1 章は短いので自分で読んだほうがわかりやすいと思います。本全体のまとめになっているので多分一回は読んどくべき場所ですし。もし希望があれば 1 章もまとめます。

# 第 2 章　情報の表現 - 記号･符号化

## ２．１　情報の表現

- ○ 情報を表現する言語には、自然言語、人工言語がある。
- ○ 情報の説明の仕方には、手続き的表現、宣言的表現がある
- ○ 情報の表現のされ方には、記号表現、パターン表現、ディジタル/アナログによる表現の違い、情報量による違いなどがある。
- ○ 目的によって，構築するモデルの表現形式が異なる。
- ○ モデルの表現形式の例：表(table)、図、グラフ、ラベル付きグラフ
- ○ グラフは道路ネットワーク/組織図/pert 図/意味ネットワークなど様々な領域で幅広く用いられる。

## ２．２　記号と表現

- ○ 情報表現の受け手側は、情報表現を適切に理解･解釈しなければならない。
- ○ 情報表現のデザイン側は、目的に応じて適切な表現手段を選択しなければならない。
- ○ 対象：表現の対象となる事物/事象を明確にする必要がある。
- ○ 目的：表現されている/する目的を理解する。
- ○ 方法：表現に関わるコストや目的に照らして，よりよい方法を選択する必要がある。
- ○ 記号表現の実際の形式には、図記号(ピクトグラム)、数の表現などがある。
- ○ 記号が表す 2 側面は、意味されるもの(シニフィエ)と、意味するもの(シニフィアン)
- ○ パターン表現は常に具体的/直接的であればいいわけではない。
- ○ 提喩の例：コンピュータの GUI におけるアイコン。ゴミ箱＝「ゴミを捨てる」という行為の隠喩。
- ○ 記号表現と命題の対応付けは恣意的である。○＝肯定、×＝否定とは限らない。
- ○ 情報表現のデザイナーは受け手側の解釈の枠組みに注意を払う必要がある
- ○ 日本語文字コードは現在，JIS,シフト JIS,EUC などの異なった日本語文字コードが混在している
- ○ 解釈の枠組みが異なれば記号の意味が異なってしまう。
- ○ ↑の例：プログラムが想定するコード体系が異なると、「文字化け」が起こる
- ○ コード体系の標準化･統一化が重要だが、困難も多い
- ○ 情報表現のデザイナーは情報表現間のトレードオフを考慮する必要がある。
- ○ コンピュータでの数の表現は「0」と「1」の 2 種類の記号を用いたビット列での表現
- ○ 16 ビットのシステムは 0～65535(=$2^{16}$)までを表現できる

## ２．３　アナログとディジタル

- ○ アナログ表現は無限の精度を必要とする。複製は元のデータの近似。
- ○ デジタル表現は複製時にデータが劣化しにくい。情報コンテンツの著作権保護への問題をもたらす
- ○ アナログ量をディジタル量に変換する際には情報を離散化する間隔を選択し、表現する必要がある
- ○ アナログ量をディジタル量に変換する際には，量子化と標本化が必要になる。
- ○ コンピュータディスプレイ装置は赤(R)緑(G)青(B)を混色した RGB 形式を用いている。
- ○ 音楽 CD は量子化のために 16 ビットを用いてい、音の振幅を $2^{16}$ の段階に分割している。
- ○ 標本化の間隔はデータの利用目的により変わる
- ○ エイリアシングの例として、モアレ縞がある
- ○ 音楽 CD の標本化：人間の鑑賞が目的なので聴覚で知覚できない高い周波数まで記録する必要はない
- ○ 適切な細かい標本間隔を用いれば，アナログ量を欠損なくディジタル量に処理できる
- ○ フーリエ変換は音声や画像などの情報表現と圧縮に用いる。画像の低周波成分から高周波成分へと足し合わせていったもので、復元に必要な情報量が分かり、データ量を圧縮できる

標本化定理(シャノン)

標本化の対象となるアナログ量 F が周波数の異なる複数の周期関数の重ね合わせで表現できる事を基本にするとき、周期関数(周期 T,周波数 $\omega$=1/T)の周波数が W 以下であるとすると、1/2W 間隔で標本化すれば，元のアナログ関数 F を復元できる。もっと詳しくは教科書 P.25 参照。

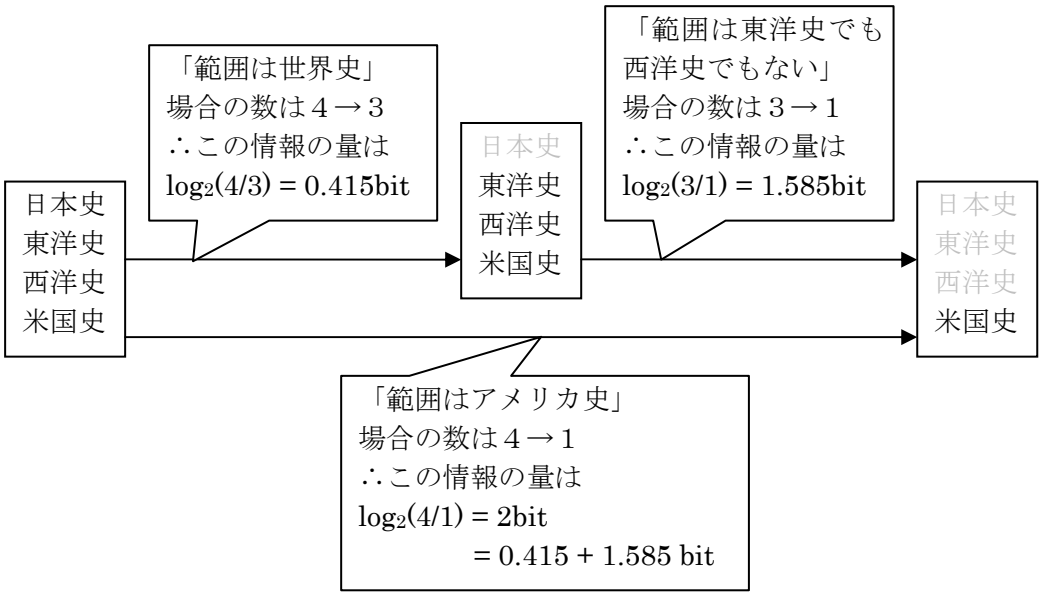
第 3 章　情報の伝達と通信

３．１　情報の伝達と情報量

○ 情報の伝達は受取側の状態の変化が本質（手紙でも手紙のコピーでもメールでも情報は伝わる）
○ 情報量は情報を受け取った効果を測る
○ 情報を受け取る効果は、受け取る人の「状態」と関係がある

