2010 年度秋期

電子計算機概論 I (数学科)

計算機数学 (情報理工学科)

(担当: 角皆)

本講義の概要

「計算」を対象とする数学

- 「計算」の定式化
- 「計算」の実現
- 「計算」の量と質

→ 「計算する」とは?

本講義の概要

「計算」を対象とする数学

- 「計算」の定式化
- 「計算」の実現
- 「計算」の量と質

$\longrightarrow$ 「**計算する**」とは?

「計算する」とは?

「計算」

||

計算機で行なえること

では、計算機では何を行なっているのか?

「計算する」とは?

「計算」

||

計算機で行なえること

では、計算機では何を行なっているのか?

計算機

- アナログ計算機
  - ⋆ 図式計算 (計算図表・ノモグラム)
  - ⋆ 計算尺
- ディジタル計算機
  - ⋆ プログラム機構方式
  - ⋆ プログラム入力方式
  - ⋆ プログラム内蔵方式 (von Neumann 方式)
- 量子計算機

計算機

- アナログ計算機
  - ⋆ 図式計算 (計算図表・ノモグラム)
  - ⋆ 計算尺
- ディジタル計算機
  - ⋆ プログラム機構方式
  - ⋆ プログラム入力方式
  - ⋆ プログラム内蔵方式 (von Neumann 方式)
- 量子計算機

計算機

- アナログ計算機
  - ⋆ 図式計算 (計算図表・ノモグラム)
  - ⋆ 計算尺
- ディジタル計算機
  - ⋆ プログラム機構方式
  - ⋆ プログラム入力方式
  - ⋆ プログラム内蔵方式
    (von Neumann 方式)
- 量子計算機

計算機

- アナログ計算機
  - ⋆ 図式計算 (計算図表・ノモグラム)
  - ⋆ 計算尺
- ディジタル計算機
  - ⋆ プログラム機構方式
  - ⋆ プログラム入力方式
  - ⋆ プログラム内蔵方式
    (von Neumann 方式)
- 量子計算機

本講義の概要 (案)

- 「計算」の実現: 計算機
  - ⋆ 数の表し方・二進法・Boole 代数
  - ⋆ 計算機の実現
    論理回路・演算回路・順序回路
  - ⋆ 計算 (プログラム) の実際
    簡易アセンブラ

本講義の概要 (案)

- 「計算」の定式化
  - ⋆ 計算機のモデル化
    有限オートマトン・Turing 機械など
  - ⋆ 計算機の扱う言語・文法
    正規表現・生成文法・文脈自由言語など
  - ⋆ 計算可能性の理論
    普遍 Turing 機械と対角線論法

本講義の概要 (案)

- 「計算」の量と質

  ⋆ 計算量の理論
  多項式時間・“P＝NP?” 問題など

  ⋆ 幾つかの数理アルゴリズム
    ∗ Euclid の互除法
    ∗ 素数判定
    ∗ 素因数分解
    ∗ 高速 Fourier 変換　　など

既習事項 (かどうか) の確認

- 計算機での数の扱い方
  - ⋆ 二進法表示
  - ⋆ 桁溢れ判定
  - ⋆ 符号付き整数の表現 (“2 の補数表示”)
  - ⋆ Endian について
  - ⋆ 浮動小数点数 (指数部・仮数部)

既習事項 (かどうか) の確認

- **計算機の実現**
  - ⋆ **論理ゲート (NOT・OR・AND)**
  - ⋆ **Boole 代数**
  - ⋆ **演算回路 (加算機・multiplexer など)**
  - ⋆ **順序回路 (feedback 回路による状態保持)**
- **計算 (プログラム) の実際**
  アセンブラプログラミング

計算機での情報の表し方

計算機内では全てのデータを 0,1 の組 (列) で表す

情報の最小単位:

0 か 1 か : bit (binary digit)

実際には幾つかの bit を組にして一度に扱う

(通常は 8 bit $=$ 1 byte (octet))

1 byte で $2^8 = 256$ 通りの値を表せる

計算機での数の表し方

計算機内の 0,1 の列を、

二進表記 (二進法) で数値として扱う

正の整数の場合、1 byte で

$$0 \leq x \leq 2^8 - 1 = 255$$

の範囲の値が表せる

十進表記 $\longleftrightarrow$ 二進表記の変換

十進	二進
0	**00000000**
1	**00000001**
2	**00000010**
3	**00000011**
4	**00000100**
5	**00000101**
⋮	⋮
253	**11111101**
254	**11111110**
255	**11111111**

