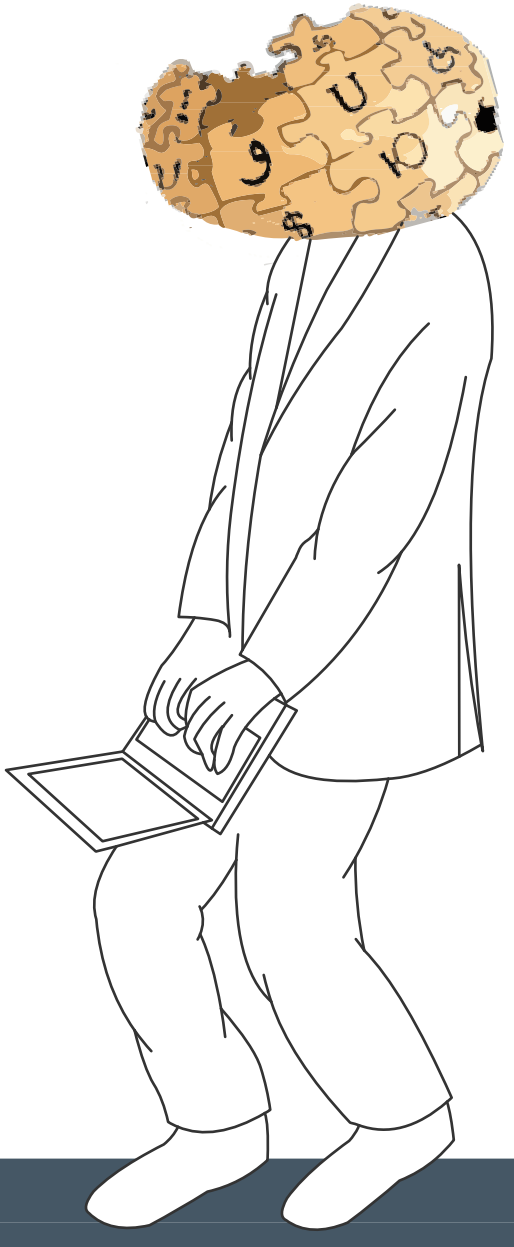
Microcomputer Making Association
the University of Electro-communications

2010 Edition



hyakumangoku

百萬石 春号

MMA
エムエムエー

部長挨拶

kyogoku42
kyogoku42@mma.club.uec.ac.jp
www.kyogoku.biz

こんにちは. MMA です.

百萬石を手にとって頂きましてありがとうございます.
今回は春号ということで, 過去に書かれた新入生向けの記事も, 再掲させていただきました.[*1]

MMA とは, "Microcomputer Making Association" の略称からきています.
歴史あるサークルですので[*2]以前は マイクロコンピューター の設計と製作をも行っていたそうですが,
現在では主に プログラム や ネットワーク の 研究や開発 などを行っています.

MMA は サークル会館 に部室を所有しており, そちらに 共用の コンピューター が 1 台 +4 台が稼働中, ブレードサーバー を 1 台 +8 台 その他多数の コンピューター を所有しており, また他にも 4, 5 台のほど インテリジェントスイッチ のほか, 工作用の機械なども多数保有しておりますので, 各人が好きなことを行うにあたって, 便利な環境をととのえています.
また, ネットワーク 環境的には .club.uec.ac.jp 以下のドメインを管理しているほか, IPv6 の グローバルアドレス を 相当数 使える環境があります[*3] ので, ここも自由がききます.

また, MMA 部室は カードキー となっております. みなさんが今使っている『オサイフケータイ』『SUICA』『PASMO』『Edy』あるいは『学科の計算機室のカードキー』がそのままキーとして使えますので, いつでも勝手に来たいときに部室に来て, 使いたいときに部室を使えるようになっています.

MMA に興味があるかたは, 気兼ねなく サークル会館 2 階 部室 へどうぞ.
いつでもお待ちしております!

[*1] 著者の先輩方へ謹んで感謝いたします.
[*2] 過去の部誌を見る限りで, 1978 年より前からあったそうですね.
[*3] 2001:02F8:0026:1800::/56

目次

情報と計算機科学のノート

kyogoku42

kyogoku42@mma.club.uec.ac.jp

www.kyogoku.biz

概要

計算機科学 の研究と発展は, 時々人類に思いがけない新たな考え方や思想のフィードバック をもたらした.

1822 年, イギリスの数学者 チャールズ・バベッジ によって, 偉大な発明がなされた–計算できる機械, 階差機関 だ. これは機械が考えることなど出来るわけないと考えていた当時の人々にはさぞ衝撃であっただろう.
その後 計算機科学 は, 人類と密接かつ不可欠な関係になりながらどんどん進歩していき, その影響は全世界に絶対に消し去れないレベルで隅々にまで浸透した. 今日の計算機はとうとう, (既に目に見えなくなっている) インフラストラクチャーとしてでない『今は我々の目の前で堂々と (目立ちながら) 活動しているコンピューター』までも, 現実的なレベルにおいて 目に見えなく意識もしないというもの へと向かいだしている.[*1]

**

計算機科学 の研究と発展は, 時々人類に思いがけない新たな考え方や思想の フィードバック をもたらした.

私が興味を持っている科学者に コンラート・ツーゼ という人がいる. 計算機の技術者・科学者である彼は, いくつかの重要な功績[*2]を残した後に, 非常に興味深い概念を提唱しだしている – 「宇宙は計算によって成り立っている」である.
彼の主張は, 「宇宙全体は, 巨大なセル・オートマトンの演算結果である」というものである. これは今日にもなれば聞き覚えのある人も多いであろう話–「シミュレーション仮説 (この世界は計算により実現された世界であるという仮説)」にも近い.
彼の主張には, 物理現象が離散的であるという主張も含まれている. 私はここにおける最小時間単位は プランク時間 (理学系の学生なら何度となく触れてきたであろう 『プランク定数』を考えた マックス・プランク により提唱された.) であろうと考えていたのだが, どうやらさらに小さな時間単位が確認されたようだった [6].
また, この話に関連するのだが, Mathematica や Wolfram|Alpha などで高名な スティーヴン・ウルフラム による, セル・オートマトン と科学の広大な範囲における関係などを指摘した ”A New Kind of Science” という著作もある.

**

これらシミュレーション仮説の話の本質的なところは, 基底的な世界であるか基底的な世界における計算機によって実現された世界かは証明不可能である, というところである.[*3][*4]
この話は哲学的な命題 – つまり「あなたが見ること感じることがあなたにとっての世界である」[*5]という話とも繋がる話である. つまり「自分が見ず感じない世界は存在していない」のであるし, 外部により基底的な低次世界が存在するかどうかということは, この世界においてそれを知るための手がかり[*6]が存在していないのであれば絶対に知ることができないのであり, 絶対に知ることが出来ないことは絶対的に存在していないのであるから, 本質的なパラドクスのようなものを抱えている.
『演算を行うことができる』とともに, 『他の存在とやりとりをすることができる』という要件を満たしていることを見れば, 現代の計算機は既に今の時点において, 『現実』を作り出すための本質的な基礎は持っているともいえる.

**

[*1] かの奇抜 SciFi 小説・映画『銀河ヒッチハイク・ガイド』の著者である ダグラス・アダムス 氏は, すでに 10 年も前に ”今我々は『椅子』を『椅子というテクノロジー』として考えず, ただ『椅子』として考えている” というような議論をしている. つまり, テクノロジーとしてのコンピューターとして意識するのに比較しての話である.
[*2] 世界初のプログラム可能なコンピューターを開発し, また世界初のコンピューター企業を興した.
[*3] ただし, 物理現象などを含むこの世界を構成する要素が全て理論的な計算可能性を持っているものである必要がある.
[*4] 計算可能性理論の話も参考になる.
[*5] 中世のデカルトや遙か紀元前の荘子のいう, 夢と現実との不可区別性の議論とも近い.
[*6] もしくはバグ

反証不可能であることは即ち (手順や姿勢としての) 科学ではない[*7]. こう言ってしまえば シミュレーション仮説 はまさしくそれに当てはまってしまうのではあるが, また我々が現実的な反証手段を持たないということとそれが科学でないか・重要でないかという問題として考えるのであれば, 現実的な検証手段を持っていない事柄はたったこれだけに限らず山高く積まれているので, 一言短く片付けられるものでもない.