情報量

定義は $\log_2 \frac{\text{元の場合の数}}{\text{情報入手後の場合の数}}$ （$=\log_2 \frac{\text{事前の場合の数}}{\text{事後の場合の数}}$ ）であり、単位は bit。情報量は場合の数が大きく減る程数が大きく、底が 2 なので二者択一(場合の数が 2 から 1 になる場合)に 1.0bit となる。また情報量の加法性（→用語集）を満たす。（具体例は下の例＝P41 図 3-2 参照




３．２　情報通信

○ プロトコルは通信の意図を理解するための決め事になる（例：「もしもし@電話」「どうぞ@ﾄﾗﾝｼｰﾊﾞ」
○ プロトコルを正しく使えば機器によらず通信可能
○ 秘密に送りたい情報の盗聴を防ぐために暗号技術が使われる。
○ 偽の要求に騙されないようにするためなどににデジタル署名が使われる。
○ 暗号の種類は鍵の性質によって共通鍵暗号と公開鍵暗号の 2 種類に分類できる。
○ 共通化偽暗号では鍵を秘密に保つ必要がある。
○ 公開鍵暗号では暗号化の鍵と復号の鍵が別で、片方の鍵(公開鍵)を公開し、もう片方の鍵(秘密鍵)を自分だけしかしらないものにすることで、自分宛に暗号化された内容は自分だけが復号できる。

３．３　通信の秘密と相手の承認

○ 回線交換では通信速度が保証されるので通信が途切れないことが重要なデータに適している一方で通信するデータが無い場合でも回線を占有するので、回線の利用効率が悪くなることもある。
○ パケット交換では 1 パケットの送信時間以上は回線を占有しないので回線上に様々な通信パケットが混在でき、回線の利用効率は良いが、パケット管理の為多少遅延があり、また混雑しているときは速度が落ちるので通信速度の保証は難しい。

３．４　インターネット

○ 現在のインターネットでは TCP/IP というプロトコル群が使われている。
○ インターネットの通信では、ネットワークの集合体をグループごとに管理する
○ ネットワーク同士を接続する中継機器をルータという。
○ TCP/IP は階層的なモデルに基づいたプロトコルで、アプリケーション層、トランスポート層、インタ

ーネット層、ネットワークインタフェース層の 4 層からなっている。仕事内容でそれぞれの層が役割分担をしているイメージ。(それぞれの階層での主なプロトコルは教科書 P56 図 3.14 参照
○ カプセル化により階層毎に制御用のデータを付加する。

通信の方法

1．ブラウザが URL(http://www.example.com/)から宛先の IP アドレス(192.168.1.3)を調べる
2．ブラウザが宛先の IP アドレスのウェブサーバに対して HTTP のメッセージを伝える
   a) 送信元マシンがメッセージをパケットに分割し、パケットごとに宛先の IP アドレスに送信
   b) 各パケットは宛先に送るのに適切なルータを通してバケツリレー的に宛先に届けられる。
   c) 受信したマシンはパケットをもともとの順番どおりに並べ、もとの HTTP メッセージを取り出し、ウェブサーバに渡す。
3．ウェブサーバがブラウザにメッセージを返す。返す方法は２．と同じ。

| TCP/IP モデル | 主なプロトコル | 主な役割 |
|---|---|---|
| アプリケーション層 | HTTP,SMTP,DNS,DHCP | アプリケーション間での通信 |
| トランスポート層 | TCP.UDP | 信頼性のある 1:1 の通信 |
| インターネット層 | IP | ネットワーク間の通信 |
| ネットワークインタフェース層 | | ネットワーク内の通信 |

第 4 章　データとモデル

４．１　データモデル
○ 大量のデータを扱うデータベースではデータを体系的に整理し、データ解釈のブレを無くすことが重要である。そのためデータモデルはデータベースでデータを扱うためのモデルとして発展した。

４．３　代表的なデータモデルと演算
○ 代表的なデータモデルに集合モデル、ネットワークモデル、階層モデル、関係モデル、(論理モデル、オブジェクト指向モデル）がある。括弧内は範囲外。
○ 集合モデルを視覚的にわかり易くするものとしてベン図やオイラー図がある。
○ ネットワークモデルは「つながり方」を表すモデルで教科書 P81 の図がそれにあたる。
○ 全ての線を重複なくたどる経路をオイラー経路という。別名、一筆書き。
○ 階層モデル（木構造）は生物の分類図のような枝分かれの構造（教科書 P85 の図のようなもの

集合モデル

| 集合名 | 説明 | 表記 | |
|---|---|---|---|
| 積集合(共通部分) | 集合 $A$、集合 $B$に共通して含まれる要素からなる集合 | $A \cap B$ | $A$ AND $B$ |
| 和集合 | 集合 $A$ の要素と集合 $B$の要素とをあわせもった集合 | $A \cup B$ | $A$ OR $B$ |
| 差集合 | 集合 $A$から集合 $B$の要素を除いた集合 | $A - B$ | $A$ AND NOT $B$ |

関係モデルの関係の操作

| | |
|---|---|
| 和 | 2 つの関係の組を集める |
| 差 | 他の関係にないものを集める |
| 直積 | 2 つの関係の組を総当たりでつないだ組を集める |
| 選択 | 指定された条件を満たすものだけを集める |
| 射影 | 不要な項目を除いた組を集める |

第 5 章 計算とプログラム

５．１　計算とその記述方法
○ 計算によりある集合 $A$ の主要素を求めるという「計数(Counting)」を例に取るとき、$A$ のデータモデルとしてどのような処理が用意されているかに依存。

○ 計数のやり方として、「取り出し型」「分割型」がある。
○ 計算を記述するために必要な要素として「変数(variable)」がある。
○ 変数の値は「代入」により変化させることができる
○ 記述されている手順は書かれている順番に順序良く処理する必要がある。
○ 逐次処理は計算処理の基本の一つである。
○ endif は対応する if と縦に並ぶ高さに書いてある。then と else は if よりも右にずらして書いてある。
これは「まとまりの構造」をわかり易くするためである。
○ 条件付き処理の方法 1 では 6 月以降に達したときに対処できないので改良の必要がある。
○ 配列(array)を使うことで要素全体をまとめて扱うことができる。(例：八十八夜問題の *daymonth* m

