I222:計算の理論

担当:上原隆平

テキスト:

「計算可能性・計算の複雑さ入門」

渡辺治著, 近代科学社

I222:Theory of Computation

- by Prof. Ryuhei UEHARA
- **Text:**
- **"Introduction to Computability and Computational Complexity"**

  by Osamu Watanabe, Kindai-Kagaku-sha

  (in Japanese)

1. 問題とアルゴリズム

1.1. 問題とは，アルゴリズムとは，そして手に負えない問題とは

問題とは何か

＝　関数の計算問題　：　入力　→ 出力

（数値計算だけでない）

ソート問題

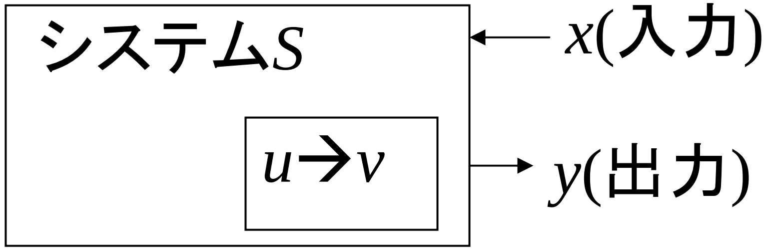
入力 : 自然数の列　$a_1, a_2, \ldots, a_n$

出力 : 入力列を小さい順に並べた列$a_{i1}, a_{i2}, \ldots, a_{in}$.

入力と出力が数学的に明確に定義されていること

~~出力 : 最高の料理法~~

例1.1. コンピュータシステムの働き



システムSの１サイクル模倣

入力 : 入力x と現在の状態 u

出力 : 出力y と次の状態 v

(x,u)を(y,v)に対応させる関数 f_S の計算問題

Chap. 1 Problems and Algorithms

1.1. What are problems and algorithms? Intractable problems?

Problem

＝ Problem of computing a function: input → output
(not only numerical computation)

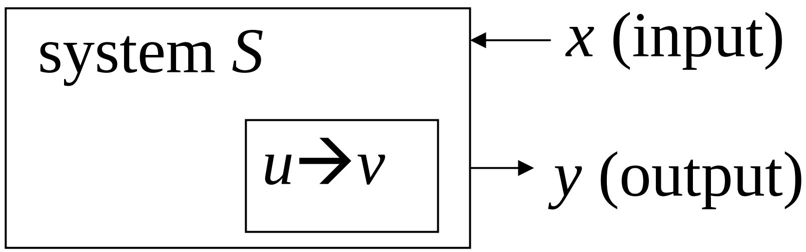
Sorting Problem

input: sequence of natural numbers $a_1, a_2, \ldots, a_n$
output: increasing order $a_{i1}, a_{i2}, \ldots, a_{in}$.

Input and output must be mathematically defined

~~output: the best recipe~~

Example: Performance of a computer system



simulation of 1 cycle of S
input: input x and current state u
output: output y and next state v

problem of computing a function to map (x,u) to (y,v)

仮定：どんな入力に対しても関数は何か値を返す
　たとえば，異常入力に対しては？を返す
　＝全域関数の立場

システムSが入力10を仮定していないなら$f_S(10)=?$と定義

問題を解くアルゴリズム(algorithm)

入力に対して問題が規定している出力を求める方法.

何を求めるか
如何に求めるか
の違い

2次方程式の根計算問題

入力：有理数a, b, c

出力：$ax^2+bx+c=0$を満たすxの一つ

出力が何かは明確だが，出力を求める方法は？

「アルゴリズム＝方法」

➔アルゴリズム＝プログラムとして実現できる計算方法

Assumption: system returns some value for any input

e.g. ? is returned for an abnormal input

＝standpoint of total function

We define $f_S(10)$=? if the system S cannot handle the input 10.

Algorithm for solving a problem

a method for computing an output specified in the problem.

What to be computed?

