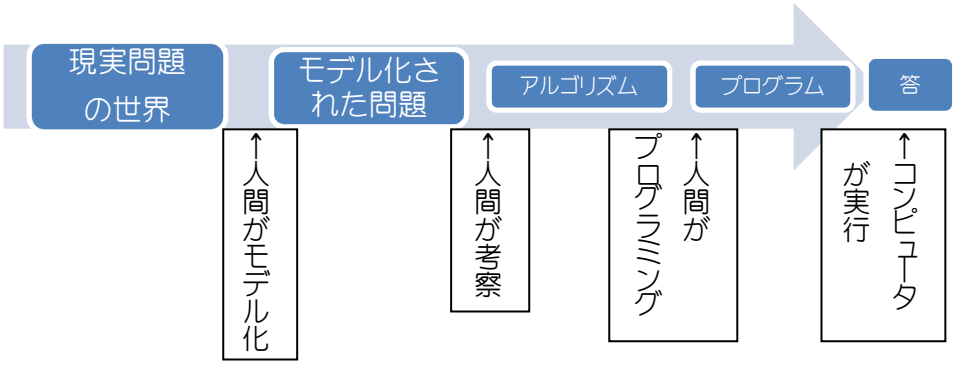
第 6 章 問題の解き方

6.1 アルゴリズム

6.1.1 アルゴリズムの役割

└ プログラムになる前の段階の計算手順のこと

☆問題解決の手順



☆アルゴリズムの重要性

•性能に大きな違いが出る･･･同じ問題を解く複数のアルゴリズムがある

→アルゴリズムによって計算時間が桁違いに変わることがある

•類型化されている･･･まったく違う問題を解くアルゴリズムが同じものになることがある

性能に関する考察•プログラミングを共通化できる

＊古いアルゴリズム

•ユークリッドの互除法（最大公約数を求める）

•ドント方式（比例代表選挙での議席配分）

6.1.2 アルゴリズムの実例

問題	① 平方根の計算	② フィボナッチ数の計算
アルゴリズム	(a) 反復法 (b) 二分法	(a) 再起法 (b) メモ化法

① 平方根の計算

小数点の計算は有限の精度で行われる→近似値しか求められない

問題例 ある正の実数xが与えられたときに、二乗するとxに近くなる正の実数 y を精度δで求める。

→$|\sqrt{x}-y|<\delta$ となるような y を一つ求める。

(a) 反復法

Y を 0 から δ ずつ足していって解の候補を順に検討する。$(y+\delta)^2>x$ となったら、その前の候補が解。

アルゴリズム（反復法による平方根の計算） x の平方根を精度δ で求める

```
y ← 0
while(y + δ)^2 < x  do
  y ← y+δ
done
return y
```

＊while～do　done
～ならば続く作業をし、～でなければ done の次の作業に移る
＊return～…～が解である

☆アルゴリズムの速度…作業の繰り返しの回数で比べる

・大ざっぱな近似だが、コンピュータの性能と無関係に検討できる。

・反面、異なる種類の計算の速度差も無視してしまう。

☆反復法の場合

・繰り返し回数は $\frac{\sqrt{x}}{\delta}$ 回　　・精度を1桁増やすと回数は10倍に増える

(b) 二分法

解の存在範囲に注目し、範囲を半分ずつに減らしていく。

アルゴリズム（二分法による平方根の計算）　x の平方根を精度δ で求める（x＞1）

```
a ← 0
b ← x
while b－a＞δ do
    c ← (a+b)/2
    if c^2 ＞x then b ← c  else a ← c endif
done
return a
```