取り出し型の計算方法
集合に対して用意されている処理：空（要素が 1 つもない）かどうかを判定する
要素を 1 つ取り出す（集合の要素数は 1 だけ減る）
計算：<答>をゼロとおく
*A* が空でないあいだは以下の処理を繰り返す
・要素を 1 つ取り出す
・<答>を 1 増やす
結果：要素が 0 になったとき、<答>は要素数を示す
図：教科書 P98

分割型の計算方法
集合に対して用意されている処理：空か，要素が 1 つだけであるかを判定する
空でない 2 つの集合に分割する
計算：A が空なら<答>は 0，要素数が 1 なら<答>は 1
そうでなければ A を B と C に分割（B も C も空集合ではない）
<答> ＝ B の要素数 ＋ C の要素数
結果：最終的に要素数 1 の集合が元あった要素数個できる。
図：教科書 P99

下の問題を計算する方法を考える。
問：今年の八十八夜(立春から数えて 88 日目)は何月何日か。今年の立春は 2 月 4 日。今年は平年。
○考え方
2 月 4 日の 87 日後は「2 月をはみ出す」 → 2 月の残り日数(28 - 4 = 24 日)を引く(88 - 24 = 63 日)
3 月 63 日は 3 月を越す → 31 日を引く(63 - 31 = 32 日)
4 月 32 日は 4 月を越す → 30 日を引く(32 - 30 = 2 日)
5 月 3 日は 5 月に収まる！→ 最終的な答えは 5 月 2 日

○表記の仕方 1
2 月 4 日の 87 日後 <残り日数>= 4 + 87 → 2 月 91 日
91> 28(2 月の日数) <残り日数>= 91 - 2 月の日数 → 3 月 63 日
63> 31(3 月の日数) <残り日数>= 63 - 3 月の日数 → 4 月 32 日
32> 30(4 月の日数) <残り日数>= 32 - 4 月の日数 → 5 月 2 日
2< 31(5 月の日数)なので計算終了

○条件付処理での表記 1
<残り日数>←4+87 ＃ℓ1 <残り日数>は 91 日ある
if <残り日数>＞28（2 月の日数） ＃ℓ2 「ℓ1 での<残り日数>」＞28 ならℓ3then、違うならℓ16else へ
then <残り日数>←<残り日数> - 28 ＃ℓ3 <残り日数>に「ℓ2 での<残り日数> - 28」を代入
if <残り日数>＞31（3 月の日数） ＃ℓ4「ℓ3 での<残り日数>」＞31 ならℓ5、違うならℓ14 へ
then <残り日数>←<残り日数> - 31 ＃ℓ5 <残り日数>に「ℓ4 での<残り日数> - 31」を代入
if <残り日数> ＞30（4 月の日数） ＃ℓ6「ℓ5 での<残り日数>」＞30 ならℓ7、違うならℓ12 へ
then <残り日数>←<残り日数> - 30 ＃ℓ7<残り日数>に「ℓ6 での<残り日数> - 30」を代入

```
            if <残り日数>＞31（5 月の日数）　＃ℓ8「ℓ7 の<残り日数>」＞31 ならℓ9、違うならℓ10 へ
              then （6 月以降の処理）　＃ℓ9 <残り日数>に代入の作業を繰り返すのでここでは省略
              else "5 月"<残り日数>"日"と表示　＃ℓ10　5 月<残り日数>日と表示する(例：5 月 2 日
            endif                                ＃ℓ11　ℓ8 で始まった条件付き処理を終わる
          else "4 月"<残り日数>"日"と表示         ＃ℓ12　4 月<残り日数>日と表示する
        endif                                    ＃ℓ13　ℓ6 で始まった条件付き処理を終わる
      else "3 月"<残り日数>"日"と表示             ＃ℓ14　3 月<残り日数>日と表示する
    endif                                        ＃ℓ15　ℓ4 で始まった条件付き処理を終わる
  else "2 月"<残り日数>"日"と表示                 ＃ℓ16　2 月<残り日数>日と表示する
endif                                            ＃ℓ17　ℓ2 で始まった条件付き処理を終わる
```

解説

代入(assignment)を”変数名” ← ”式” で表すとする。

※ X ← X＋3　のようになっている場合はまず矢印の左の $X_{左}$(新しく求まる X)と右の $X_{右}$(ここまでの過程で求まった X)とは別のものと考え、$X_{左}$に $X_{右}$＋3 を代入する。

条件付処理の操作を以下のように表すことにする（赤字は説明

```
if 「条件」　「条件（例えば<残り日数>が 28 より大きいか？）」があったとする。
             条件にあてはまるなら then へ。あてはまらないなら else へ。
  then "条件が成立した場合に行なう処理"　　“”内の処理をして下へ
  else "条件が成立しない場合に行なう処理"　“”内の処理をして下へ
endif　　条件付き処理終わり
```

○条件付処理での表記 1

```
<残り日数> ← 4+87　＃ℓ1　<残り日数>は 91 日ある
m ← 2　　＃ℓ2　m＝2 つまり、2 月から考えはじめる
while <残り日数> ＞ daymonth m do　＃ℓ3　<残り日数> ＞daymonth m なら下の処理を行い、違えば処理終了
  <残り日数> ← <残り日数 > - daymonth m　＃ℓ4　<残り日数>に「ℓ3 の<残り日数> - daymonth m」を代入
  m ← m + 1　　＃ℓ5　m に m＋1 を代入し、次の月の事を考える
done　　＃ℓ6　一周したのでℓ3 に戻る
```

解説

$m$ 月の日数を $daymonth_m$ と書く。例えば $daymonth_2 = 28$。

$m = 1$ のとき $daymonth_1 = 31$、2 のとき 28、3 のとき 30…とあらかじめ入力しておく。

反復処理(repetitive processing)

```
while 「条件」 do　「条件」を充たしている間は、
  "処理"　　"処理"の内容を行う。
done　　「条件」を充たしたら終了
```