参考文献

[1] Simulation argument, http://www.simulation-argument.com/

[2] Calculating Space - Wikipedia, http://en.wikipedia.org/wiki/Calculating_Space

[3] セル・オートマトン - Wikipedia, http://ja.wikipedia.org/wiki/%E3%82%BB%E3%83%AB%E3%83%BB%E3%82%AA%E3%83%BC%E3%83%88%E3%83%9E%E3%83%88%E3%83%B3

[4] Zuse's Thesis: The Universe is a Computer http://www.idsia.ch/~juergen/digitalphysics.html

[5] プランク単位系 - Wikipedia, http://ja.wikipedia.org/wiki/%E3%83%97%E3%83%A9%E3%83%B3%E3%82%AF%E5%8D%98%E4%BD%8D%E7%B3%BB

[6] "Shortest time interval measured" - B.B.C. NEWS, http://news.bbc.co.uk/2/hi/science/nature/3486160.stm

[7] シミュレーション仮説 - WIkipedia, http://ja.wikipedia.org/wiki/%E3%82%B7%E3%83%9F%E3%83%A5%E3%83%AC%E3%83%BC%E3%82%B7%E3%83%A7%E3%83%B3%E4%BB%AE%E8%AA%AC

[8] シミュレーテッドリアリティ - Wikipedia, http://ja.wikipedia.org/wiki/%E3%82%B7%E3%83%9F%E3%83%A5%E3%83%AC%E3%83%BC%E3%83%86%E3%83%83%E3%83%89%E3%83%AA%E3%82%A2%E3%83%AA%E3%83%86%E3%82%A3

[9] 計算可能性理論, http://ja.wikipedia.org/wiki/%E8%A8%88%E7%AE%97%E5%8F%AF%E8%83%BD%E6%80%A7%E7%90%86%E8%AB%96

[10] How to Stop Worrying and Learn to Love the Internet – DOUGLAS ADAMS.COM, http://www.douglasadams.com/dna/19990901-00-a.html

[*7] 似たような話としては, 上述『銀河ヒッチハイク・ガイド』にも, こんなセリフが登場する: 「疑問ではありません. 疑問の意味が分かれば答えの意味も分かるでしょう.」(注釈: 「人生宇宙全ての答えとは？」とは, に対して. 疑問がなんなのかわかっていれば, 答えを聞いたときに, 意味するところを理解できるでしょう？ ということ.)

A note on duplex double-leaved printing

moechar

moechar@mma.club.uec.ac.jp

概要

2009 年秋号では psbook をあてにせず、ページ数に応じて両面袋綴じ印刷に適したページの順序に並べ替える方法を提案しました [2]。本稿では 2009 年秋号の記事とは別の方法で両面袋綴じのページ配置を生成する方法と、逆に両面袋綴じ印刷された冊子をページ昇順に並び替えて保存する方法及び活用例をまことしやかに提案します。多分。

1 両面袋綴じ印刷

両面袋綴じ印刷とは原稿の 2 倍の広さの紙の裏表両面の各面に 2 ページを割り当て、本誌の様に重ねて見開きがページで連番になるように原稿を配置する印刷方法のことです。両面袋綴じ印刷で配置された 8 ページの冊子の見開きの様子を**図** 1-**図** 5 に、2 ページ分配置された紙面を**図** 7-**図** 10 に示します [1]。これらを**図** 6 の様に重ねることで見開きのページ配置が得られます。

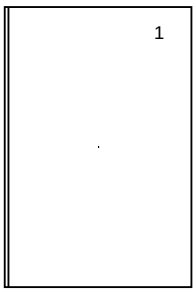


図 1　1 ページ

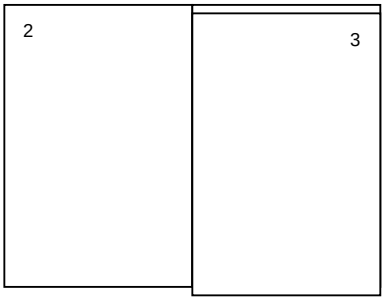


図 2　2,3 ページ

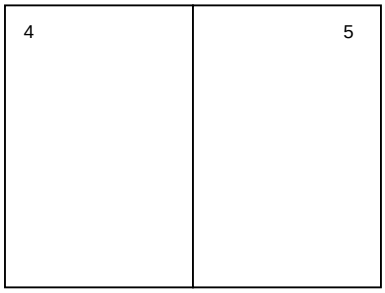


図 3　4,5 ページ

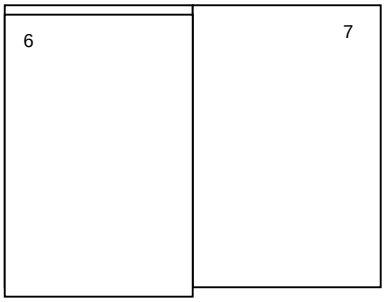


図 4　6,7 ページ

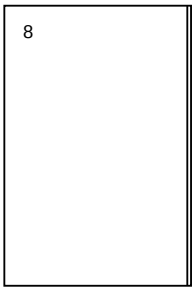


図 5　8 ページ

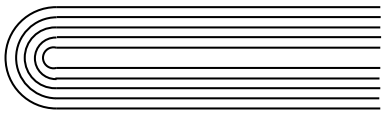


図 6　両面袋綴じ印刷された冊子 (断面図)

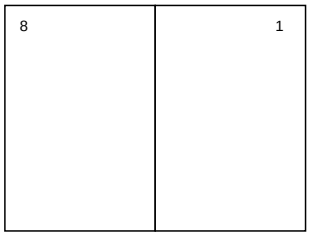


図 7　1 枚目山面 (8,1 ページ)

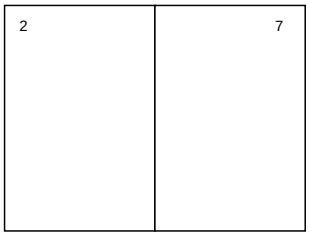


図 8　1 枚目谷面 (2,7 ページ)

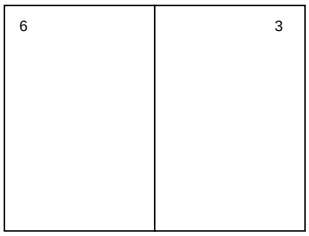


図 9　2 枚目山面 (6,3 ページ)

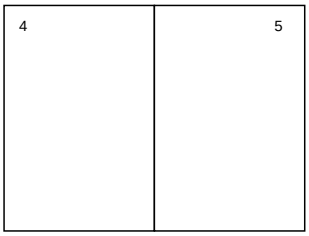


図 10　2 枚目谷面 (4,5 ページ)

印刷用紙ベースで両面袋とじのページ配置の順序を表現します。折り曲げたときの山側の面の左側を A 面、右側を B 面、谷側の面の左側を C 面、右側を D 面とします。(**図** 11, **図** 12)

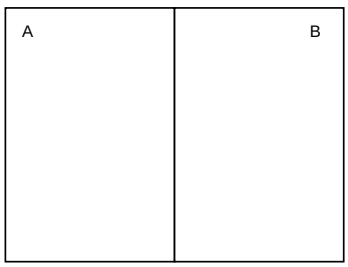


図 11　山側の面

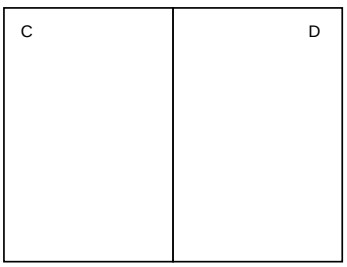


図 12　谷側の面

組版後のページ数を n 枚とします。n が 4 の倍数であるならばページ配置順序は**表** 1 のように表すことが出来ます。[2]

表 1　両面袋綴じ印刷でのページ配置順序

枚数	A	B	C	D
1	n	1	2	$n-1$
2	$n-2$	3	4	$n-3$
⋮	⋮	⋮	⋮	⋮
k	$n-2k+2$	$2k-1$	$2k$	$n-2k+1$
⋮	⋮	⋮	⋮	⋮
$\lceil n/4 \rceil$	$2\times\lceil n/4\rceil+2$	$2\times\lceil n/4\rceil-1$	$2\times\lceil n/4\rceil$	$2\times\lceil n/4\rceil+1$

もし、n が 4 の倍数でなければ n までは**表** 1 のように配置して、$n+1$ から $4\times\lceil n/4\rceil$ までは空白ページとして配置します。$n=20$ としたときの見開きでのページ数と両面袋綴じでのページ配置の関係は**表** 2 の様になります。

表 2　両面袋綴じ配置 ($n=20$)

見開き	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
両面袋綴じ	20	1	2	19	18	3	4	17	16	5	6	15	14	7	8	13	12	9	10	11