二進の九九 (一一?) の表

+	0	1
0	0	1
1	1	10

×	0	1
0	0	0
1	0	1

文字の表し方

符号化 (coding) して数値 (文字番号) で表す (文字コード)

アスキーコード (ASCII code) 対応表

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
2		!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

計算の理論

計算機に於ける「計算」の各ステップ
(＝ 命令の実行) は、

計算機内の記憶素子 (メモリ) の
現在の値 (状態) に従って、

その値を変更 (書込) すること

計算の理論

プログラム内蔵方式 (von Neumann 型) では、

プログラム・データを区別なくメモリ上に置くが、

プログラムとデータとは、やはり本質的に違う

- **プログラム: 一つの問題では固定**
- **データ: 可変な入力**

⇓

どんな (有効な) データ (入力) が来ても、
所定の出力を返すことが要請される

計算の理論

プログラム内蔵方式 (von Neumann 型) では、

プログラム・データを区別なくメモリ上に置くが、

プログラムとデータとは、やはり本質的に違う

- **プログラム: 一つの問題では固定**
- **データ: 可変な入力**

どんな (有効な) データ (入力) が来ても、
所定の出力を返すことが要請される

計算の理論

プログラム内蔵方式 (von Neumann 型) では、

プログラム・データを区別なくメモリ上に置くが、

プログラムとデータとは、やはり本質的に違う

- **プログラム: 一つの問題では固定**
- **データ: 可変な入力**

どんな (有効な) データ (入力) が来ても、
所定の出力を返すことが要請される

計算の理論

或る問題の「計算が可能」

⇕

その計算を行なうプログラムが存在

⇓

計算機の機能 (＝「計算」のモデル) を決めて議論

⟶ 代表的な「計算のモデル」を幾つか紹介

計算の理論

或る問題の「計算が可能」

$\Updownarrow$

その計算を行なうプログラムが存在

$\Downarrow$

計算機の機能 (＝「計算」のモデル) を決めて議論

$\longrightarrow$ **代表的な「計算のモデル」を幾つか紹介**

計算の理論

或る問題の「計算が可能」

$\Updownarrow$

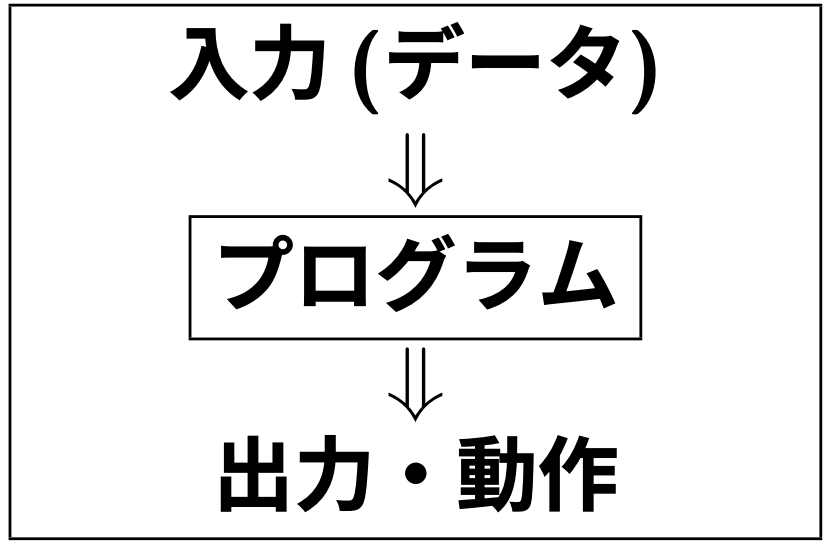
その計算を行なうプログラムが存在

$\Downarrow$

計算機の機能 ($=$「計算」のモデル) を決めて議論

$\longrightarrow$ **代表的な「計算のモデル」を幾つか紹介**

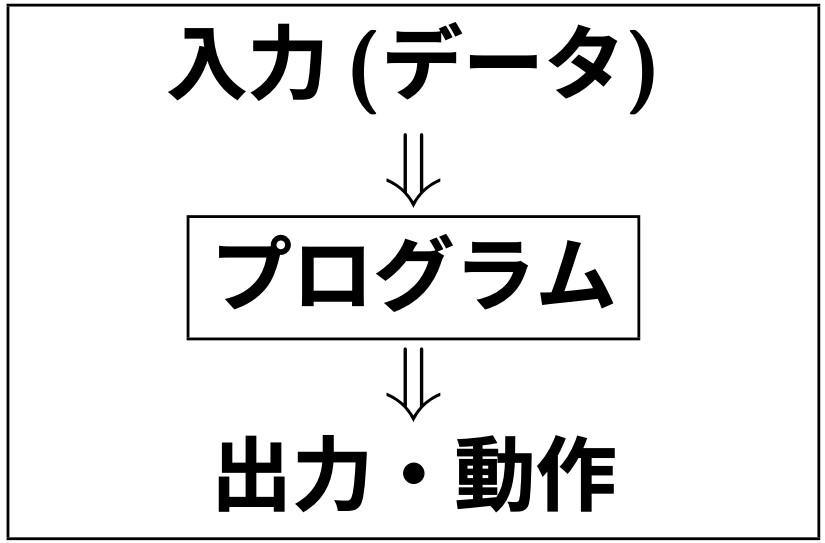
問題を「計算する」とは