## 5．3　プログラムとプログラム言語

- ○ プログラム(program)は計算を記述したもので、コンピュータを使った問題解決における活動単位である。一般に、人間には読み書き不可能である。
- ○ プログラム言語(programming language)は記述するための約束事をまとめたもので、人工的につくられた言語である。人間にも読み書き可能である。
- ○ 機械語(machine language)で書かれたプログラムはすべて 2 進数か決まった長さのビットパターンでバイナリ(binary) プログラムと呼ぶ。
- ○ 各種のプログラムをソフトウェアと呼ぶ
- ○ アセンブリ言語は「データとしての」プログラムを作る
- ○ アセンブラ(assembler)は「データとしての」バイナリプログラムを作る
- ○ 高水準言語ではコンパイラ(compiler)を使うことで機械後に変換する
- ○ コンパイラの作業には手間がかかり、修正が何回も必要な場合などには不都合なので、コンパイラを通さず実行可能なインタプリタ(interpreter)がつくられた。
- ○ 教科書 P118 の図を眺めとくと良いかもしれません。

第６章　問題の解決

６．１　アルゴリズム

○ 現実問題からプログラムを作る際の計算手順の(1)計算のモデル化し、(2)モデル化された問題を解く計算手順を考え、(3)手順どおりに計算するプログラムを作る、という段階の計算手順がアルゴリズム。
○ アルゴリズムの重要性として、「性能に大きな違いが出る」「類型化されている」がある、
○ 同じ問題を解く複数のアルゴリズムがあり、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。
○ 全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り、性能に関する考察・プログラミングを共通化できる
○ アルゴリズムの速度は繰り返しの回数で比べられ、プログラムの実行時間の非常におおざっぱな近似となる。実際のコンピュータの性能、異なる種類の計算を速度差も無視してしまう。

問題解決の手順

現実世界の問題
↓　人間がモデル化
モデル化された問題
↓　人間が考察
アルゴリズム
↓　人間がプログラミング
プログラム
↓　コンピュータが実行
答

アルゴリズムの実例

問題　　：平方根の計算　$\sqrt{x}$ を求める。)
アルゴリズム：反復法、二分法
注意点　　：小数の計算は有限の精度で行われるので近似値しか求められない
やり方　　：ある正の実数 $x$ が与えられたときに、2 乗すると $x$ に近くなる正の実数 $y$ を精度 $\delta$ で求める。つまり、$|\sqrt{x} - y| < \delta$ となるような $y$ を 1 つ求める

アルゴリズム１　反復法

$x = 90$, $\delta = 1$ の場合、「$y = 0, 1, 2, 3, \ldots$ を順に検討してゆき $(y+\delta)^2$ が 90 より大きくなったら、その 1 つ前が解」

```
y ← 0                    #ℓ1  y に 0 を代入する
while (y+δ)² < x do      #ℓ2  (y+δ)² < x なら下の処理を行い、違うなら処理を終える
   y ← y+δ               #ℓ3  y に y+δ を代入する。上の例なら 2 週目には y に 1 が入る
done                     #ℓ4  一周したのでℓ2 に戻る
return y                 #ℓ5  y を解として計算を終了
```

※return $y$ は $y$ を解として計算を終了するという意味

反復法の速度：繰り返しの回数は約 $\frac{\sqrt{x}}{\delta}$ 回。精度を 1 桁増やすと回数も 10 倍に増える

もっと速度を早くしたい
アイデア：1 桁ずつ順に求めていく。
特徴：解がある範囲を $\frac{1}{10}$ ずつ狭めてゆく

$\sqrt{2}$ を求める場合

0,1,2,3,... と検討 →1≦y<2
1.0, 1.1, 1.2, ... と検討 →1.4≦y<1.5
1.40, 1.41, 1.42, ... と検討→1.41≦y<1.42
1.410, 1.411, 1.412, ... と検討→1.414≦y<1.415　・・・と続く

アルゴリズム２　二分法

名前のとおり「区間の幅」が $\frac{1}{2}$ ずつ減ってゆく

区間$(a,b)$を解の存在する範囲、$c$ を区間の中央とする

```
a ← 0                    #ℓ1  a に 0 を代入する
b ← x                    #ℓ2  b に x を代入する
while b - a > δ do       #ℓ3  区間がδよりも大きければ精度δに達していないので処理実行
  c ← (a+b)/2            #ℓ3  c に a, b の平均値＝区間(a,b)の中央の値を代入
  if c² > x              #ℓ4  区間(a,b)の中央の値の二乗がもし x よりも…
    then b ← c           #ℓ5    大きかったら新しい b にℓ4 での c を代入する
    else a ← c           #ℓ6    小さかったら新しい a にℓ4 での c を代入する
  endif                  #ℓ7  条件付き処理終了
done                     #ℓ8  一周したのでℓ3 に戻る
return a                 #ℓ9  a を解として計算を終了
```

二分法の速度：$n$ 回繰り返し後の区間の幅 $\frac{x}{2^n}$ がδ以下になるのに要する回数なので

およそ $\log_n \frac{x}{\delta}$ 回となる。

反復方は約$\frac{\sqrt{x}}{\delta}$回、二分法はおよそ $\log_2 \frac{x}{\delta}$ 回。$x=2$、$\delta=0.0000000001$ のとき（$\sqrt{2}$ を少数第 10 位まで求める）、反復法では約 141 億回、二分法では 35 回となる。

６．１．４　計算量

○ アルゴリズムによって計算時間が大きく違い、また、解決すべき問題に求めるのにかかって良い時間が異なるので、計算時間の見積もりは重要となる。（例：天気予報の計算に 3 日もかけられない
○ 計算量はコンピュータ性能の違いやプログラムの作り方を無視した「問題の大きさ」に対する関係のおおまかな見積り
○ アルゴリズム同士を比較することでプログラムを作る前に良いアルゴリズムを選べる。
○ 計算量の見積もり方：問題の大きさを変数で表わし、計算の回数を式で表わす。詳細な式ではなく「オーダー」を使う
○ 計算量の見積もり方のポイントとして、定数倍の差はコンピュータの性能の違いやプログラムの作り方ですぐに変わるので、定数を無視し、各変数について一番変化の大きい項だけを残すことがある。
○ 計算時間の差は $n$ と $2n$ よりも $n$ と $n^2$ の方が大きいので、ひとつのアルゴリズムを工夫して数倍は訳するよりも、別のアルゴリズムを使ったほうが計算が速くなることが多い。