表 2 の区切り方によっては、ページ配置順序は i で表現できることが多々あります。とりあえず $f(i)$ とします。**表** 1 からのアナロジーから、$i=1$ から 4 セルごとに区切り、$i \mod 4$ の値で数字列をグルーピングして、その数字列と i の値との関係を調べると、$f(i)$ は多分、次式の様に表すことが出来ます。

$$f(i)=\begin{cases} n-\dfrac{i-1}{2} & (i \mod 4=1)\\ \dfrac{i}{2} & (i \mod 4=2)\\ \dfrac{i+1}{2} & (i \mod 4=3)\\ n-\dfrac{i}{2}+1 & (i \mod 4=0) \end{cases} \tag{1}$$

対象ファイルがページ昇順の配置であるという仮定の下で両面袋綴じのページ配置になるように並べ替えるツールとしては psutils に含まれる `psbook` や、百萬石 2009 年秋号に私が記事中に ([2]) 掲載したスクリプト `permutation.sh` が挙げられます。

2　両面袋綴じ配置の逆変換

前節では両面袋綴じ印刷的に配置されたページ順序の生成規則を示しましたが、本節ではその逆を考えて行きます。まず、議論の簡単のために総ページ数 n が 4 の倍数であるという前提を置きます。尚、本稿は技術誌へ掲載する記事でもないので読者の皆様のイメージが湧きやすいように、n の値を具体的に 20 として話を進めて行きたいと思います。

2.1　メタ視点でのスキーム

本節では (1) 式の知識がないという前提でページ順序を逆変換する方法を提案します。**表** 1 の各行の配置順序を連結したものとその順序の数字が発生する順番を i としたもの、および配置順序を 1, 2,. . . ,20 の順に走査していく際に対応する i の値を**表** 3 に示します。

表 3　全体の走査順序 ($n=20$)

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
両面袋綴じ	20	1	2	19	18	3	4	17	16	5	6	15	14	7	8	13	12	9	10	11
走査順序	2	3	6	7	10	11	14	15	18	19	20	17	16	13	12	9	8	5	4	1

アルゴリズム的な事情から、配置順序を元に戻すのには i または n を使って表現できることが好ましいので、走査順序を i または n を使って表現していきたいと思います。**表** 3 をみると、1 行目はまだしも、2~3 行目にはなにやらよく分からない数字が並んでいるかと思われます。走査順序をなんとかして符号化出来れば、一意に両面袋綴じ順序をページ昇順に対応させることが出来るので、ここでは 2 行目を考えることはやめて 3 行目だけを考えることにしましょう。

1 行目と 3 行目の $i = 1, 2, \ldots, 10$ の部分をみると $g_1(i)$ を走査順序とすると次式の規則性があることが分かります。

$$g_1(i) = \begin{cases} 2i & (i \text{ が奇数}) \\ 2i - 1 & (i \text{ が偶数}) \end{cases} \tag{2}$$

つまり前半部分の走査順序は i の値によって自動的に与えることが可能なようです。しかし、どうやらこの規則性は $i = 11, 12, \ldots, 20$ の部分には適用できないようです。ここは別の発想で、i を n から引いたものを補助情報としてみることにします (**表** 4)。

表 4　後半部分の走査順序 ($n = 20$)

i	11	12	13	14	15	16	17	18	19	20
走査順序	20	17	16	13	12	9	8	5	4	1
$n - i$	9	8	7	6	5	4	3	2	1	0

2 行目と 3 行目の $i = 11, 12, \ldots, 20$ の部分をみると前半部分と同様に $g_1(i)$ を走査順序として、次式の規則性があることが分かります。

$$g_1(i) = \begin{cases} 2(n-i) + 2 & (i \text{ が奇数}) \\ 2(n-i) + 1 & (i \text{ が偶数}) \end{cases} \tag{3}$$

つまり、要素数が n で添字が 1 から始まる配列の各セルに両面袋綴じ配置のページが**表** 3 の様に格納されていれば i ごとに上記の (2),(3) 式で走査順序を生成してその順序で配列を走査することで $1, 2, \ldots, n$ のページ列を得ることが出来るという訳です。

本節のスキームで復元された昇順ページを $\hat{i}_1$、$f(i)$ の値が i に応じて**表** 2 の様に格納された添字が 1 から始まる配列を $F[i]$ とします。これらと $g_1(i)$ を用いて $\hat{i}_1$ を表すと、

$$\hat{i}_1 = F[g_1(i)]. \tag{4}$$

2.2　逆写像でのスキーム

$f(i)$ は多分、全単射なので逆写像 $f^{-1}(\cdot)$ が存在します。$g_2(i) = f^{-1}(i)$ として逆関数を求めると次式の様になります。

$$g_2(i) = \begin{cases} 2(n - f(i)) + 1 & (i \mod 4 = 1) \\ 2f(i) & (i \mod 4 = 2) \\ 2f(i) - 1 & (i \mod 4 = 3) \\ 2(n - f(i)) + 2 & (i \mod 4 = 0) \end{cases} \tag{5}$$

つい最近見かけた式の形が現れました。偶奇の違いこそあれども $f(i)$ を i で置き換えれば (2),(3) 式と同じ形をしています。本質的に同じ物を求めていたことがわかります。多分。

本節のスキームで復元された昇順ページを $\hat{i}_2$ として $f(i)$ と $g_2(i)$ を用いて $\hat{i}$ 表すと、

$$\hat{i}_2 = g_2 \circ f(i) = g_2(f(i)). \tag{6}$$

3　活用例

本稿を書くに至った動機として過去の百萬石 (部誌) を Web で公開する作業を行っていました。電子媒体での原稿が見当たらないものについては紙媒体から復元せざるをえません。ほとんどの号の紙媒体は両面袋綴じ印刷でページが配置されていました。見開きでスキャンしようにもずれてしまって作業が難航しました。そこで、ワタシは適度に手抜きしたかったので冊子の紙面ごとにスキャンしてなんとか出来ないだろうかと画策し、前節のような規則性を見出しました (キリッ)。

本節ではこの作業をもとに前節の逆両面袋綴じ配置の活用例をケーススタディ的に示します。活用例では ImageMagick に含まれる `identify` と `convert` コマンドを利用しています。もし未導入なら ports の `graphics/ImageMagick` を `portinstall` してください。

実際の作業では富士ゼロックス社の複合機 DocuCentre-III 2000 を使いました。

行った作業とそれに対応するスクリプトのうち、前節で議論したメタ視点でのスキームで実装した逆変換スクリプト `permedjpgconv.sh` を示します。[*1]

(1) 冊子から表紙の紙を除いて、折った状態での外側の紙から　山面 → 谷面 → 次の紙の山面 → ··· → 一番内側の紙の谷面の順に自動両面原稿送り装置でスキャンします。

[*1] そのほかのスクリプトは `http://delegate.uec.ac.jp:8081/club/mma/~moechar/hyakumangoku/10a/` からダウンロード頂けます。多分。

この時点では冊子のサイズの二倍の広さの画像に両面袋綴じ的に配置されたページが 1 枚の画像に 2 ページずつ格納されているので、これをなんとか分解して、並び替えてやる必要があります。分解した後は landscape かもしれないので回転を加えて portrait にする必要もあるかもしれません。

(2) ファイル名にスペースが含まれていたのでスペースを詰めます。(pack-name.sh)
(3) 画像を 2 分割します。(divide.sh)
(4) 画像を右に 270 度回転させます。(rotate.sh)
(5) ページ昇順に並び替えて、再度ナンバリングして (0 埋め処理)、jpg に変換します。(permedjpgconv.sh)

permedjpgconv.sh

```
1   #!/usr/local/bin/bash
2   n=$1
3   ((half=n/2))
4   for ((i=0;i<n;i++)) do
5       ((k=i+1))
6       ((o=k%2))
7       if [ $k -le $half ]; then
8           if [ $o -eq 1 ]; then
9               ((perm[i]=2*k))
10          elif [ $o -eq 0 ]; then
11              ((perm[i]=2*k-1))
12          fi
13      elif [ $k -gt $half ]; then
14          if [ $o -eq 1 ]; then
15              ((perm[i]=2*(n-k)+2))
16          elif [ $o -eq 0 ]; then
17              ((perm[i]=2*(n-k)+1))
18          fi
19      fi
20  done
21  before=‘ls -al | grep .tif | awk -F" " ’{print $9}’ | awk -F"." ’{print $1}’‘
22  vbefore=($before)
23  for ((j=0;j<n;j++)) do
24      ((i=j+1))
25      contain[$i]=‘echo ${vbefore[$j]}‘
26  done
27  for ((i=0;i<n;i++)) do
28      index=‘echo ${perm[$i]}‘
29      ((j=i+1))
30      if [ $j -lt 10 ]; then
31          j=0$j
32      fi
33      convert ${contain[$index]}.tif ./$j.jpg
34  done
```

