IT （情報通信技術）と数学、数学教育そして情報教育

IT, Mathematics and Education

亀 子　　正 喜
KAMEKO　Masaki

1．序

現代社会は「高度情報化社会」であるといわれる。われわれの日常生活はコンピュータに代表される情報処理技術に支えられている。しかし、コンピュータの存在を大部分の人間がより身近に強く感じるようになったのは安価で高速なコンピュータが普及し始めた最近のことであると言ってもよいであろう。そして昨今のいわゆるIT（情報通信技術）革命は、この流れを決定的なものにした。ここ数年でコンピュータとそれに関連した情報通信技術は一般の人間にとっても急速に身近なものになってきて、われわれの生活習慣そのものをかえつつある。

私自身についてはというと、現在は富山国際大学地域学部という学際的な色彩の強いところで「コンピューターリテラシー」「基礎統計学」「情報数学特論Ⅱ」などの科目を教えている。このような状況なのである意味で自然の成り行きとして「情報」と「数学」そして「教育」について考えることが多い。ここではITと数学、数学教育、情報教育について日ごろ考えていることの一部をまとめてみる。

この小論ではまず、コンピュータリテラシーという言葉が現実には何を意味について考え直してみる。ここではコンピュータを単に便利な道具としてみるという視点があることを確認する。つぎにコンピュータを抽象的なアルゴリズムの実行機械とみて数学との接点について考えてみる。さらに、アルゴリズムの実行機械としてコンピュータをみるとき、どのようなことを数学教育、情報教育の中で取り上げるかについて、連立一次方程式の解法を例にとって考えてみる。そして最後にアルゴリズムと実際のプログラミングの違いについてふれておく。

2．「コンピューターリテラシー」とは？

ＩＴの急速な進歩に伴い情報教育の必要性が叫ばれている。インターネットの普及とあいまって銀行業務だけでなく行政サービスなどのオンライン化までもが計画され、実行されつつある。このような状況では、国民の大部分がインターネットへのアクセスをもつことが必要になってくる。そしてそのための道具としてのコンピュータ、いわゆるパソコンの重要性が認識されてきている。これを受けて、多くの小学校、中学校、高等学校、大学でコンピュータの基本的な利用方法が教えられている。多くの大学では「情報」という文字が学部や学科の名前の一部に取り入れられているし、コンピュータ関係の科目が開講されている。また、同時に、コンピューターリテラシーと題した本も多数出版されている。コンピュータをある程度自由に使える能力、すなわちコンピュータリテラシーが求められているのである。そのコンピュータリテラシーの内容はというと多くの場合、コンピュータの基本的な使い方としてワードプロセッサーや表計算ソフトウェアの使い方、インターネットでの情報収集、情報発信の方法などである。富山国際大学の場合は「コンピューターリテラシー」という科目名の１年次の必修科目で、コンピュータとコンピュータネットワークの利用方法が教えられて

いる。富山国際大学における「コンピューターリテラシー」の2001年度の教科書はつぎの9章からなっている。

1. 実習用ＰＣ教室の利用について
2. WINDOWS の操作
3. キーボードの操作
4. 日本語入力
5. インターネット
6. 電子メール
7. 文書作成ソフトウェア
8. 表計算ソフトウェア
9. ホームページ作成

また生涯教育の一環としてのパソコン教室、ＩＴ講習会なども上の「コンピューターリテラシー」と同じような内容でかなり盛んである。

上述の意味でのコンピュータリテラシーは着実に向上しており、コンピュータの利用者数は着実に増えてきている。新聞によると小学生の発表に関しても、模造紙に発表内容をまとめて発表する小学生は減り、パソコン上でプレゼンテーション用のソフトウェアを用いて発表内容をまとめる小学生が増えているというし、いくつかの小学校では小学生がホームページを作成して情報発信している。小学生と大学生がまったく同じことをしているというわけではないが大学生が「コンピューターリテラシー」で小学生と同じ様なことを学んでいる。このように「コンピューターリテラシー」ではコンピュータを小学生にもわかる形で教えていることになる。そこでは当然コンピュータの便利な道具としての使い方が主なテーマになる。

コンピュータの内部構造やその動作原理を理解するにはかなりの知識が必要になる。それに反して、コンピュータの使い方そのものはコンピュータの動作原理を知らなくても実際に使ってみればすぐわかる。それゆえに「コンピューターリテラシー」ではコンピュータの内部構造や動作原理にはあまり触れずコンピュータの使い方をワードプロセッサーの使い方や表計算ソフトウェアの使い方を実習することによって教えている。このように「コンピューターリテラシー」ではコンピュータを、その内部構造についてはまったく知ることのできないブラックボックス、「魔法の箱」として教えている。

この方法でコンピュータの使い方を教えることはできるし、とりあえずコンピュータを利用できるようになりたいというのであれば手っ取り早いよい方法である。ワードプロセッサーで文書作成できないようでは、その学生を雇おうとする企業も限られてくるであろうから、職業訓練の一部としてコンピュータの使い方を教えることには意味がある。実際に多くの「パソコン教室」がこのためにある。この意味で、コンピュータを単なる便利な道具、「魔法の箱」として、入出力装置の取り扱いを「呪文」として取り扱いその使い方だけを教えることは悪いことではない。

３．アルゴリズムとコンピュータ

しかし、コンピュータが他の便利な道具と大きく異なる点がある。それはコンピュータがアルゴリズムの実行を行う機械であるという点である。これこそがコンピュータの本質であるはずなのだが、上の「コンピューターリテラシー」の教え方ではこのアルゴリズム実行機械としての面が見過ごされてしまう。