原理・理論を考える際には、
出力は最も単純に「0 か 1 か」とする

- 0：拒否 (reject)
- 1：受理 (accept)

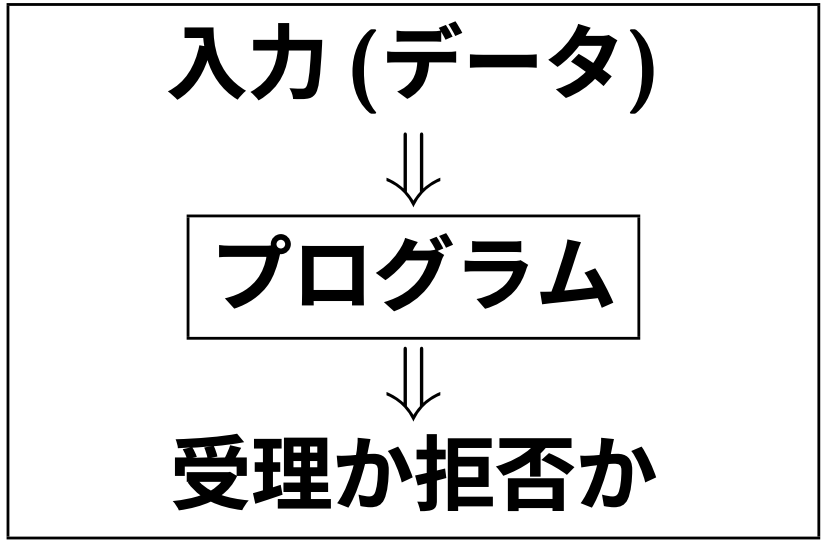
問題を「計算する」とは



原理・理論を考える際には、
出力は最も単純に「0 か 1 か」とする

- 0：**拒否** (reject)
- 1：**受理** (accept)

「問題」とは



解くべき「問題」：入力を受理する条件

「問題」の例

入力の範囲: 文字 a, b から成る文字列

「問題」: 入力を受理する条件

- **a と b との個数が同じ**
- **a が幾つか続いた後に b が幾つか続いたもの**
- **a で始まり a, b が交互に並んで b で終わる**
- **同じ文字列 2 回の繰返しから成る**
- **回文 (palindrome)**

などなど

「問題」とは

それぞれの「問題」に対し、
定められた計算モデルで、
受理／拒否判定が可能 (問題が解ける) か ？

受理される文字列が
「文法的に正しい」文字列だと思えば、

「問題」とは「文法 (言語)」である

「文法的に正しい」かどうかの判定
. . . 「構文解析 (syntactic analysis)」

「問題」とは

それぞれの「問題」に対し、
定められた計算モデルで、
受理／拒否判定が可能 (問題が解ける) か ？

受理される文字列が
「文法的に正しい」文字列だと思えば、

「問題」とは「文法 (言語)」である

「文法的に正しい」かどうかの判定
. . . 「構文解析 (syntactic analysis)」

「問題」とは

それぞれの「問題」に対し、
定められた計算モデルで、
受理／拒否判定が可能 (問題が解ける) か ？

受理される文字列が
「文法的に正しい」文字列だと思えば、

「問題」とは「文法 (言語)」である

「文法的に正しい」かどうかの判定
. . . 「構文解析 (syntactic analysis)」

代表的な計算モデル

- 有限オートマトン (有限状態機械)
- プッシュダウンオートマトン
- チューリングマシン　　など

チューリングマシン

無限 (非有界) のメモリにランダムアクセスできる計算機モデル

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 (algorithm)」の定式化

万能チューリングマシン

プログラム内蔵方式 (von Neumann 型)
$\cdots$ プログラムもデータとして保持
$\longrightarrow$ 一つの機械で様々な計算を柔軟に実現

同様の働きをするチューリングマシンが存在

$\cdots$ 万能チューリングマシン
(universal Turing machine)

全てのチューリングマシンの動作を模倣する

計算可能性の理論

チューリングマシンは、
どんな問題 (言語) でも計算できるのか?