まず、メタ視点でのスキームを用いて配列 perm に走査順序を格納します (2~20 行目)。そして、対象となる TIF 画像ファイル名ののリストから拡張子を取り除いた物を vbefore という配列に格納します (21~22 行目)。作業の都合上、配列の添字をインクリメントして新たに配列 contain に格納します (23~26 行目)。perm の要素を用いて contain を走査し、JPEG 画像に変換します (27~34 行目)。

usage

```
$ ./permedjpgconv.sh n
```

4 おまけ

4.1 冊子の途中に半分サイズの紙を含む場合

実は、2002 年春号は B4 紙を 2 つに折った B5 サイズで配布されました。しかし全て B4 紙で構成されていませんでした。当時の編集担当の裁量かも知れませんが最も内側の紙は B5 紙で、23 ページと 24 ページが配置されていました。この 2002 年春号をページ昇順に保存するにあたっては上記のスキームはそのまま適用できないので B4 紙パートと B5 紙パートに分けて取り組む必要がありそうです。

B4 紙に関しては前節の (1)~(4) までのステップは同じで、(5) のステップで用いる permedjpgconv.sh を $n/2$ ページと $n/2+1$ ページを走査しないように書き換えます。メイン処理部である 2~20 行目を書き換えたものを右に示します。(usage はもとの permedjpgconv.sh と同じです。)

B5 紙に関しては手動でスキャンして変換して、$n/2$ ページか $n/2+1$ ページのどちらかに対応するようリネームします。

```
n=$1
((half=n/2))
((ihalf=half+1))
for ((i=0;i<n;i++)) do
    ((k=i+1))
    ((o=k%2))
    if [ $k -le $half ]; then
        if [ $k -lt $half ]; then
            if [ $o -eq 1 ]; then
                ((perm[i]=2*k))
            elif [ $o -eq 0 ]; then
                ((perm[i]=2*k-1))
            fi
        fi
    elif [ $k -gt $half ]; then
        if [ $k -gt $ihalf ]; then
            if [ $o -eq 1 ]; then
                ((perm[i]=2*(n-k)+2))
            elif [ $o -eq 0 ]; then
                ((perm[i]=2*(n-k)+1))
            fi
        fi
    fi
done
```

4.2 袋綴じ印刷の場合

袋綴じ印刷は両面袋綴じ印刷とは違い、片面の山面にだけページを配置する手法です。ページ配置が両面袋綴じとは違い、$(1,2),(3,4),\ldots,(n-1,n)$ のようにインクリメンタルにページが配置されます。折った後に角と角を重ね合わせてステイプラーで綴じるのが一般的なやり方です。反面、紙の消費量が両面袋綴じの 2 倍であったり、折った方が盛り上がりやすくなるなどといったデメリットもあります。

袋綴じ印刷の冊子を保存する方法を考えてみましょう。両面袋綴じ印刷での (1)~(4) までのステップは同じで、(5) のステップで用いる `permedjpgconv.sh` を変更することになります。袋綴じ印刷の場合は左面 → 右面 → 次の紙の左面 → $\cdots$ の様にインクリメンタルにページが配置されるので走査順序を生成する部分は不要となります。変更後のスクリプトを以下に示します。(usage はもとの `permedjpgconv.sh` と同じです。)

```
#!/usr/local/bin/bash
n=$1
before=‘ls -al | grep .tif | awk -F" " ’{print $9}’ | awk -F"." ’{print $1}’‘
vbefore=($before)
for ((j=0;j<n;j++))
do
    ((i=j+1))
    contain[$i]=‘echo ${vbefore[$j]}‘
done
((n++))
for ((i=1;i<n;i++)) do
    index=‘echo ${perm[$i]}‘
    ((j=i))
    if [ $j -lt 10 ]; then
        j=0$j
    fi
    convert ${contain[$i]}.tif ./$j.jpg
done
```

5 おわりに

本誌のデジタル原稿 (ページ昇順) は多分、近日中に MMA の Web サイトで公開されると思いますので、本稿の 3 節の様な作業を踏まれる必要はありません。「それまで待てないッ!」という方が居られましたらお試しください。

参考文献

[1] ポストスクリプトプリンタで両面印刷, `http://www.d2.dion.ne.jp/~imady/unixtips/duplexps.html`

[2] moechar, “bash で遊ぼう ~ 自動組版の劣化実装編 ~”, 百萬石 2009 年秋号, 電気通信大学 MMA, pp. 13–15, 2009 年 11 月.

間違いだらけの趣味プログラムへの取り組み方

takkun ⟨takkun@mma.club.uec.ac.jp⟩

2001 年 11 月

概要

趣味でプログラムを書く時に、ちょっとでも、真面目にプログラムを書こう、とか思うと、車輪の再発明をしないように、とか、きちんとしたアルゴリズムを考えてから、とか、いろいろと調べ物をしたり、頭を悩ませるような事を多くする事になります。そういう事というのはとても重要なのですが、ここでは、あえて、そういう面倒くさい事をしないプログラムの書き方について考えてみたいと思います。

1 まず書け

とりあえず、プログラムを書こうと思っているのであれば、何か目標とする最終形態のような物があると思います。たとえば、縦書きエディタが作りたいとか、メジャーでないフォーマットの画像ファイルのビューワを作りたい、とか、そういう目標のような物です。

そういう物は、あくまでも最終目的です。そんなものは最初からできっこないのです。とりあえず、おぼろげながらの考えでいいので、code を書きはじめましょう。書きはじめるというのはそれなりに重要です。書きはじめないと何もはじまらないし、書き出すことによって初めてわかってくる問題というものはそれなりにあります。もちろん、書きはじめる前に、熟考してもいいかもしれません。しかしながら、書きはじめる前に、熟考して気がつく問題というのは、書きはじめてしまってからも気がつくものです。それに、ただ考えるだけで何も書かない人よりも、そのアイディアを形にした人の方が、世間的評価はだいたい高い方向にあります。要するに、口だけのクズは掃いて捨てろ、とにかく書け、と、いうことです。

2 玉砕せよ、しかし走り続けよ

では、書きはじめてからそれが完成するまでそのまま走り続けられるか、と、いうと、普通はそうは問屋が卸してくれません。だいたい、最初に詳細な設計なんて物を考えずに書きはじめるわけですから、話がでかくなるに従ってボロが出はじめます。そう感じたら、今まで書いてきたその code は捨てましょう。そうです。捨てるのです。さっさと捨てて新しく code を書き直しましょう。なお、ここで、熟考してはいけません。ボロが出た部分、すなわちマズいと感じた部分を、マズくない形に一から作りなおすことだけに集中して、さっさと書き直すのです。code を書くのを止めてはいけません。ただただ書きつづけるのです。書けばかくほど、書き直せばかきなおすほど、ボロの真の意味がわかってくると思います。真の意味がわかったその時には、素晴らしいデータ構造とアルゴリズムがあなたのプログラムに実現するに違いありません。それに、新しく書き直す場合には、別に今までのプログラムの形やインタフェースにこだわる必要はなくなります。きっと、あなたのプログラムはよりシンプルな構造をもつようになるでしょう。

3 使える物を使え

さて、あなたの作りたいプログラムの最終形態は、いったいどんな動作をするのでしょう。コマンドラインで便利なフィルタのような物でしょうか、それとも、ウィンドウがあって、ボタンがあって、というような、便利な GUI のプログラムでしょうか。

当然、その目的に応じたライブラリやプログラム言語というものが、多分、存在しているでしょう。でも、どんなものがあるのか調べて使うという事は、あえてせずに、自分の知っている言語、自分の知っているライブラリで code を書きはじめてしまいましょう。たぶん、その便利なライブラリやプログラム言語というものは、知っているだけで使った事が無いものだったりするものが多いと思います。自分のよく知っているライブラリやプログラム言語というものは、なんとなくこれをこういう風に使おう、と、ほややーん、と、思い浮かぶはずです。そういう、慣れている物を使いましょう。新しい物を触ると、たいていは今までにない事が出来たり、今までいろいろ面倒だった事が簡単になっていたりする、と、言われていると思います。しかし、そんなものは幻想です。そんなことができるようになるには、大抵は時間がかかってしまいます。あなたの作りたいプログラムは、今、作りたいのです。今、作りたいというその意欲が重要なのです。そんなことで足踏みなどせずに、気にしないで自分の知っているものだけでガリガリ書きはじめましょう。足踏みなどをするということは、「明日やるよ」と、言っているようなものです。そんなことではいつまでたっても完成しやしません。