＊区間(a,b)が解の存在範囲
c を区間の中央とする

☆二分法の場合

・1回繰り返すごとに区間の幅が半分になるので、n回後の区間の幅は $\frac{x}{2^n}$

→これがδ以下になるのに要する回数は、約 $\log_2 \frac{x}{\delta}$ 回

☆2 の平方根を小数点10桁まで求めると、反復法では約141億回、二分法では35回で済む。

② フィボナッチ数の計算　（必修・選択範囲ではないが、授業で扱っていたので軽く紹介）

(c) 再帰的アルゴリズム

このアルゴリズムでは、あるフィボナッチ数を求めるために、より小さなフィボナッチ数の計

算を何度も行う必要がある。

Ex)f(4)を求めるのに8回、f(5)を求めるのに13回、f(6)を求めるのに21回の計算を行う。

(d) メモ化アルゴリズム

前に計算したものを次の計算で使用する場合、前の計算結果を記録しておき、それを使って計算するほうが計算量は少なくてすむ。

Ex)f(4)を求めるのに6回、f(5)を求めるのに8回、f(6)を求めるのに10回の計算を行う。

6.1.4 計算量

☆**計算量＝アルゴリズムの計算時間**、

計算量を見積もることでよいアルゴリズムが選べ、プログラムの計算時間が予測できる。

☆計算量の見積もり方

① 問題の大きさを変数で表し　　Ex)n個のデータを処理する

② 計算の回数を式であらわす　　Ex)3n＋8回、$5\log_2(n+1)$回

③ 大まかな尺度である「オーダー(O)」を使う　　Ex)$O(n)$回、$O(\log n)$回

Point　定数は無視し、各変数について1番変化の大きい項だけを残す♪

問題例　平方根の計算―精度δで x の平方根を求める

・問題の大きさ　x とδ

・計算量　(a)反復法アルゴリズム　$O(\frac{\sqrt{x}}{\delta})$　　(b)二分法アルゴリズム　$O(\log\frac{x}{\delta})$

問題例　フィボナッチ数の計算―n番目のフィボナッチ数の求める

・問題の大きさ　n

・計算量　(c)再帰アルゴリズム　$O(2^{\frac{n}{2}})$　　(d)メモ化アルゴリズム　$O(n)$

・オーダーの大きさは　$O(\log n)<O(n^2)<O(2^n)$ となる。

6.2 計算のモデル化

6.2.1 機械的な計算モデル

計算のモデル化により、アルゴリズムの効率や、そもそもコンピュータで解ける問題かどうかなどを考える。

(a) 有限状態機械

・内部に状態を持ち、入出力が行える機械を単純化したもの

＝現実のコンピュータで使われる電子回路を単純化したもの

→現実のコンピュータより計算能力が低い(記憶装置が矮小)

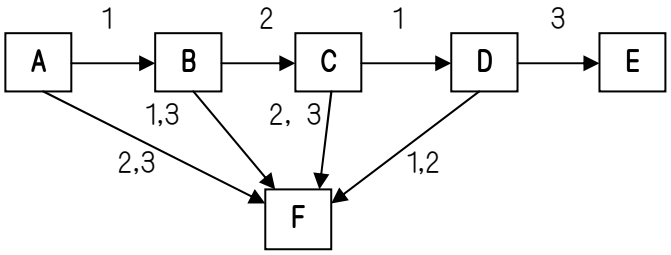
•機能•••信号を入力する(数個のボタン)、　信号を出力する(yes/no のみ)

•動き方•••入力＆現在の状態によって、出力＆次の状態を決める

例 電子錠

入力•••ボタン1,2,3　　出力•••鍵の開閉　　働き•••ボタンを「1213」とおせば解錠

A 最初
B 「1」と押された
C 「12」と押された
D 「1」と押された
E 「1213」と押された
F その他の状態



アルゴリズムで示すと

状態 ← 最初の状態
while 状態≠最終状態 **do**
　ボタン ← 押されたボタン
　図から(状態、ボタン)に対応する次状態を引く
　状態 ← 次状態
done

第7章 コンピュータの仕組み

7.1 計算の実現機構

7.1.1 コンピュータの基本構成

☆コンピュータの扱う情報

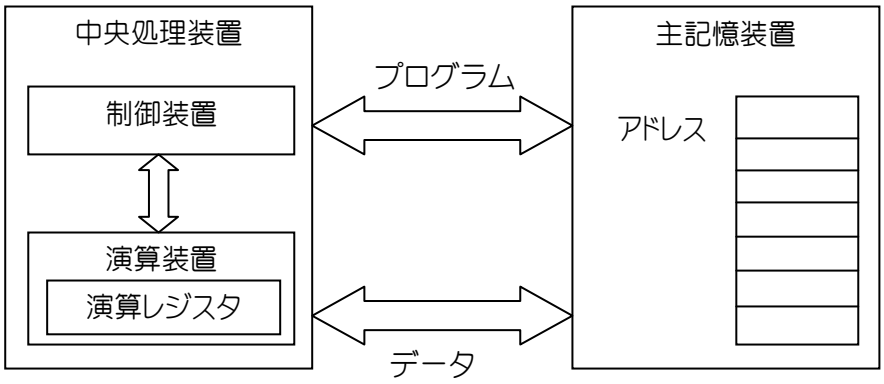
- **プログラム**…コンピュータの実行する計算処理手順にかかわる情報
- **データ**…コンピュータが処理する対象の情報

→**プログラム内蔵方式**(**フォンノイマン型コンピュータ**)

- メモリ上にプログラムとデータを保持
- プログラムに従い計算を進める

☆コンピュータの基本構成

- **中央処理装置**(CPU)
  - …これを1つの半導体集積回路として実現したもの＝**マイクロプロセッサ**(MPU)
  - **演算装置**…データに対して演算処理を行う、**演算レジスタ**がデータを保持
  - **制御装置**…主記憶装置と演算装置を制御する
- **主記憶装置**(メインメモリ)
  - …情報(データ)の格納と選択的な読み書き
  - アドレスによってデータの位置を特定