How to compute?

difference

Problem of calculating a root of a quadratic equation

input: rational number a, b, c

output: an x that satisfies $ax^2+bx+c=0$

Output is clearly defined but how can we find it?

"Algorithm = method"

➔algorithm＝a method that can be realized as a program

難しい問題とやさしい問題

→計算の複雑さ

前半の講義では，「計算不可能な問題」を扱う

手に負えない問題(intractable problems)

「計算可能性の理論」

「帰納関数論」

例1.2.　計算不可能な問題の例

停止問題（停止性判定問題）

入力：プログラムA（1入力）とそれへの入力 x

出力：A へ x を与えて実行させると停止するか?

停止するならYES，しないならNO.

この問題は計算不可能であることが証明できる

→後述

Hard and Easy Problems

→Complexity of Computation

First half of the lecture deals with "incomputable" problems

intractable problems

"Theory of Computability"

"Theory of Recursive Functions"

Example 1.2. Incomputable problems

Halting Problem（Problem of deciding halting）

input：a one-input program A and an input x

output: whether does it terminate if x is given to A?

YES if it terminates and NO otherwise.

We can prove that this problem is incomputable

→to be explained later

プログラムを作るのは“難しい”が，計算は“やさしい”問題

・コラッツの予想は正しいか？
入力：なし
出力：Yes か No

コラッツの予想：
どんな自然数でも
「偶数ならば2で割り、奇数ならば3倍して1を加える」ことを繰り返すと1になる

計算可能であっても難しい問題

・計算時間がかかり過ぎる
・多量のメモリーが必要である
・計算コストを考えた上での計算可能性
➔「計算の複雑さの理論」　→　後述

事実上計算不可能の基準

＝多項式時間で計算することが不可能
➔手に負えない問題
（多項式時間は手に負える問題の基準ではないことに注意）

否定的な結果を示すための基準

A problem that can be easily programmed but can be computed easily

- Is Collatz Problem true?
  - input: nothing
  - output: Yes or No.

Collatz Conjecture:
Iterating "÷2 if it is even, and ×3+1 if it is odd" always returns 1 for any positive integer.

Computable but hard problems

- too much time for computation
- too much space for storage
- computability based on computation cost
  - ➔"Theory of computational complexity" → later

Criterion on practical incomputability

＝impossible to be computed in polynomial time

➔intractable problems

（Note that polynomial time is not the criterion to be tractable.）

$\mathcal{NP}$完全問題

(1) 解を教えてもらえば，それが解の条件を満たしているか否かは簡単にチェックできる.

(2) しかし，解の候補数が（入力のサイズに関して）指数関数的に増大するので，「一つ一つの候補をチェックする」という単純なプログラムでは，時間計算量が指数関数的に増大してしまう.

1950年代から研究開始

例1.3.　箱詰め問題(bin packing problem)

・n 個の棒状の荷物：長さは $a_1, a_2, \ldots, a_n$

・長さがそれぞれ b の k 個の箱にうまく収まるように詰めることができるか?

単純な方法では，少なく見積もっても指数関数的な時間がかかってしまう.

$\mathcal{NP}$ Complete Problem

(1) Given a solution to the problem, it can be easily checked whether it satisfies the condition for solution.

(2) But, a simple program checking every solution candidate takes exponential time since the number of candidates grows exponentially.

The study starts in 1950's.

Example1.3. Bin packing problem

- n items of lengths $a_1, a_2, \ldots, a_n$
- Is it possible to pack all the items into k boxes of length b?

A simple algorithm takes at least exponential time.

$\mathcal{P} \neq \mathcal{NP}$予想

100万ドルの懸賞金つき!!

「$\mathcal{NP}$完全問題は多項式時間では解けない．」

例1.4. どんな多項式も指数関数よりは緩やかに増加する.

$p(n)$を任意の多項式，$e(n)$を任意の指数関数とすると

➔十分大きな n に対して，$p(n) << e(n)$ が成り立つ.