計算量の例：平方根の計算
問題：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：反復法アルゴリズム　$O(\frac{\sqrt{x}}{\delta})$、二分法アルゴリズム　$O(\log_2 \frac{x}{\delta})$

計算量の例：フィボナッチ数の計算
問題：$n$ 番目のフィボナッチ数を求める
問題の大きさ：$n$
計算量：再帰アルゴリズム　$O(2^n)$、メモ化アルゴリズム　$O(n)$
※再帰アルゴリズム、メモ化アルゴリズムは範囲外

## 第 7 章　計算の機構

### 7．1　計算の実現機構

- ○ フォンノイマン(von Neumann)型コンピュータは、メモリ上にデータとプログラムを保持し、機械語プログラムを解釈し実行する。
- ○ コンピュータの基本構成は「中央処理装置(CPU)」と「主記憶装置(メインメモリ)」
- ○ CPU は主記憶装置と演算送致の制御を行う「制御装置」とデータに対する演算の処理を行う「演算装置」とからなっている。
- ○ メインメモリは情報（データ）の格納と選択的な読み書きを行う。また、アドレスによってデータの位置を特定する。
- ○ 機械語レベルのプログラムは命令集合の並びとして記述される。
- ○ 各命令は「命令コード(operation code)」と「オペランド(operand)」とで構成される。
- ○ 実際の機械語プログラムはビットパターン（0,1 の列）であり、電圧の高低や電流の ON/OFF に対応。
- ○ アセンブリ言語によって、下記のような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
- ○ コンパイラは高水準言語をバイナリプログラムに変換する。

命令の例：基本的に英単語の意味を考えれば予想がつくと思います。

| | | |
|---|---|---|
| データ転送命令 | load A | レジスタにアドレス A のデータを読み込む |
| | store A | レジスタのデータをアドレス A に書き込む |
| 演算命令 | add A | アドレス A の値をレジスタの値に加える |
| | subtract A | アドレス A の値をレジスタから減ずる |
| 分岐命令 | jump A | アドレス A にプログラムの実行を移す |
| | jumpzero A | レジスタの値が 0 のときアドレス A にプログラムの実行を移す |
| その他 | write | 演算レジスタのデータを出力する |
| | halt | プログラム停止 |

図 7.2 解説：アドレス 2001,2002,2003 のデータは初期状態で 0,10,1 となっている。
値は 1008 までは一週目のもの。1009 からは jumpzero したあとのもの

| | | レジスタ | 2001 | 2002 |
|---|---|---|---|---|
| 1001 load 2001 | レジスタにアドレス 2001 のデータを読み込む | ? → 0 | 0 | 10 |
| 1002 add 2002 | レジスタの値にアドレス 2002 の値を加える | 0 → 10 | 0 | 10 |
| 1003 store 2001 | レジスタの値をアドレス 2001 に書き込む | 10 | 0 → 10 | 10 |
| 1004 load 2002 | レジスタにアドレス 2002 のデータを読み込む | 10 → 10 | 10 | 10 |
| 1005 subtract 2003 | レジスタの値からアドレス 2003 の値を引く | 10 → 9 | 10 | 10 |
| 1006 store 2002 | レジスタの値をアドレス 2002 に書き込む | 9 | 10 | 10 → 9 |
| 1007 jumpzero 1009 | レジスタの値が 0 なら 1009 へ、違うなら 1008 | 9 | 10 | 9 |
| 1008 jump 1001 | 1001 に戻る | 9 | 10 | 9 |
| 1009 load 2001 | レジスタにアドレス 2001 の値を読み込む | 0 → 55 | 55 | 0 |
| 1010 write | 結果を出力する | 55 | 55 | 0 |
| 1011 halt | プログラム停止 | 55 | 55 | 0 |
| 2001 0 | 足し算の結果。最初は 0 | | | |
| 2002 10 | 次に足すもの。10 から | | | |
| 2003 1 | いくつおきに数を足すか。1 個ずつ足す。 | | | |

アドレス 2002 に次に何を足すか、2001 に足し算の答えを記録しています。
結果的にやっていることは 10+9+8+7+6+5+4+3+2+1 になっています。

## 7．2　論理演算と組合せ回路

○ 2 進符号表現の演算を表現する回路は「組合せ回路」と「順序回路」とに大別される。
○ 0, 1 からなる論理演算（真理値表と論理関数）を利用する。
○ 論理演算を実現する組み合わせ回路の構築
○ 論理関数は{AND, OR, NOT}の組み合わせによりあらゆる真理値表が実現可能。

1 ビットの加算の真理値表の見方

2 進数で $x + y = c_{out}\ s$ となっています。$s$ は sum(和)、$c_{out}$ は carry out（桁上げ）を表しています。$c_{out}$ と $s$ の左右は逆に考えたほうが数値を読みやすいかと思います。

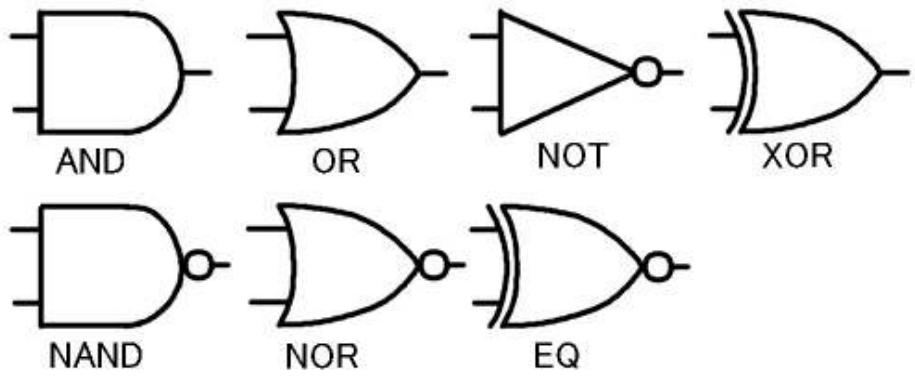
| $x$ | $y$ | $s$ | $c_{out}$ | 意味 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 + 0 = 00 |
| 0 | 1 | 1 | 0 | 0 + 1 = 01 |
| 1 | 0 | 1 | 0 | 1 + 0 = 01 |
| 1 | 1 | 0 | 1 | 1 + 1 = 10 |

