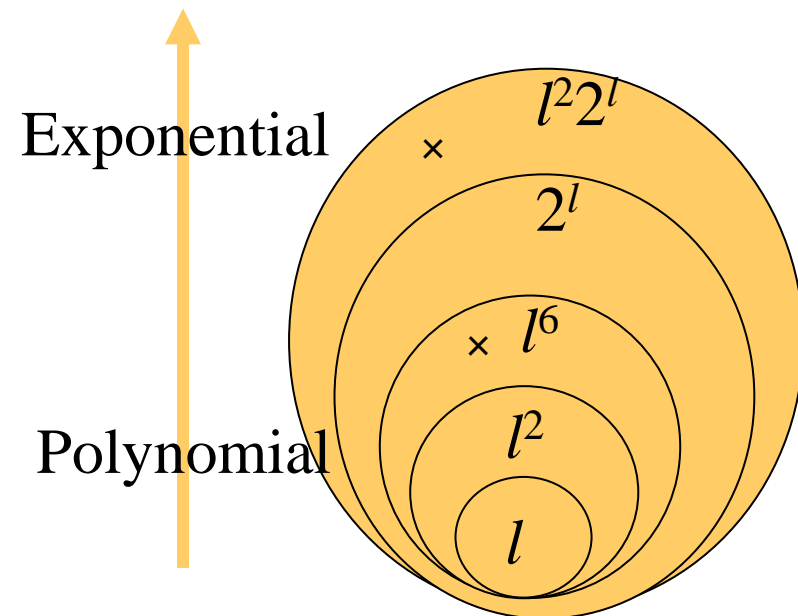
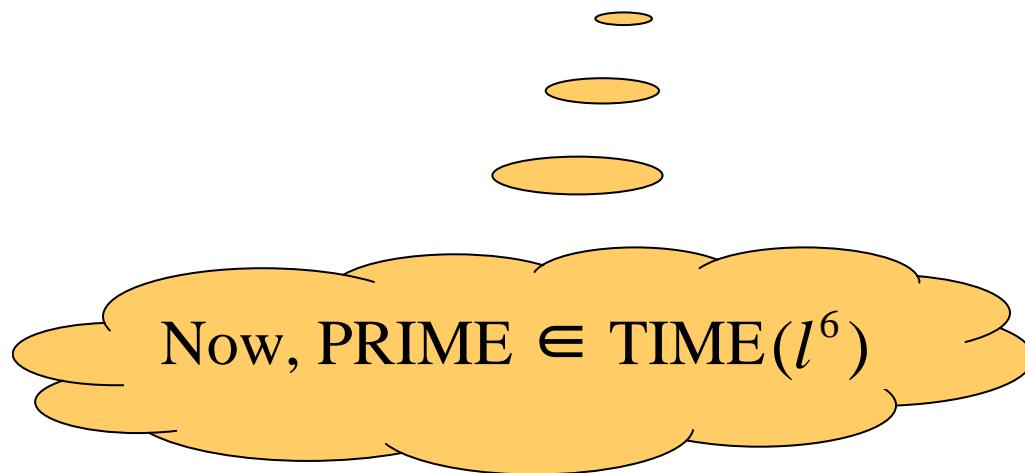


Def.4.5.

For a function t over natural numbers, the set of all sets (i.e. recognition problems) with time complexities $O(t)$ is called **$O(t)$ -time complexity class**, and it is denoted by **TIME(t)**. And such a function t is called a time limit.

For example, a class of sets recognizable in time $O(l^2 2^l)$ is TIME($l^2 2^l$), and the set PRIME is one element.

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



制限時間 t にふさわしい関数の条件

自然な制限時間(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).

Conditions for a function t appropriate to time limit

(Definition of) natural time limit

- (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) Given any input x , we can compute $\overline{t(|x|)}$ (unary representation of $t(|x|)$) in $O(t)$ time.

Condition (a): It takes n time to read input.

Condition (c): To allow us to use a timer to break computation at time $t(n)$.

(We decrement the unary representation).