(例) 十分大きな n については $n^{10000} << 1.0000001^n$

定義1.2.

(1) その問題を解くプログラムが存在しない問題を，（計算不可能という意味で）手に負えない問題という.

(2) その問題を多項式時間以内で解くプログラムが存在しない問題を，（計算困難という意味で）手に負えない問題という.

One of $1000000 Millennium prize problems

$\mathcal{P} \neq \mathcal{NP}$ Conjecture

Any $\mathcal{NP}$ complete problem cannot be solved in polynomial time.

Example 1.4. Any polynomial function grows more slowly than an exponential function.

Let $p(n)$ be any polynomial function and $e(n)$ be any exponential function

➔for sufficiently large n, $p(n) << e(n)$.

e.g., $n^{10000} << 1.0000001^n$ for sufficiently large n.

Definition 1.2.

(1) A problem for which there is no program to solve it is called "intractable" in the sense that it cannot be computed.

(2) A problem for which there is no program to solve it in polynomial time is called "intractable" in the sense that it is hard to be computed.

1.2. 準備

1.2.1. 集合，関数，述語など

(1) 数

特に断らない限り，自然数(0を含む)のみを扱う.

x が実数のとき，$[x]$で x の整数部を表す（切り捨て）

(2) 集合

標準的な記号： $A \cup B, A \cap B, \overline{A}, A \subseteq B$

$A \times B$: A と B の要素の順序対全体の集合

$\|A\|$: 集合Aの要素数

原則として，大文字アルファベットで集合を表す．例外は

Γ　我々のプログラミング言語で文字として許される記号

Σ　$\{0, 1\}$

N　自然数の全体(0を含む)

1.2. Preparation

1.2.1. Set, function, predicate, and etc.

(1) **Number**

Only natural numbers (including 0) are considered.

$[x]$ represents the integral part of x (rounding off)

(2) **Set**

standard notations : $A \cup B, A \cap B, \overline{A}, A \subseteq B$

$A \times B$ = a set of all pairs of elements of A and B

$\|A\|$: number of elements of the set A

In principle, sets are denoted by capital letters.

Exception: Γ symbols used in programs

Σ {0, 1}

N a set of all natural numbers (including 0)

X: 任意の有限集合

X上の文字列＝Xの各要素を“文字”とみなし，その文字を有限個（0個を含む）並べて得られたもの

文字列の長さ＝文字列を構成する文字の数

$|x|$:　文字列 x の長さ

文字列 0100 の長さは4.

長さ0の文字列を空列といい，ε という記号で表す．

Σ^* : 0 と1を並べてできる文字列全体の集合（空列を含む）

辞書式順序（もどき）: 長さ優先の辞書式順序

x,y: Σ^* 上の文字列

$x<y$ $\Leftrightarrow$ (a) $|x| < |y|$, あるいは,

(b) 文字列x, yで最初に異なる文字をx_i, y_iとするとき

$x_i < y_i$.

(例) 101 < 0011 < 0100

通常の辞書式順序との違いは何か？

なぜ，このような順序を導入するのか？

X: any finite set

strings on X = a finite sequence of elements of X (each element of X is regarded as a "letter")

length of a string = the number of letters in the string

$|x|$: length of a string x

the length of a string "0100" is 4.

A string of length 0 is called an empty string, denoted by ε .

Σ^* : a set of all strings consisting of 0 and 1(including empty string)

(Pseudo-)Lexicographical Order:(with length preferred)

x, y: strings on Σ^*

$x<y \Leftrightarrow$ (a) $|x| < |y|$, or,

(b) for the first different letters in x and y be x_i, y_i,

$x_i < y_i$.

(example)101 < 0011 < 0100

What is the difference from usual lexicographical order?

What is the reason of introducing such an order?