4 下らないことは考えるな

プログラムが思ったとおりに動きはじめると、とても楽しい気分になって、勝手に将来の夢が膨らむことと思います。ああいう機能を付け加えると楽しそうだな、とか、こういう事ができるようにもできちゃったりするな、とか、です。そんな事は考えない事です。考えても気にしない事です。だいたい、まだ最初に考えた機能だって全てが実装できているわけじゃないのに、新しい機能を付け加えるなんて、ありえません。そんなことは考えちゃいけません。さっさと最初の最終目的まで作り上げようと努力すべきです。そうです。作り上げる事が重要なのです。だって出来上がってはじめて、あなたの当初の目的が達成されるのですから。

5　やる気がなくなったら、やめちまえ

がりがり code を書いていると、だんだんそのプログラム自体を作り上げる元気がなくなってくることがあります。そうしたら、そのまま捨ててしまいましょう。いいんです。捨ててしまいなさい。そんな作る元気がなくなったようなプログラムなんて、完成させなくたっていいんです。というか、作る元気がなくなったということは、そのプログラムのニーズ自体がなくなったとか、あなたの望む事が変わったとか、そういう理由が背後にあるのです。だって、本当に欲しいプログラムであるならば、やる気がなくなることはありえないはずです。そうです。やる気がなくなってまで、そのプログラムを書きつづける必要はありません。さっさとあきらめて、次のプログラムを書きましょう。その捨てたプログラムを作る事によってあなたが得られた知識はいつまでも残ります。それでいいんです。十分ではないですか。

6　他人に意見を求めよ

調子に乗ってプログラムが書けていたりする時は、いいのですが、たまに、簡単には回避できなかったりする問題が発生することがあります。そうしたら、お友達にそのことについて聞いてみましょう。そのお友達がその問題について興味が無くたってかまいません。相手に自分の問題を理解させるという行動は、自分の考えを整理するのにとても役に立つからです。もちろん、お友達がその問題について興味津々だった場合はもっと便利です。多分、そのお友達は答えを知っているか、答えに近い考えを持っていると思われるからです。第一、お友達は、あなたの質問に興味が無くたって、無下に扱いはしないでしょう。そうでなくても、お友達と話をするというのは、楽しい事です。少し息抜きが出来たということでも十分に効果的です。

7　とりあえず公開せよ

プログラムが思ったとおりに動きはじめると、とても楽しいです。その楽しい気分の時に、そのまま、WebPage でも作って、公開してしまいましょう。多分、自分のプログラムなんて、誰も嬉しいと思ってくれません。それでも、公開しましょう。それで、いいんです。人は、そうそうあなたに面と向かって「クズソフトを公開しやがってこのクズ！」とは言わないものです。陰口なんて、言わせておけばいいんです。観測できないものは存在しないも同じ。というか、陰口を言われるということは、それだけその相手に気にされたということです。あなたの着眼点は少しだけズレていたにしろ、いいところをついていたということです。素晴らしいではないですか。それよりもなによりも、公開するというのはなんとも大きなプレッシャーになります。このプレッシャーはかなり、重要です。公開する、ということを考えると、あなたの code はまた一味違った美しい code となるでしょう。また、伊達でもよいので、たくさんのソフトを公開しているゼェーというのは、ハッタリとしては十分な効果を持ちます。あなたは他人から少し偉く見えるようになるでしょう。以上のように、公開していて損をすることというのは、ほとんどありません。さっさと公開してしまいましょう。

あ、免責事項はきちんと書いておきましょうね。

8　ドキュメントを書け

code を書きはじめたら、がむしゃらに code を書きましょう。けれども、一緒にドキュメントも書きましょう。そのプログラムが何をするのか、コマンドラインオプションは何を受け付けるのか、環境変数を参照するのか、ユーザインタフェースはどうなっているのか。これについては片手間でかまいません。でも書きましょう。なぜなら、きっとあなたはしばらく前に書いた code、たとえば今書いているもののふたつ前に書いていた code の中身なんてものは、きっとおぼろげながらにしか覚えていないでしょう。そういう code というのは、後から読むと、まるで赤の他人、いや、赤の自分の書いた code となっていて、理解できっこないようになっているに違いありません。そういう時に、何をするもので、どういう事ができるのか、とかを残しておくというのは重要です。もちろん、それなりの量のあるドキュメントというものは、後で公開する時にハッタリとして十分に効果的ですしね。

9　本を読め

学校への行き帰り、部屋でぼへーっとしたい時、いろいろと空白の時間というものはできるものです。また、四六時中モニタに向かっていると腰が痛くなります。手も疲れます。肩も凝ります。たまには息抜きもいいでしょう。そういう時には、本を読みましょう。本には先人達の通ってきた道が記されています。特に、先人達の開発したアルゴリズムなどは良い題材です。ほけーっとした時間で、あなたの code をきれいにするインスピレーションを与えてくれるかもしれません。もちろん、大抵の本には、今あなたが作っているプログラムに対する答えなんて物はないでしょう。そんなことは気にしないでいいんです。多分、後で作るプログラムで役に立ちます。暇な時間は本を読みましょう。

10　そして書け、書きつづけよ

code を書くということは、それなりに大変な作業です。データ構造とアルゴリズムを考えて、それを code に書き起こす、それだけの事でも、結構ミスを繰り返す物です。しかしながら、そういうミスを繰り返す事によって、あなたのプログラミングのセンスはよくなることはあっても、悪くなることはないでしょう。実際、code を書きつづけられるという能力は、かなり、重要

です。どんな時でも code を書いていたゼェーというのは、精神的に強くなれます。ちょっとやそっとの code なんてのは、書けない事はない、とか、思って書きはじめる原動力になります。良い意味の強気になりやすいということです。いかなる時でも書きつづけるというのは、大変です。時には辛い事もあると思います。でも、そういう時こそ、ニヤリと笑って書きつづけましょう。端からみれば男です。物が完成するまで走り続けるのです。なにかをやり遂げるという事は素晴らしい事です。途中で投げ出す事は誰にでも出来ます。やり遂げる、これが、大事なのです。code を書きつづけなさい。完成に向かって。

11　最後に……

趣味プログラムにかける時間があるって事は、とても素晴らしい事ですね。

デバッグ

山本 ⟨yamamoto@mma.club.uec.ac.jp⟩

2001 年 11 月

1　はじめに

「デバッグ」と言えばプログラムの「バグ」を取り除くこと当たり前のように使っているが、まだ当時１９４７年にはそんな言葉はなかった。リレー式計算機エイケン MARK　に紛れもない「虫」そのものが突っ込むまでは。その「蛾」はリレー式スイッチに挟まり MARK　を故障させた。そして、その亡き骸は故障の原因として業務日誌に貼り付けにされて、スミソニアン協会の国立アメリカ歴史博物館にそれが展示されてる。以後、エイケンに仕事の進捗状況を尋ねられるたびに、彼女、ホッパーは「虫を取っています」と答えた。また、Mark II はしばしば動作しなくなったが、そのたびに他のオペレーターたちも「デバッグ」を口にした。

これが、デバッグの由来。って蛾の死体をセロハンテープでノートに貼るか？普通。ページを開くたびに蛾の死体を目にするのは想像したらなんとも嫌な気分になる。今じゃ水分が無くなってゾンビ化[*1]してると思うが、当時はまだ瑞々しかったと思う。（蛾の死体に瑞々しいって言葉使うのにも気が引けるものがあるが。）これを読んでくれてる人も今まで生きてて目にしたことがあると思われるが、例えば、道端の犬の糞なんてその典型例だ、うんこはうんこでも乾燥して干からびてるのと、まだ水分を保っている状態のものでは不快度のレベルが違う。しかも湯気が立ってたりしたときにはもう、最悪だ。強烈な悪臭とその不快感で即刻その場を立ち去れと自分の脳が命令する。他にも、蛙やミミズが道路で横たわっているのでも同様だ。それを、それを貼り付けて保存するなんて普通じゃないな、こいつらは。最初これについての記事を読んだとき、そんな経緯があったんだとちょっと感慨深く思ったが、スミソニアン博物館のページで、写真でその蛾の死体の貼り付けてるノートを見ると、そんな思いも吹っ飛んでしまった。