2 変数演算（入力が $x$ ,$y$ のみの演算）の真理値表、論理関数表現、(ブール代数表現（範囲外))

↓上二行が入力値、下 16 行が出力結果

| | | | | 論理関数表現 | (ブール代数表現) | 意味 |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | $x_1$ | $x_1$ | 変数の $x_1$ 値 |
| 0 | 1 | 0 | 1 | $x_2$ | $x_2$ | 変数の $x_2$ 値 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 を出力 |
| 0 | 0 | 0 | 1 | AND($x_1$ , $x_2$) | $x_1 \cdot x_2$ | $x_1$ , $x_2$ 共に 1 なら 1 を出力 |
| 0 | 0 | 1 | 0 | | $x_1 \cdot \overline{x}_2$ | $x_2$ の入力を逆にしたものと $x_1$ 共に 1 なら 1 を出力 |
| 0 | 0 | 1 | 1 | $x_1$ | $x_1$ | $x_1$ に入力されたものを出力 |
| 0 | 1 | 0 | 0 | | $\overline{x}_1 \cdot x_2$ | $x_1$ の入力を逆にしたものと $x_2$ 共に 1 なら 1 を出力 |
| 0 | 1 | 0 | 1 | $x_2$ | $x_2$ | $x_2$ に入力されたものを出力 |
| 0 | 1 | 1 | 0 | XOR($x_1$ , $x_2$) | $x_1 \oplus x_2$ | $x_1$ , $x_2$ どちらか一方のみが 1 なら 1 を出力 |
| 0 | 1 | 1 | 1 | OR($x_1$ , $x_2$) | $x_1 + x_2$ | $x_1$ , $x_2$ 少なくとも一方が 1 なら 1 を出力 |
| 1 | 0 | 0 | 0 | NOR($x_1$ , $x_2$) | $\overline{x1+x2}$ | $x_1$ , $x_2$ 少なくとも一方が 1 なら 0 を出力(OR の逆) |
| 1 | 0 | 0 | 1 | EQ($x_1$ , $x_2$) | $x_1 \odot x_2$ | $x_1$ , $x_2$ どちらか一方のみが 1 なら 0 を出力(XOR の逆) |
| 1 | 0 | 1 | 0 | NOT($x_2$) | $\overline{x}_2$ | $x_2$ に入力されたものと逆の値を出力 |
| 1 | 0 | 1 | 1 | | $x_1 + \overline{x}_2$ | $x_2$ の入力を逆にしたもの, $x_1$ どちらか 1 なら 1 を出力 |
| 1 | 1 | 0 | 0 | NOT($x_1$) | $\overline{x}_1$ | $x_1$ に入力されたものと逆の値を出力 |
| 1 | 1 | 0 | 1 | | $\overline{x}_1 + x_2$ | $x_1$ の入力を逆にしたもの, $x_2$ どちらか 1 なら 1 を出力 |
| 1 | 1 | 1 | 0 | NAND($x_1$ , $x_2$) | $\overline{x1 \cdot x2}$ | $x_1$ , $x_2$ 共に 1 なら 0 を出力(AND の逆) |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 を出力 |

MIL 記法




大抵、左上の線から入ってくる値が $x_1$、左下の線から入ってくる値が $x_2$ で、右から出力する○には入ってきた(もしくは出て行く)信号の 0,1 を逆にする効果があると考えると良いかも。
上段四つ(＝真理値表の黄マーカー)は重要だと思います。
NAND は NOT AND、NOR は NOT OR と理解すると良いかも。

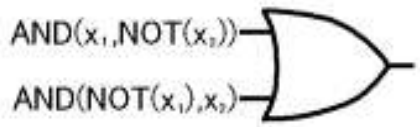
($x_1$・ $\overline{x}_2$)+ ( $\overline{x}_1$・ $x_2$) = OR(AND($x_1$,NOT($x_2$)),AND(NOT($x_1$),$x_2$))を MIL 記法で表すやり方

1．まず、AND($x_1$,NOT($x_2$) )と AND(NOT($x_1$),$x_2$) )はそれぞれ一つの塊と見て、OR を書く

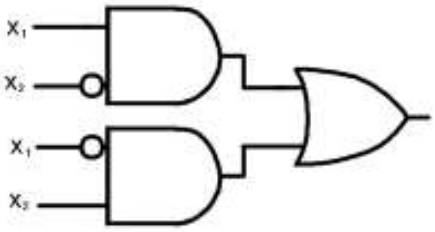
2．次に、AND を図に直す。NOT は入力部分に○を書いて表す

3．$x_1$ と $x_2$ の入力を一つにまとめる（交叉点は●で書く）

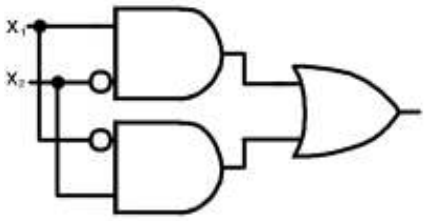
1．




2．




3．




7．3　演算回路

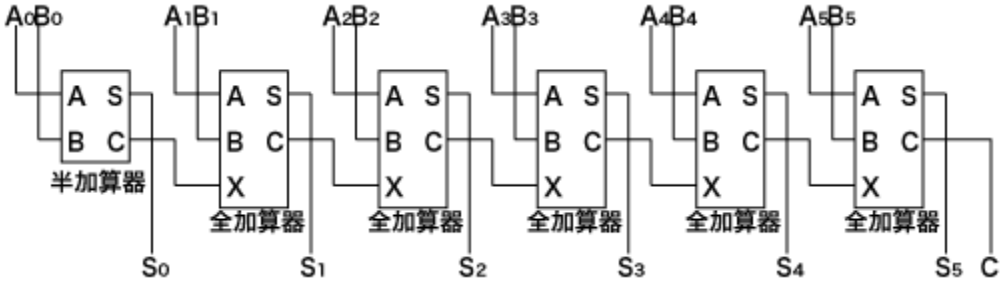
○ 2つの1ビット入力 $x$,$y$に対する加算は上の真理値表と1bit加算の真理値表より $s = x \oplus y$ = XOR($x$, $y$)、$c_{out} = x \cdot y$ = AND($x$, $y$)であることがわかり、これを MIL 記法で書くと教科書 P164 図 7.7 になる。これを半加算機といい、真理値表は上の 1bit 加算の真理値表(= 教科書 P158 表 7.2)