論理記号

用例	意味
$P \wedge Q$	P かつ Q
$P \vee Q$	P または Q
$\neg P$	P でない
$P \rightarrow Q$	P ならば Q ($\neg P \vee Q$と同値)
$P \leftrightarrow Q$	P ならば Q かつ Q ならば P
$\exists x \in L[R(x)]$	L に属するある x で $R(x)$
$\forall x \in L[R(x)]$	L に属する任意の x で $R(x)$
$\overset{\infty}{\exists} x \in L[R(x)]$	$R(x)$ となる x が L の中に無限個ある
$\overset{\infty}{\forall} x \in L[R(x)]$	L の中の有限個を除いたすべての x で $R(x)$

演習:

$\overset{\infty}{\forall} x \in L[R(x)]$ ならば必ず $\overset{\infty}{\exists} x \in L[R(x)]$

だが、逆は真ではない。なぜか。

Logic symbols

example	meaning
$P \wedge Q$	P and Q
$P \vee Q$	P or Q
$\neg P$	not P
$P \rightarrow Q$	if P then Q
$P \leftrightarrow Q$	if P then Q and if Q then P
$\exists x \in L[R(x)]$	for some x in L, $R(x)$ holds
$\forall x \in L[R(x)]$	for any x in L, $R(x)$ holds
$\overset{\infty}{\exists} x \in L[R(x)]$	there are infinitely many x in L with $R(x)$
$\overset{\infty}{\forall} x \in L[R(x)]$	for any x except finitely many elements in L, $R(x)$ holds

Exercise:
$\overset{\infty}{\forall} x \in L[R(x)]$ implies $\overset{\infty}{\exists} x \in L[R(x)]$.
However, opposite direction is not true. Why?

命題論理式

命題変数と論理記号($\wedge, \vee, \neg$)から成る式

例： $F(X_1, X_2)=[X_1 \vee X_2] \wedge \neg X_1$

真偽値の割り当て

与えられた命題論理式の各命題変数に真偽値を代入すること.

上の例では,

(0,0), (0,1), (1,0), (1,1)

の4通りの割り当てが存在.（0：偽，1：真）

命題論理式の分類

リテラル： 命題変数あるいはその否定(記号は$\neg$)

和項： リテラルをOR(記号は$\vee$)でつないだ項

和積式： 和項をAND(記号は$\wedge$)でつないだ式

二和積式： 和積式の形の命題論理式で，しかも各和項がちょうど2個のリテラルからなるもの

三和積式： 和積式の形の命題論理式で，しかも各和項がちょうど3個のリテラルからなるもの

拡張命題論理式： 論理記号として,$\rightarrow$, $\leftrightarrow$ も許したもの

Propositional Logic Expression

Expression consisting of propositional variables and logic symbols $\wedge, \vee, \neg$ e.g. $F(X_1, X_2) = [X_1 \vee X_2] \wedge \neg X_1$

Truth assingnment

Assigning truth value to each propositional variable in each logic expression. e.g. there are 4 different assignments (0,0), (0,1), (1,0), (1,1) for the expression above. (0: false, 1: true)

Classification of propositional expressions

literal: logic variable or its negation

sum term: term in which literals are connected by OR

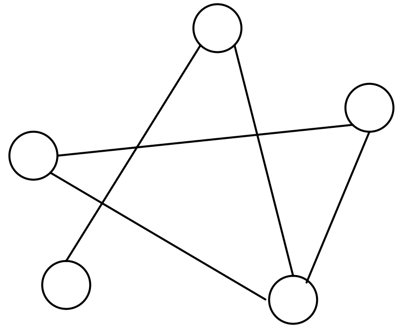
sum-multiply expression: expression in which sum terms are connected by AND

2-sum expression: logic expression in the sum-multiply form and each sum term consists of exactly two literals

3-sum expression: logic expression in the sum-multiply form and each sum term consists of exactly three literals

extended logic expression: one that may include $\rightarrow, \leftrightarrow$ as well.