これではあまりにも短いので、これに関連した物を紹介すると、

エイケン MARK　は、ハーバード大の H. エイケン が、１９４３年にＩＢＭと開発した Mark Ｉの後継機。Mark Ｉは数値も計算式もカードにより入力し、記憶装置に入力後、２３桁の１０進数の乗算を数秒で行う自動逐次計算方式で、現在の計算機の原形と言えるものだった。ただし、機械内部ではリレー回路による自動処理が行われていて、一回の計算ごとにワイヤーをつなぎ変えてプログラムをセットする（ワイヤードロジック：Wired Logic）電気機械式の計算機で、２進数による初の計算機。リレー（継電器）は、電磁石が動力で、回路に流れる電流を制御し入切断する。接点がくっついたり外れたりするスイッチの開閉状態をそれぞれ１と０に割り当てることにより計算。ワイヤードロジックは、機械語の命令を、配線論理によって実現する方法。現在は RISC ベースの CPU で利用される制御方式。また、一般に公開された最初のコンピュータで、コンピュータというものが広く認識される契機を作った。

このエイケン Mark　って Computer でいうと古代ってことになるのか？これの前なんか歯車式（石器時代）だし、この後に電子式（これが中世か？）の ENIAC あたりが登場することを考えれば。ノイマン型以前の Computer について知らなかった（以後も大して知らないが）自分にとっては、どれも古代遺産と言っても、その印象はさほど変わらない。

グレース・マレー・ホッパーは、ニューヨークのバッサ・カレッジで数学の教授で、海軍の数学アドバイザーとして 1944 年、ハーバード大学の Mark I プロジェクトに参加、Mark I のプログラムとマニュアルの作成を担当。そして保険業界で初めてコンピュータを導入したプルデンシャル社に Mark I を売り込んだ、コンピュータ業界最初で最大の営業マン。その後ハーバード大学で数学を教えて、プルデンシャル社ではプログラミングを手伝っていたが、コンピュータでの事務処理を実現させるため、1949 年、エッカート・モークリ社に入社。そこで UNIVAC I 用に英語を用いてデータ処理を行える言語 FLOWMATIC（フロウマティック）を開発。

FLOWMATIC は、ソースコードに英文を用いた最初のコンパイラ型言語。1959 年 4 月、彼女は国防総省のプログラミング担当として CODASYL に参加。こうしてデータ処理言語 FLOWMATIC を土台に、COBOL が生まれた。

1986 年 8 月 14 日、海軍少将として 79 歳で退官。米海軍史上最年長の将軍だった。その後も、DEC 社の上級コンサルタントとして活躍。1992 年 1 月 1 日息を引き取った。享年 85 歳。

この人、鬼のようにすごい人だな。実際、アメージング（驚異の）・グレースって呼ばれてたらしいが、彼女は電子工学には長けてなかったそうだが、それが逆に一般の人にわかりやすい言語を生む要因のひとつになったそうだ。１９５０年前後に女性で教授ってのも、あまり想像できないし、学者としてにせよ、軍に入り、プロジェクトにおいて大きな役割を果たしたと言うのは、アメリカだからなのか（アメリカでも２次大戦前後で女性の地位に変化があった）。そう考えると日本は後進国だったんだと思えてくる。

COBOL は、ビジネス言語で, データ処理は科学計算とは異なり、計算自体は単純であるが、扱うデータが複雑となるし、処理を行う人が数学に慣れていないことが多いので、数学でなく英語で記述できるように開発された。

ホッパーによって規格化が提唱、1959 年の会合で共通ビジネス言語の望ましい特徴がリストアップされ、CODASYL（データシステム言語委員会）と呼ばれるコンピュータの専門化のグループによる統一言語の開発が開始。六ヶ月で COBOL が誕生した。

Computer の歴史に興味があったり、もっと詳しく知りたい人は、下記のページなどを見たらいいと思います。多分新たな発見があると思います。汚い話もあってすいませんでした。

[*1] ミイラ化？ (編)

参考にしたもの

HOTWIRED APAN http://www.hotwired.co.jp/news/（デバッグについてのページは再発掘できず）
デジタル進化論 http://www.nhk.or.jp/denno/rekishi/
コンピュータの歴史 http://isweb10.infoseek.co.jp/novel/l-branch/gaku-comhistry.html
ソフトウェアの２０世紀　http://www.shoeisha.com/book/pc/20c/
広島経済大学　情報処理概論　コンピュータの歴史 http://members.tripod.co.jp/miku2001/ip02.html

セルオートマトンのおはなし

ottan

2000 年 4 月

1　まえふり

そそるネタを思いつかなかったので、ToDo list にあることをそのまま原稿にしてみる。

2　セルオートマトンってなに？

セルオートマトン (cellular automaton 略して CA)ってのは、ある種の理論であり、シミュレーションもしくは数値計算の手法なんだけど、感じとしては、ライフゲームを思い浮かべてもらうと一番分かりやすい。

ライフゲームを一般化して自然現象とかにも応用しようとして出来たモノってぐらいに考えればいい。ほんとは逆なんだけどね。

あと、”オートマトン”とは名前についてるけど、状態遷移図かいたり 正規表現解析したりする有限オートマトンとは見た目あんま関係ない。理論的背景ではあるけど。

言葉の意味は、”次の状態がセル (cell) [*1]単位で独立して自動的に生成される” ぐらいかな。

3　なりたち

元ネタを考えたのは、あのノイマンさん。

ENIAC の真空管の故障による機能停止[*2]の多さに辟易して、人間の脳のように、部分的にどっかの細胞に寿命がきても別の細胞がその役割を代行している間に新しい細胞がその役割を担うという自己修復機能をつけたかったってのががキッカケらしい。あと、人間の脳をモデル化するのに有限オートマトン理論を発展させてなんとかしちゃろうってのもあったようだ。

で、ウラムとかいうおっちゃんから均一なセルを敷き詰めたセル空間というアイデアを得て、CA 理論を考えついたようだ。

しかし、ノイマンさんも忙しい人だったようで考察だけで終わってしまい実装するところまではいけず、50 年代、60 年代は一部の人々によって細々と研究されるにとどまってしまう。

そして、70 年代のライフゲームの流行[*3] を契機にやっとメインストリームにでてくる。

その後、80 年代にはウルフラム[*4]の自然現象への応用と、流体なんかへも応用できるような理論の拡張[*5]を行ったりして、現在にいたるわけだ。

4　しくみ

基本原理は単純で、

- 同じ大きさの均一なセルにのしきつめられた空間を想定
- 各セルは k 種類の状態をとる
- 次の時間のセルの状態は現在の自身と近く (近傍) のセルの状態だけの局所的な規則できまる
- 出来上がるパターンはセルの初期条件と適用する規則によってきまる

の 4 つだけ。こんな単純なルールだけなのだけど、扱う対象によっては自然現象をあらわした微分方程式[*6]をちゃんと解いたときと同等な結果が得られ、またそれぞれのセルは付近のセルのデータさえあれば計算できるので簡単に並列化できる。

5　ライフゲーム

で、ライフゲームの場合は、

- 各セルは生と死の 2 種類の状態をとる

[*1] この文章では、セルという言葉をコンピューターサイエンス的な単位元というか、自然現象を再現するモデルの最小単位というような意味で使っています。で、生物学的な cell を細胞と表記しています。

[*2] きっと箱庭で災害が起こる確率に似たようなものでしょう（ ^^; 個々の発生率は低くても、システム全体や ある程度以上のスパンで見て、べき乗してくと結構な確率になる。

[*3] この頃のハッカー達のよいなぐさみものだったらしい。ネタ本 [1] にはライフゲームがハッカー文化の発展とコンピューターグラフィックスの発達をうながし、ゲームソフト開発の契機となったとまで書いてある...