7.1.2 機械語レベルのプログラム例

- 機械語レベルのプログラム＝コンピュータに対する命令集合の並び
- **命令集合**…CPU が利用できる単純な命令群の集合で、メモリと演算レジスタを操作する　(↓表)
  - ☆**命令コード**…命令の種類をあらわす符号　　Ex)Load A の Load
  - ☆**オペランド**…命令の付加情報　　Ex)Load A のアドレス A
- 実際の機械語プログラム(**バイナリプログラム**)＝ビットパターン
  電圧の高低や電流の On/Off に対応
- **アセンブリ言語**…命令列をビットパターンからなるバイナリプログラムに変換
- **コンパイラ**…高水準言語で書かれたプログラムをバイナリプログラムに変換

命令集合の例

種類	命令	意味
データ転送命令	load A	アドレス A のデータを演算レジスタに読み込む
	store A	演算レジスタのデータをアドレス A に読み込む
演算命令 (算術演算や論理演算を実行)	add A	アドレス A のデータを演算レジスタの値に加える
	subtract A(sub A)	アドレス A のデータを演算レジスタの値から引く
	multiple A(mul A)	アドレス A のデータを演算レジスタの値にかける
分岐命令	jump A	アドレス A にプログラムの実行を移すー☆
	jumpzero A(JZ A)	演算レジスタのデータが 0 の場合、☆
	jumpminus A(JM A)	演算レジスタのデータが負の場合、☆
その他	write	演算レジスタのデータを出力する
	halt	プログラムの実行を停止する

(課題でやったと思うので例は割愛します。わからなかったら聞いてください。)

7.2 論理演算と組み合わせ回路

7.2.1 真理値表と論理関数(完備性の証明を除く)

7.2.2 ブール代数

7.2.3 MIL 記法

・**真理値表**…2 進演算の入出力の関係を表に示したもの

2 進演算の各桁の加算は 2 入力 2 出力(1ビット半加算器)ないし 3 入力 2 出力(1ビット全加算器)ととらえられる

→ n入力 1 出力の論理関数は 2^{2^n} 種類

・**ブール代数**…真理値表と同値、いかなる演算についても唯一に定まる形式(標準形)が存在

☆**加算標準形**(積和標準形)…n個のリテラルからなる論理積の論理和

☆**乗算標準形**(和積標準形)…n個のリテラルからなる論理和の論理積

＊**リテラル** ： 単一の変数または単一の変数にビット反転のかかったもの

・**論理関数** (「」内はブール代数表現)

☆NOT(否定)…1 入力を反転したものを出力 「$\bar{x}$」

☆AND(論理積)…2 入力がともに 1 であれば 1 を出力し、それ以外の場合 0 を出力 「・」

☆OR(論理和)…2 入力がともに 0 であれば 0 を出力し、

x	0	0	1	1
y	0	1	0	1
AND(x,y)	0	0	0	1
OR(x,y)	0	1	1	1
NOT(x)	1	1	0	0

それ以外の場合１を出力　「＋」

☆**NAND**(否定論理積)･･･NAND(x,y)=NOT(AND(x,y))

☆**XOR**(排他的論理和)　「⊕」･･･$x\cdot\bar{y}+\bar{x}\cdot y=(x+y)\cdot(\bar{x}+\bar{y})$

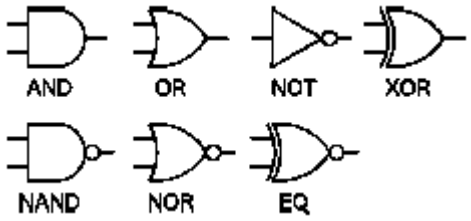
☆**NOR**(否定論理和)

☆**EQ**(等値)　「⊙」･･･$x\cdot y+\bar{x}\cdot\bar{y}=(x+\bar{y})\cdot(\bar{x}+y)$

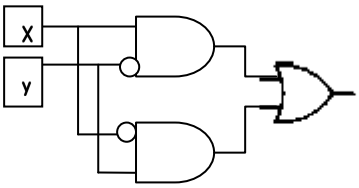
･完備性･･･　2^{2^n} 通りのすべての論理関数を実現できること

完備性を備えた演算としては、{AND,OR、NOT}の組み合わせ、単独の NAND、NOR がある

･MIL 記法

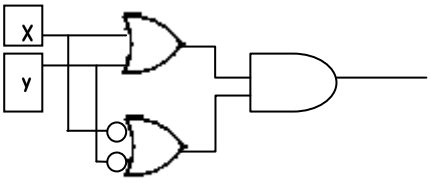


○が NOT を表している。



XOR の加算標準形

XOR(x,y)=OR(AND(x,NOT(y)),AND(NOT(x),y))



XOR の乗算標準形

XOR(x,y)=AND(OR(x,y),OR(NOT(x),NOT(y)))