○ 桁上げ入力 $c_{in}$ も考慮した真理値表(教科書 P158 表 7.3)に対応する加算機を全加算器といい、MIL 記法で書いたものは教科書 P164 図 7.8。$x + y + c_{in} = c_{out}s$ となる。最大は 1+1+1=11(=10 進数の 3)

○ 各段階における中間的な構成要素（教科書 P164 図 7.8 の半加算機など）をモジュールという。

複数ビットの加算機(Wikipedia より)

前述の半加算器 1 個と、この全加算器を何個か組み合わせる事によって、任意の桁数の 2 進数加算器が構成できる。下図は 6 桁の加算器の回路図である。($A_5A_4A_3A_2A_1A_0+B_5B_4B_3B_2B_1B_0 \rightarrow CS_5S_4S_3S_2S_1S_0$)




7．5　プログラムの内蔵方式

○ 順序回路では、計算内容の変更には順序回路の構成変更が欠かせないが、プログラム内臓方式では計算の手順をプログラムとしてメモリ上に置き、メモリ上のプログラムを読み出しながら処理を進めることで、様々な計算を実行できる。

第８章　情報システムの役割

8．1　見えにくい情報システム

○ 情報システムとは、広義には情報処理機器（コンピュータなど）と情報伝達ネットワークとをあわせて様々なサービスや機能を提供するシステムのことである。

○ 情報システムとは、狭義にはいわゆるビジネス・アプリケーション企業，政府機関，サービス機関などが、外部に提供する情報サービスシステムや内部で用いる情報管理，意思決定支援システムのこと。

○ 情報システムをより広義にとると、計測・制御系、組込みシステム、インターネット、ゲームも含む。

○ ソフトウェアは抽象的記述で、物理的実体がない。

○ 組み込まれたコンピュータは見えず、また、組み込んであるものは携帯電話から炊飯器まであまりに日常的で情報システムであることが意識されない

○ PC は端末で、ネットワークの向こうで見えないコンピュータが多数稼動している。

８．２　情報システムの仕組み

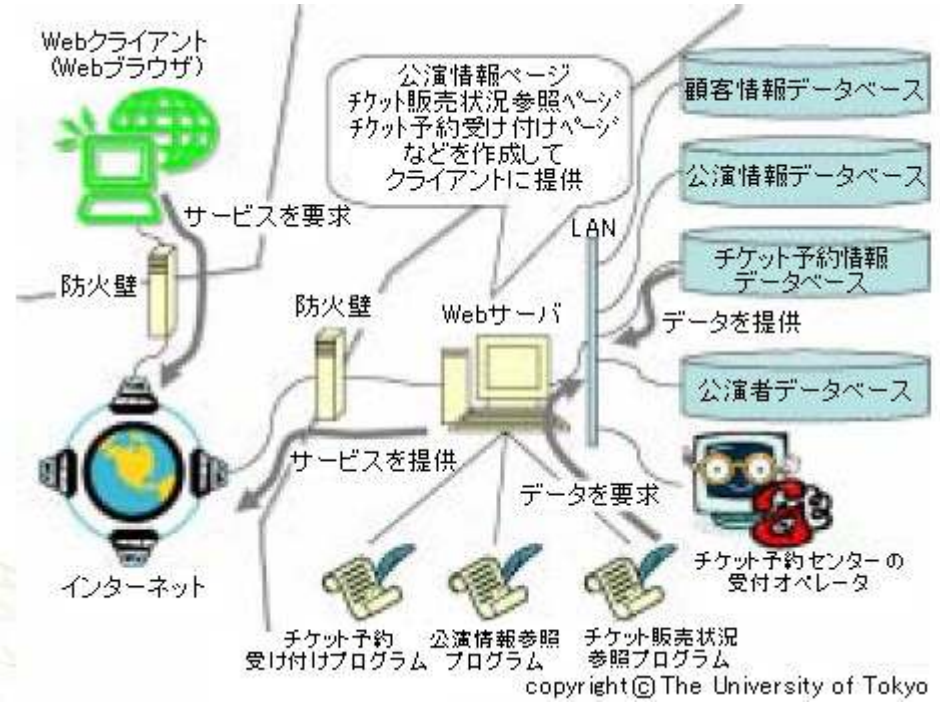
チケット予約システムの購入者の立場から見たシステムを例に考える…らしい

- ○ ウェブサーバは利用者からの要求に基づいて、公演情報などを表示するページを作成し、ウェブブラウザに送り返す。この際、チケット予約に関する具体的な処理を行うプログラムがウェブサーバから呼び出されてその機能を果たす。
- ○ チケット予約に関する具体的な処理を行うようなプログラムを適用(application)プログラム(アプリケーション)という。
- ○ アプリケーションがデータベースサーバにアクセスして情報を得る段階では、アプリケーションがクライアントとなり、データベースを管理する側がサーバになる。
- ○ ウェブサーバの置かれたサイトは通常防火壁(ファイアウォール)を備えてデータベースサーバ内のデータが誤って破壊されたり、無意味なデータが登録されるのを防ぐ。
- ○ 通信規約 HTTP（HyperText Transfer Protocol）はクライアントとサーバ間のやりとりに関する約束事(protocol)の一種。

チケット予約システムで提供される機能は以下を参照。多分重要じゃないと思う。

情報照会
　ジャンル別地域別，公演照会
　チケット販売スケジュール照会
公演詳細情報提供
　公演内容（日時，演目）
　公演者（名前，プロファイル）
　公演会場（会場名称，住所）
　チケット情報（座席別料金，座席表）
　予約関連情報（予約受付先，予約受付期間，予約状況）
　主催，協賛（名前，住所）
予約機能
　公演の指定
　チケット受け取り方法指定
　予約確定
　決済
付随機能
　誤入力の処理 キャンセルの処理

チケット予約システムの仕組みの図




通信要求のコマンド例

GET：クライアントがサーバから情報の資源(ページ)を取得するために出す要求

POST：クライアントがサーバに情報を与えるために出す要求

第 9 章　ユーザインタフェース

９．１　世の中、かくも使いにくい物ばかり？

○ ドアを押すのか引くのかわからない、電灯とスイッチの位置が無関係、携帯電話の機種によって使い方が違う、ワープロが勝手に入力を変更するというのは使いにくい。
○ インタフェース設計者の工夫が予期せぬ形でユーザに解釈されることがある。
○ ユーザによる誤解はインタフェースが設計者や人工物とユーザとを結ぶ唯一の接点となっているために生じる。