[*4] Mathematica の作者でもあるらしい

[*5] 格子ガスオートマトン法とか、こっちは各ステップの対象が個々のセルではなく格子の交差点 (オセロから囲碁へ、と言えばわかってもらえるかな) で、その格子の上でのみ粒子を動かすというモノ

[*6] 微分方程式は連立 1 次方程式を解くこと、つまり行列の計算に帰結できるのだけれど、行列の計算てのは各々の要素の独立性は低いので、並列化するのは一苦労のようだ

- 次のステップのセルの状態は隣接する 6 つのセル[*7]の状態から以下のような規則によって決まる
  (1) 隣接する 6 つのセルでで生きてるセルの合計を sum とする
  (2) $2 <= sum <= 3$ のとき、生きてるセルは生きつづける
  (3) $sum >= 4 \quad || \quad sum <= 1$ のとき、セルは死ぬ
  (4) $sum == 3$ のとき、死んでるセルは生き返る

というような感じになる。
たった、これだけのルールで、ライフゲームの複雑な動きが作り出される。
これは、周囲にほどよい数がいるとき生存できて、過疎や過密のとき死滅し、最適な条件のとき生まれるという、動物やバクテリアの環境を単純化したモデルだったりする。

6　しましま

ライフゲームだとゲームとしての印象が強く自然現象の再現という部分ではちょっと薄いので、もうすこしそれっぽいモデルを取り上げてみよう。
で、動物の体表面の紋様 (シマウマのしましまとか) の生成をシミュレーションしてみる。

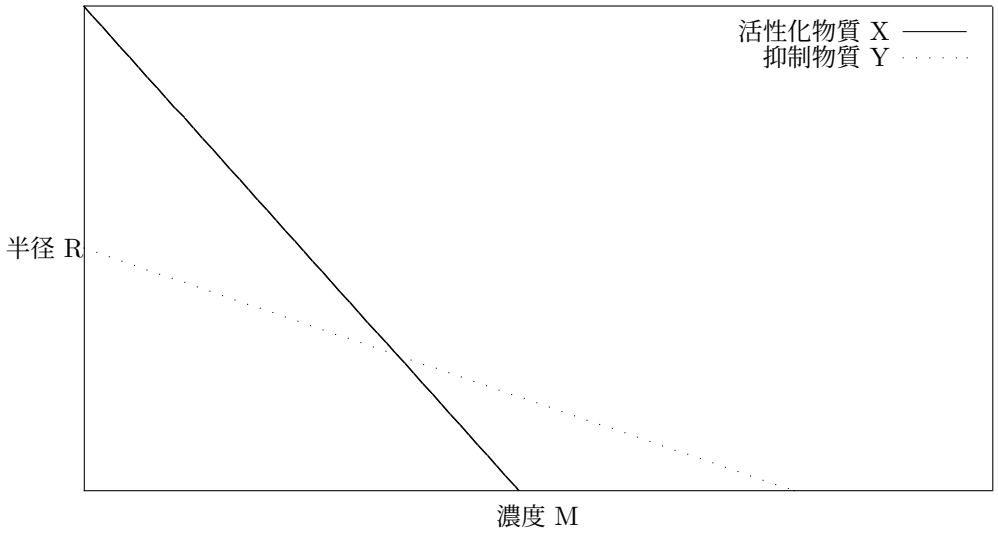


図 1　微分方程式なグラフ

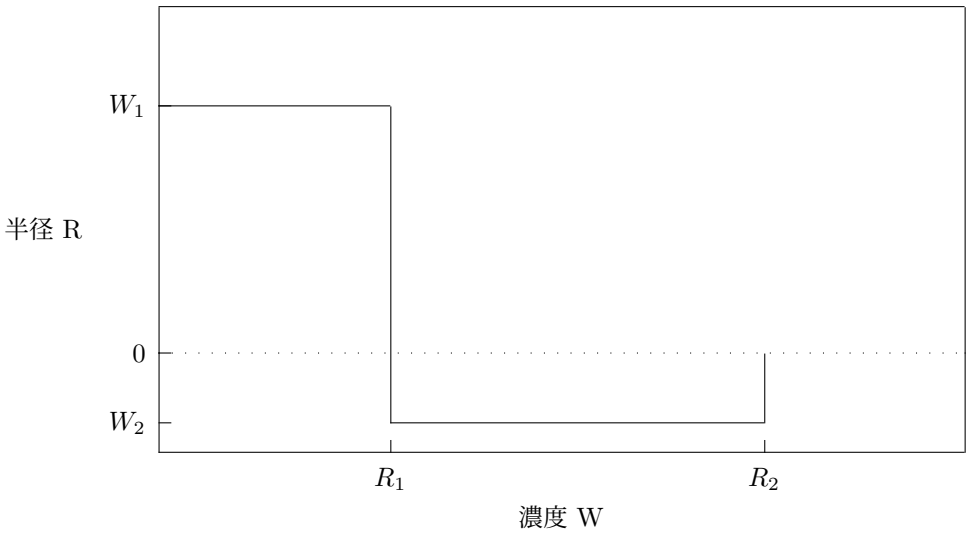


図 2　デジタルなグラフ

[*7] 上下左右斜めの 6 つをムーア近傍という、ちなみに上下左右の 4 つはノイマン近傍という

シマシマやブチってのは、メラニン色素の発色した細胞が集まったところだ。
で、発色した細胞は、狭い範囲で強い影響を与える周囲の細胞を活性化して発色を促す物質と、薄く広く影響を与える周囲の細胞を抑制して発色を抑える物質の両方をだす。発色してない細胞は周囲に何も影響を与えない。
ようするに、発色した細胞は近所同士で活性化を促しあい、ある程度遠いとこには活性化を抑制しようとするので、発色してる部分はある程度固まり、シマシマやブチになるわけだ。
発色した細胞からの影響をグラフにしてみると図 1 のようになるが、このままだと、フツーに微分方程式を特のとか湾内ので、図 2 のように離散化して処理する。
これを基本原理のパターンに沿ってまとめてみると、以下のような感じになる。

- 各細胞は活性化して発色した状態と活性化しておらず発色してない状態の 2 種類の状態をとる
- 次のステップの細胞の状態は
  (1) 活性化させられる半径内に存在する発色した細胞の総数を x とする
  (2) 活性化させられる半径以上 抑制される半径以内 に存在する発色した細胞の総数を y とする
  (3) 活性化させる影響力を係数 W_1 とし、抑制する影響力を係数 W_2 とする
  (4) $W_1 * x + W_2 * y > 0$ なら活性化し発色する
  (活性化した細胞は活性化したまま)
  (5) $W_1 * x + W_2 * y == 0$ ならそのまま
  (6) $W_1 * x + W_2 * y < 0$ なら抑制され発色していれば消える
  (活性化していない細胞は活性化しないまま)

こいつを繰り返してみると、5 ステップ程度で平衡に達して安定なパターンが得られる。

図 3　しましまの図

ちゃんと、それっぽく見えるよね。こんな単純な規則からこんな自然現象に近い複雑な結果がでてくるなんて、かっこいいとおもいません？

7　まとめ

ここでは図 1 のアナログで連続な微分方程式から図 2 のデジタルで離散的なオートマトン法モデルへの展開をさらっと流してしまっていますが、実際に自分でモデルを組んでシミュレーションするときはこの辺が味噌になってくると思います。
ここに書いてある内容はネタ本 [1] の縮小再生産なだけのような気もするので、ちゃんと知りたい人はそっちもあたってください。
あと、実際の描画には CA 用統合環境って感じのソフトである [2] を使っています。

参考文献

[1] 加藤恭義 光成友孝 築山洋 共著, セルオートマトン法, 森北出版
[2] dana@rucs.faculty.cs.runet.edu,
The Cellular Automata Simulation System,
`http://www.cs.runet.edu/ dana/ca/cellular.html`

書評

ikidou@mma.club.uec.ac.jp.

2001 年 4 月

概要

新年度も始まり学習欲が瞬間最大風速を記録している人も多い[*1]と思います。この文章を読んでる人は何かしら計算機に関して興味をもっているのではないでしょうか? その興味を満たしてくれる情報は様々な方法で入手可能です。例えば、Web、マニュアル、詳しい人、プログラムソース、書籍等々があります。それぞれ長所短所がありますがここでは書籍に焦点を当て、個人的に面白いと感じている書籍を紹介します。

1 書籍の選び方

大学に入学し計算機関連の技術書を生まれて初めて購入したとき、値段の高さに驚いたものです。安くても 2k 前後、高いと 7k 前後もします。一冊買うと数日分の食費が無くなってしまうのは貧乏学生にはつらいですよね? 単純に値段が高いからといってその本がわかり易いというわけでもなく、高くてハズレの本を買った時は自己嫌悪してました。

そんなわけで書籍の選び方の自分ルールを書いときます。

- 興味や疑問をはっきりする
  一見あたり前なのですが「なんとなく必要そう」という理由で買っちゃう場合が結構ありました。そうした本は大抵読まれることなく「積読 (つんどく)」状態になってます。自分の中で疑問や興味を具体的にしてから、本を選ぶようにすれば良いでしょう。プログラムをしたこと無い人が「C 言語を勉強したいなぁ」と思って「有名だから」という理由で K&R の C 言語本を買ったら多分挫折する[*2]でしょう。「UNIX 上の C 言語で Hello World プログラムを書きたい」等と自分の要求を具体的にしたほうが本の選択の助けになります。
- 立ち読みして内容を確認
  自分が本に求める物がはっきりしたら本を立ち読みしてみましょう。知りたいことが自然に頭に入ってくるような本があったら当たりです。買っちゃいましょう。もしなかったら無理して買わない方が良いです。
- 欲しい本は買う
  良い本というのは何かと参照する機会が多いです。その度、図書館で借りるのも一つの手段ですが、買った方が便利です。良い本を作った出版社、著者のためにも買っちゃいましょう。