「計算できる」とは?

- 認識する：
  正しければ受理 (そうでなければ受理しない)
- 判定する：
  正しければ受理、そうでなければ拒否

(認識可能だが) 判定不可能な問題が存在する!!

計算可能性の理論

チューリングマシンは、
どんな問題 (言語) でも計算できるのか?

「計算できる」とは?

- **認識する：**
  正しければ受理 (そうでなければ受理しない)
- **判定する：**
  正しければ受理、そうでなければ拒否

(認識可能だが) 判定不可能な問題が存在する!!

計算可能性の理論

チューリングマシンは、
どんな問題 (言語) でも計算できるのか?

「計算できる」とは?

- **認識する：**
  正しければ受理 (そうでなければ受理しない)
- **判定する：**
  正しければ受理、そうでなければ拒否

(認識可能だが) 判定不可能な問題が存在する!!

計算可能性の理論

チューリングマシンは、
どんな問題 (言語) でも計算できるのか?

「計算できる」とは?

- **認識**する：
  正しければ受理 (そうでなければ受理しない)
- **判定**する：
  正しければ受理、そうでなければ拒否

(認識可能だが) 判定不可能な問題が存在する!!

計算量の理論

問題の難しさを如何に計るか ？

$\longrightarrow$ (計算モデルを固定して)
解くのに掛かる資源の分量で計る
$\cdots$ 計算量 (complexity)

- 時間計算量: 計算に掛かるステップ数 (手間)
- 空間計算量: 計算に必要なメモリ量 (場所)

計算量の理論

問題の難しさを如何に計るか ？

$\longrightarrow$ (計算モデルを固定して)
解くのに掛かる資源の分量で計る
$\cdots$ 計算量 (complexity)

- **時間計算量: 計算に掛かるステップ数 (手間)**
- **空間計算量: 計算に必要なメモリ量 (場所)**

計算量の理論

計算量はアルゴリズム (計算方法) によって変わる

$\cdots$ 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

$\longrightarrow$ 問題そのものの難しさ の評価

計算量の理論

計算量はアルゴリズム (計算方法) によって変わる

$\cdots$ 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

$\longrightarrow$ 問題そのものの難しさ の評価

計算量の理論

入力データが大きくなれば計算量も増える

$\longrightarrow$ 入力データ長に対する増加のオーダーで表す
(Landau の O-記号)

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”

計算量の理論

入力データが大きくなれば計算量も増える

$\longrightarrow$ 入力データ長に対する増加のオーダーで表す
(Landau の O-記号)

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”

計算量の例

- **加法:** $O(n)$
- **乗法:** $O(n^2)$ **かと思いきや** $O(n \log n \log \log n)$
  (高速 Fourier 変換 (FFT))
- **互除法:** $O(n^3)$ **(FFT で** $O(n^2 \log n \log \log n)$**)**
- **素数判定: 多項式時間 P**
- **素因数分解: 多項式時間 P かどうか判っていない**

計算量の例

- **加法:** $O(n)$
- **乗法:** $O(n^2)$ **かと思いきや** $O(n \log n \log \log n)$ **(高速 Fourier 変換 (FFT))**
- **互除法:** $O(n^3)$ **(FFT で** $O(n^2 \log n \log \log n)$**)**
- **素数判定: 多項式時間 P**
- **素因数分解: 多項式時間 P かどうか判っていない**

計算量の例

- **加法:** $O(n)$
- **乗法:** $O(n^2)$ **かと思いきや** $O(n \log n \log \log n)$
  (高速 Fourier 変換 (FFT))
- **互除法:** $O(n^3)$ **(FFT で** $O(n^2 \log n \log \log n)$**)**
- **素数判定: 多項式時間 P**
- **素因数分解: 多項式時間 P かどうか判っていない**

計算量の例

- **加法:** $O(n)$
- **乗法:** $O(n^2)$ **かと思いきや** $O(n \log n \log \log n)$
  (高速 Fourier 変換 (FFT))
- **互除法:** $O(n^3)$ **(FFT で** $O(n^2 \log n \log \log n)$**)**
- **素数判定: 多項式時間 P**
- **素因数分解: 多項式時間 P かどうか判っていない**

“非決定性” 計算量

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)