グラフの表現



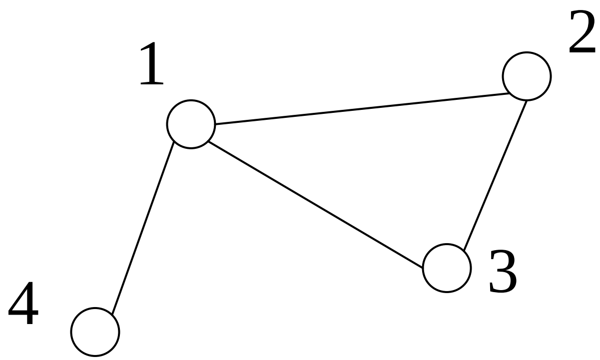
グラフの各頂点に1から順に番号をふる.

グラフの辺:　(i, j)

グラフの表現　$G = (n, E)$

　　　n: 頂点数,　E:辺の集合

無向グラフの例:

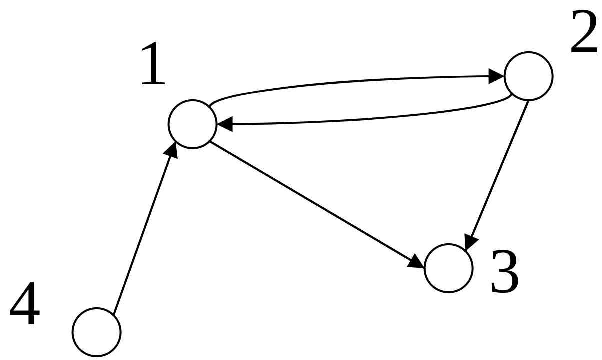


E={(1,2), (1,3), (2,3), (1,4)}

G = (4, {(1,2), (1,3), (2,3), (1,4)})

例えば(1,2)と(2,1)は区別しない

有向グラフの例:

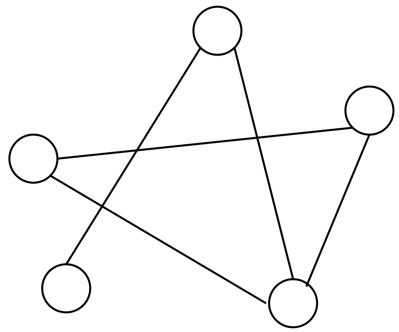


E={(1,2), (2,1), (1,3), (2,3), (4,1)}

G = (4, {(1,2), (2,1), (1,3),
(2,3), (4,1)})

辺の向きを区別する

Expression of a graph



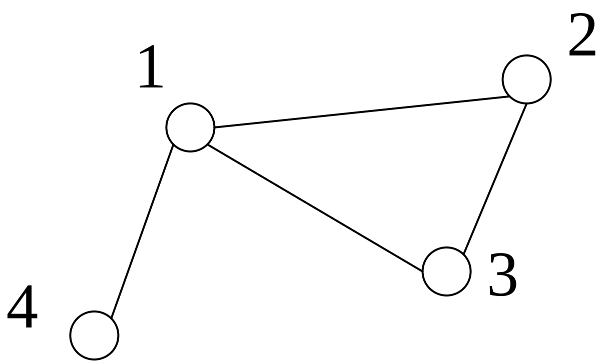
graph vertices are numbered sequentially

graph edge： (i, j)

expression of a graph $G = (n, E)$

n: number of vertices, E: set of edges

Example of a graph:

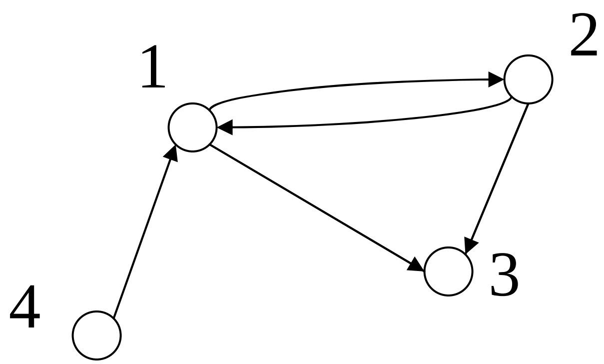


E={(1,2), (1,3), (2,3), (1,4)}

G = (4, {(1,2), (1,3), (2,3), (1,4)})