例4.8: 集合 $D = \{ \langle a, b \rangle : a \text{ は } b \text{ で 割り 切 れ る} \}$ 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, 1); b:=get(x, 2);
  %  $x = \langle a, b \rangle$  の形でないときは, この時点でreject
  if  $a \bmod b = 0$  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 $x = \langle a, b \rangle$ の形の場合は $O(|x|^2)$ 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, $O(|x|)$ 時間で終わってしまう. 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.

Ex.4.8: Time complexity of a set $D = \{ \langle a, b \rangle : b \text{ divides } a \}$.

Consider the following program to recognize the set D .

```

prog D(input x);
begin
  a:=get(x, 1); b:=get(x, 2);
  % if x is not in the form of  $\langle a, b \rangle$ , reject immediately.
  if  $a \bmod b = 0$  then accept else reject end-if
end.

```

← computable in time $O(|a||b|)$

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

If an input is in the form $x = \langle a, b \rangle$, it takes $O(|x|^2)$ time.

Otherwise, we can reject it before computing mod, and thus it terminates in time $O(|x|)$. This contradicts to the condition (b) of a natural time limit, but to discuss about the worst case performance of D it suffices to evaluate it as $time_D(l) = O(l^2)$ by using n^2 as the time limit.

例4.9. 制限時間 n^2 が条件(c)を満たすこと

入力列 $x \rightarrow O(|x|^2)$ 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 $2+(l+1)(3+3l) = 3l^2 + 6l + 5$
 $\text{time_sq}(l) = O(l^2)$

Ex.4.9: Time limit n^2 satisfies the condition(c).

input string $x \rightarrow$ Output $0^{|x|^2}$ in time $O(|x|^2)$.

A basic idea is as follows: (program sq)

```

w1:=x # 0; y:= ε; x is an input variable, y is an output variable
while w1 ≠ ε do
  w1:=right(w1); w2:=x;
  while w2 ≠ ε do           In this loop
    w2:=right(w2); y:=y # 0;   |y| ← |y|+|x|
  end-while
end-while;

```

Let l be the length of an input.

inner while loop is iterated l times (3 steps per iteration)

outer loop is iterated $l+1$ times

in total $2+(l+1)(3+3l) = 3l^2 + 6l + 5$

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,
 $t_1 = O(t_2) \rightarrow \text{TIME}(t_1) \subseteq \text{TIME}(t_2)$.

証明:

すべての L で, $L \in \text{TIME}(t_1) \rightarrow L \in \text{TIME}(t_2)$ 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, $\text{time}_A = O(t_1)$.

$t_1 = O(t_2)$ だから,

$\text{time}_A = O(t_2)$.

よって, L の時間計算量は $O(t_2)$.

すなわち, $L \in \text{TIME}(t_2)$.

証明終

Theorem 4.2 Let t_1, t_2 be any natural time limits. Then, so are $t_1 + t_2$, $t_1 \times t_2$, and $t_1 \circ t_2$.

Theorem 4.3 For any time limits t_1 and t_2 , we have $t_1 = O(t_2) \rightarrow \text{TIME}(t_1) \subseteq \text{TIME}(t_2)$.

Proof:

It suffices to show $L \in \text{TIME}(t_1) \rightarrow L \in \text{TIME}(t_2)$ for any L .

By definition, there is a program A recognizing L in time $O(t_1)$.

That is, $\text{time}_A = O(t_1)$.

Since $t_1 = O(t_2)$, we have

$\text{time}_A = O(t_2)$.

Thus, the time complexity of A is $O(t_2)$.

Therefore, $L \in \text{TIME}(t_2)$.

End of Proof

Information

- 10月30日(火曜日)は中間テスト
 - 時間は11:00~12:30 (30分以上遅刻したら入室禁止)
 - 範囲は10月23日の授業分まで
 - テキスト、資料は持ち込み**禁止**
- Mid-term exam will be on October 30th, Tue.
 - Time: 11:00-12:30 (You cannot take it after 11:30)
 - About: up to October 23rd
 - Texts and other materials are **not allowed** to bring

Today's Office Hour:
Answer & Comments on
the Report (3)