2 読まない方が良い本

- 情報に嘘がある
  これは判別が難しいですが、運悪くこういう本にあった場合は野良犬に噛まれたと思って下さい。本に書いてある話はかならずしも真実では無いと常に頭にすみにいれておきましょう。
- 日本語が変
  文章が読みにくい本は内容が正しくても駄目です。苦労して読んで脳ミソが疲れた割には得るものが少ないと悲しくなります。ただ、知りたいことが書いてあるのがその読みにくい本だけであれば我慢するしか無いですけどね。
- 自分の知識が著者の対象としている前提知識に達していない
  これは本が駄目というわけではないですが、無理して読む必要はないでしょう。むしろその分野に拒絶反応がでる可能性があるので読まないのが吉です。成長してからのお楽しみということで。

参考文献

[1] 現実的な C プログラミング. Steve Oualline 著/岩谷宏訳. ソフトバンク.
C 言語の勉強をするために買った本です。初心者向けにわかりやすく書いてあります。文法の話だけではなく、上手にプログラムを組むためのプログラミング技術[*3]も説明されてます。現在は第 3 版がオライリー・ジャパンより出版されています。

[2] エキスパート C プログラミング. Peter van der Linden 著/梅原系訳. アスキー.
C 言語でプログラムが書けるようになったら読んでみましょう。楽しみながら C 言語に対する理解を深めることができます。翻訳者によるこまかなツッコミが面白いです。

[3] UNIX システムプログラミング. 羽山博著. オーム社.
システムコールを使ったプログラムを組むことで、システム (UNIX) がどういう物かがわかります[*4]。システムコールを使ったプログラムを解説する本では珍しく初心者向けなので、システムコール初めて (は〜と) という人にお薦めです。

[4] UNIX ネットワークプログラミング. 金内典充, 今安正和共著. オーム社.

[*1] 大抵ゴールデンウィーク後風力が急激に弱まります。

[*2] K&R の本はプログラム経験のある人向けだと思います。

[*3] 効果的なコメントの書き方、わかりやすい変数名、プログラムの設計法等々。

[*4] プロセスが何かとかファイルが何かとか...

[3] の姉妹本でネットワークプログラミングに特化した内容です。同じく初心者にお薦めです。
[5] 詳解 UNIX プログラミング [新装版]. W.Richard Stevens 著/大木敦雄訳. ピアソン・エデュケーション.
[3] よりも詳しくシステムプログラムについて書いてあります。今まで疑問だったことがこの本を読む事で次々解決していきます。高いけどそれだけの価値はあるとおもいます。
[6] プログラミング作法. Brian W.Kernighan, Rob Pike 著/福崎俊博訳. アスキー.
上手なプログラムを組みたい人は必読です。私は洋書を読んでいたのですが、訳本が出たのでそっちを読みました。編集協力に harada さんの名が:)
[7] 珠玉のプログラミング. Jon Bentley 著/小林健一郎. ピアソン・エデュケーション.
[6] と同じで上手にプログラムを組みたい人向け。多少アルゴリズムよりかも。只今読んでる途中ですが目から鱗が落ちること度々です。
[8] 情報処理入門コース 2 オペレーティングシステム. 清水謙多郎著. 岩波書店.
OSって何やってるのか知りた〜いという人にお薦め。特定の OS に特化した内容ではないので、OS の概念を幅広く知ることができます。著者の清水先生は電通大で助教授をされていたこともあり、画面サンプルに uec の文字があったりします。
[9] UNIX カーネルの設計. Maurice J.Bach 著/坂本文, 多田好克, 村井純訳. 共立出版.
UNIX の内部構造を知りたい人向け。System V 系の話ですが BSD 系の人でもあまり気にならないと思います。アルゴリズムがソース形式で紹介されており読みやすいです。
[10] LINUX デバイスドライバ. Alessandro Rubini 著/山崎康宏, 山崎邦子訳. オライリー・ジャパン.
カーネルプログラムしたいけど良くわからないという人向け。Linux のカーネル 2.0 系ですけど参考になるでしょう。ソース例が多く実際に動作させながら読むのがお薦めです。

カバーの説明

本書のカバーの生命体は、ネットブックスタイラーです。ネットブックスタイラーとは、路上電算（ストリートコンピューティング）を営む、背面から人間のはえたコンピューターです。その体は軽快なフレームと電力的パフォーマンスの高いパーツでできており、UDX において八時間動作し続ける耐久力があります。主に 秋葉原 に出没します。ときどき背面からはえた人間を駆使して リナカフェに向かいますが、これは、その体のバッテリーの充電のためと言われています。

ネットブックスタイラーはスリムな生命体で、背面から生えた人間の平均的な大きさは 156 センチ 程度です。外出先でも、巧みに ワイ○ックス や イーモバ○ル の電波を食料としています。

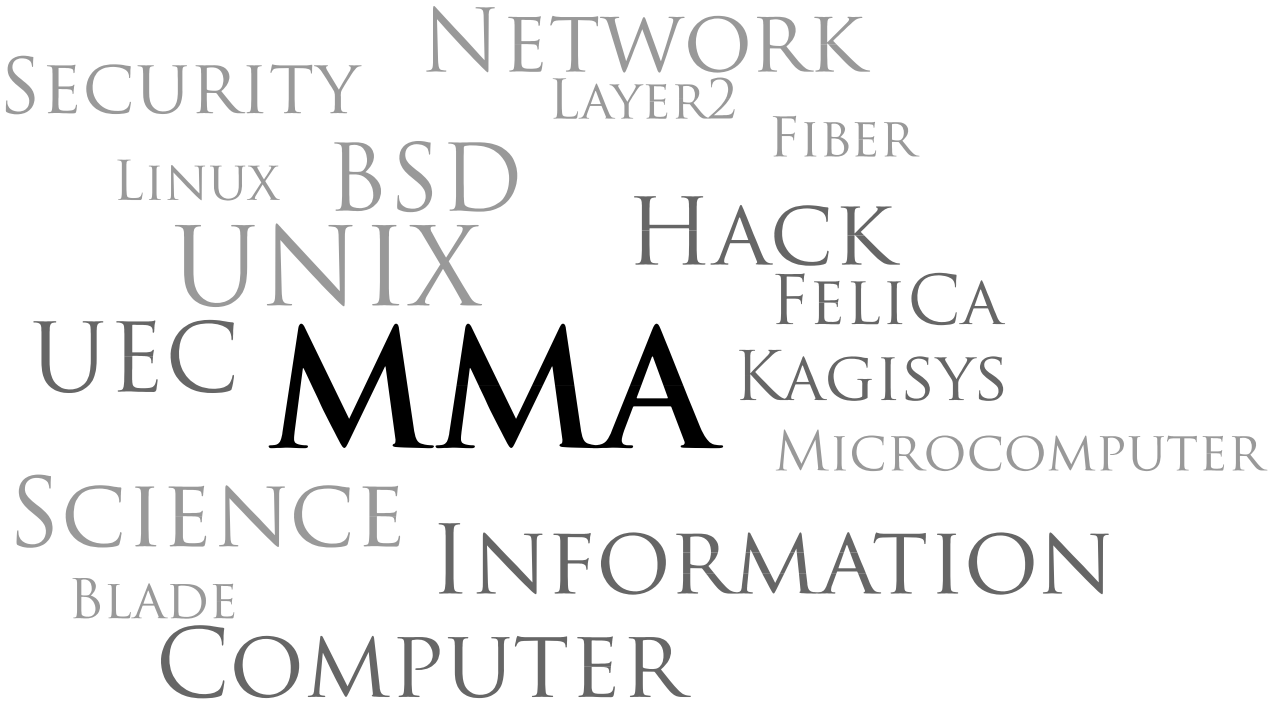
百萬石 2010年 春号 © 電気通信大学 MMA

2010 年 4 月 4 日　初版第 1 刷発行　【本書の無断転載を禁ず】

著　者　kyogoku42, moechar, takkun, yamamoto, ottan, ikidou
編集者　kyogoku42, ytoku
発行者　電気通信大学 MMA
発行所　電気通信大学 MMA 部室
〒182−8585 東京都調布市調布ヶ丘 1−5−1 サークル会館 2 階
http://www.mma.club.uec.ac.jp/ (IPv6)
http://delegate.uec.ac.jp:8081/club/mma/ (IPv4)
印刷所　電気通信大学 MMA 部室
製本所　電気通信大学 MMA 部室
表　紙　kyogoku42

Hyakumangoku

Hyakumangoku 電気通信大学 MMA の機関誌



MMA　発行所／エムエムエー