９．２　インタフェースの定義とモデル

○ インタフェースとは 2 つの異なる存在の境界面のことである。
○ この章で扱うインタフェースはコンピュータなど人工物とユーザ（人間）との間のインタフェースで、「ユーザインタフェース」、あるいは「ヒューマンインタフェース」と呼ばれる。
○ ユーザはインタフェースを通して人工物を操作するため、人工物が機能を最大限に発揮するためには使いやすいインタフェースが必要
○ コンピュータのインタフェースにおいて、道具への働きかけが対象への働きかけと等価であるとは限らない。例えば、エンター(リターン)キーを押す目的は力の伝達ではなく、情報(意思決定)の伝達。
○ インタフェースは二重接面性を持つ。第一接面（操作インタフェース）はユーザ（心理的世界）と人工物（道具・機械の世界）の間の直接的なもの。第二接面（制御インタフェース）は人工物と物理的なタスク（仕事世界）の間の間接的なもの(教科書 P214 図 9.2 参照)。
○ ユーザの目的は物理的なタスクの実行であるが，操作可能なのは第一接面。
○ コンピュータなど，高度な人工物には二重接面性が存在する。

９．４　まとめ

○ インタフェースはシステム設計者とユーザとがコミュニケーションをもてる唯一の場である。
○ システムのユーザビリティの大部分はインタフェースのユーザビリティによって左右される。
○ インタフェースの構成には，ユーザの感覚運動および認知の行動特性を考慮することが重要。

第 10 章　情報技術と社会

10．１　技術と社会

○ 技術の流通は社会の構成員一人一人のリスクや安全、セキュリティとプライバシー問題などと直結。
○ 技術と社会との関係を論じる際の 4 つの論点として、「技術の中立性」「技術と民主主義」「技術倫理」「リテラシー論」がある。

10．２　技術情報による技術上の変化とその影響

○ インターネット技術を含めた情報技術の影響は幅広く、それまでの社会の体制や権力を支えていた構造をひっくり返す力や社会を変革させる力を持つ
○ インターネット技術により、ホストコンピュータ管理による閉じられた世界から、村（ネットワーク）中心の開かれた世界へ
○ インターネット技術はコンピュータが独立にネットワークにつながることができる技術で、相対的に中央集権的なやりかたを弱める。
○ 技術変化の結果として、「場所の制約からの解放」「時間の制約からの解放」「経路の制約からの解放」「輸送コストのほぼゼロ化」がもたらされた。

コンピュータ技術の変遷

| | |
|---|---|
| 1960 年代<br>1970 年代 | 巨大データベースの管理と大規模計算 |
| 1980 年代 | マイクロコンピューターの誕生<br>消費者市場の中にコンピュータ技術が参入<br>ソフトウェアの所有権にかかわる問題が出現<br>1980 年代まで，人工知能への関心 |
| 1990 年代以降 | インターネット技術の普及<br>プライバシーの問題が再燃 |

10．3　技術情報に固有な社会との軋轢

○ 無形性と複製可能性（ジョンソン，2002）が所有と権利に関わる法制度や倫理との間の軋轢を生む。
○ インターネットのグローバルな通信射程と匿名性がプライバシーとセキュリティの軋轢を生む。
○ ホームページにある他人の文章を引用したり、ネット上で配信されている楽曲を保存したりすることに対して、利用者の心理的障壁が低くなり、著作権の侵害に対する利用者の意識が軽薄になっている。
○ デジタルコンテンツの著作権は使用許諾の概念が曖昧であり、メモリ上の電子情報は市場や流通機構なしに直接個人の手にわたることが可能。
○ 匿名性の高いファイル交換ソフトはデジタルコンテンツの著作権の侵害を幇助する側面を持っている。

| | |
|---|---|
| 著作物として保護されるもの | ソースプログラム<br>オブジェクトプログラム<br>オペレーティングシステム<br>アプリケーション |
| 著作物として保護されないもの | プログラム言語<br>規約<br>解法 |

コンピュータープログラムの著作権に関する事例

Ａ社はビデオゲームＸを開発し、その著作権を所有している。
Ｂ社は都内で経営する喫茶店にゲームＸの無断複製ビデオゲームを設置して顧客に利用させた。
Ａ社は著作権侵害をＢ社に対し訴えることができるだろうか？

○答え

この場合、ビデオゲーム機に取り付けられた ROM に収納されているオブジェクトプログラムはＡの著作物(ソースプログラム)の複製物である。したがって、B が使用したビデオゲーム機のように、ROM のオブジェクトプログラムを他の ROM にコピーして製造した偽造ゲーム機は、A のソースプログラムの著作権を侵害する。

Winny 開発者の起訴問題

起訴した側の主張：著作権侵害を幇助する意味で罪
開発者の支援者や技術者の主張：現在の技術で違反が発生してしまうことが問題で技術進歩と共に法律も進歩すべき

○論争における論点の幅が大きい

10．4　情報技術論

○ 情報技術において、ハードウェアが選択淘汰され、基本ソフトウェアが選択淘汰され、情報技術やブラウザが選択淘汰され、現在の形になってきていると考えられる。
○ われわれ一人一人の選択が次世代の技術に影響を与えうるのだから、技術開発と同時に社会への影響を考え、選択を公に開くことが必要となる。
○ 情報技術の日常生活への浸透度が高く、また、日々の技術革新の速度も速いので、広範囲の人への情報における批判的思考の育成が必要。
○ 技術の便利さの進展が、操作の裏側を知らずに行う操作を増やしているので、情報技術の特徴を押さえたうえでの批判的思考が必要。

10．5　これからの世代の情報

○ 情報技術の普及により、よい側面（意思決定への直接参加の機会促進、民主的な公共空間の創出など）とダークサイド(セキュリティとプライバシー、有害情報の規制、既存の社会規範が追いつかないための問題)の双方が同居し、またそれが技術の発展と共に加速されている。
○ われわれの今の選択は、将来世代の情報技術に影響を与えるので、問題が発生するごとにそれらの議論に「参加」し、新しい社会規範の構築に「参加」できるだけの情報リテラシーを身につけてこれからの社会を生き抜いて欲しいらしい。