Do not distinguish (1,2) from (2,1)

Example of a directed graph:



E={(1,2), (2,1), (1,3), (2,3), (4,1)}

G = (4, {(1,2), (2,1), (1,3),
(2,3), (4,1)})

(1,2) and (2,1) are different arcs

1.2.2. アルゴリズムの記述方法

PASCAL風の手続き型プログラミング言語

例：2進表現で与えられた自然数を通常の自然数に変換

```
prog TR(input x: string on Σ): integer;
label LOOP;
var n: num; c: string;
%単にstringと型指定したときはstring on Γ型を意味する.
begin
    if x ≠ 0 ∧ head(x) = 0 then LOOP: goto LOOP: end-if;
    %2進表記でないものが入力されると無限ループに入る.
    n:=0;
   while x > ε do  % εは空列を表す定数
       c:=head(x);
       if c=1 then n:=2*n+1
                 else  n:=2*n end-if;
       x:=right(x)
    end-while;
    halt(n)
end.
```

1.2.2. Algorithm Description

PASCAL-like procedural programming language

Ex. Conversion from a binary natural number into an ordinary one.

```
prog TR(input x: string on Σ): integer;
label LOOP;
var n: num; c: string;
% string implies a type of string on Γ.
begin
    if x ≠ 0 ∧ head(x) = 0 then LOOP: goto LOOP: end-if;
    %if non-binary expression is input then goto infinite loop
    n:=0;
   while x > ε do  % ε is a constant for an empty string
       c:=head(x);
       if c=1 then n:=2*n+1
                 else  n:=2*n end-if;
       x:=right(x)
    end-while;
    halt(n)
end.
```

注意事項:

- 入出力に関する記述は省く.
- TR: プログラム名 ( )内が入力変数とその型指定,
  ( )の右が出力の型
- f_TR: プログラムTRが計算する(部分)関数
- 正常終了と無限ループ
  - 出力が得られるのはhalt文で正しく停止するときのみ.
  - 出力が得られない場合, プログラムが計算する関数値は<u>未定義</u>とみなす.

$$f\_\mathrm{TR}(001) = \perp$$

Remarks:

- description concerning input and output are omitted.
- TR: program name (input variable and its type declaration)
  the type of output follows
- f_TR: the (partial) function computed by the program TR
- normal termination and infinite loop
  - Output is obtained only when it terminates correctly by a halt sentence.
  - When an output is obtained, the function value computed by the program is considered as "undefined"

    $f\_\mathrm{TR}(001) = \perp$

変数の型

自然数型：　num型

文字列型：　string型

文字列を構成する“文字”として許される記号0, 1, 2, … , a, b, …

の全体を Γ とする.

文字列型データの基本演算

head(x)	x の最初の1文字
right(x)	x の2文字目から右の部分
tail(x)	x の最後の1文字
left(x)	x の先頭から最後の2文字目までの部分
$x \# y$	x と y の連接
$x \leq y$	長さ優先の辞書式順序による大小比較

ただし, head(ε)=right(ε)=tail(ε)=left(ε)= ε

Types of variables

natural number type: type num

string type:

Let Γ be a set of all symbols 0, 1, 2, ... , *a*, *b*, ... used in strings

Elementary operations on strings

head(x)	the first letter of x
right(x)	the part of x after its first letter
tail(x)	the last letter of x
left(x)	the part of x before its last letter
$x \# y$	concatenation of x and y
$x \leq y$	comparison based on lexicographic order with length preferred

where, head(ε)=right(ε)=tail(ε)=left(ε)= ε