コンピュータをアルゴリズム実行機械とみなす場合にも、コンピュータ内部でのデータの表現形式、ＣＰＵの動作原理などに触れずにコンピュータを「魔法のアルゴリズム実行機械」として取り扱うこともできる。そして、コンピュータの重要性、その可能性について考えるためにはこのアルゴリズム実行機械としてコンピュータとアルゴリズムについての知識と経験が必要になる。それゆえにアルゴリズム実行機械としてのコンピュータとアルゴリズムについては「コンピュータリテラシー」と同時またはその後に後期中等教育または高等教育で学ぶべきものである。実際に、高等学校での数学の一部に選択科目として簡単なＢＡＳＩＣによるプログラミングが含まれている。四則演算、べきの計算、三角関数を用いた三角形の面積の計算などに公式に数値を代入してみる、簡単な数列を生成してみるなどの話題が取り扱われている。これは三角関数や数列などの数学を学んでから、計算の一部をコンピュータにやらせてみるという形になっている。コンピュータのアルゴリズム実行機械としての使い方を教えているのである。

これに対して、コンピュータにアルゴリズムを実行させることによりコンピュータをアルゴリズム実行機械として使うことで数学を教えること、研究することもおこなわれている。物理学などはコンピュータによるシミュレーション、数値実験などが盛んであるが、数学、たとえば代数、においてもコンピュータによる計算実験が可能である。ここではグレブナー基底というものがあることを紹介しておきたい。

D. コックス、J. リトル、 D. オシー著の大学院生向けの参考書「グレブナー基底1」、シュプリンガー・フェラーク東京（2000年）の序文には次のようにある。

「近年、多項式方程式系を扱う斬新なアルゴリズムの発見は、安価ながら高速なコンピュータの急激な普及と相まって、代数幾何の研究と実践にささやかな革命を巻き起こした。」

上の革命は数学におけるＩＴ革命の一部分ということができる。また同じ著者による学部学生向けの教科書 “Ideals, Variables, and Algorithms” Springer (1998) では、上の「斬新なアルゴリズムの発見」とそれを実行できるほど高速な安価なコンピュータの出現は、手で計算を行うには複雑すぎる例について計算を通して研究することを可能にしたと同時に、以前には高度な抽象代数を必要とするために大学院レベルでしか扱えなかった話題を学部学生に19世紀風の、計算と例を強調した方法で教えることができるようになったとある。このように、アルゴリズム実行機械としてのコンピュータがアルゴリズムを強調しつつ数学を教えること、研究することを可能にしたのである。

４．連立一次方程式の解法、情報教育と数学教育

コンピュータの本質についてある程度考えることができるようになるためには「コンピューターリテラシー」とは別に情報教育の中でコンピュータをアルゴリズム実行機械としてとらえ、アルゴリズムとはどのようなものかを教えることが必要である。また、数学においてもコンピュータをアルゴリズムを解釈し実行する機械として取り上げることはグレブナー基底の例が示すように大いに意味がある。これについて、以下、数学、とくに抽象代数、を考えるときにコンピュータをどのように利用できるか、その逆にアルゴリズム実行機械としてのコンピュータとアルゴリズムについて教えるときに数学、とくに抽象代数の問題をとりあげて教えることにより相乗効果が期待できるかについて考える。具体的な問題として、ここでは連立一次方程式の解法について考える。

連立一次方程式は線形代数の基礎であり、多変数のデータを扱う際の重要な道具のひとつで

ある。たとえば、中学校における数学のなかでも、2つの直線の交点の計算などにも自然にあらわれてくる。このように連立一次方程式は数学の中でももっとも基本的なもののひとつであり、中学校レベルの数学ですでに教えられている。2直線の交点の計算のような変数の数の少ない簡単な連立一次方程式の解法は単に代入により変数の数を減らすことにより行われる。しかし、中学校レベルの数学のなかでのその解法は、ある意味では行き当たりばったりの解法である。それゆえに、変数の数が増えてきたときにどのように対処すべきかは中学校レベルの数学の知識だけではわかりにくい。

はきだし法は変数の数が多い連立一次方程式を取り扱う場合にどのように対処すべきかという問題への解のひとつである。考え方そのものは一次方程式以外の知識を必要とするものではないが、実際には、変数の数が多い場合のはきだし法は大学レベルの数学（線形代数）で取り扱われているのが現状である。線形代数はベクトル空間や線形写像を取り扱うものである。そして、はきだし法はそれらの概念を扱うための計算上の道具であり、ある意味で線形代数の出発点である。しかし、ベクトル空間や線形写像の概念とは切り離して、アルゴリズムとしてのはきだし法だけを学ぶことも可能である。

さて、定数項のない連立一次方程式には変数 x[1], ... , x[N] と一次方程式とその係数があるが、変数の名前は連立一次方程式の本質に関係がないのでその係数だけを抜き出して並べた行列でもってこの連立一次方程式をあらわすことができる。この場合、連立一次方程式の m 番目の方程式が m 行目の行ベクトルに対応するとし、n 列目が x[n] の係数に対応するものとする。

はきだし法そのもので要求される行の基本変形と呼ばれる操作は

(1) 1つの行ベクトルにゼロでない定数倍をかける
(2) 1つの行ベクトルを定数倍して別の行ベクトルにたす
(3) 2つの行ベクトルを入れ替える

の3つの操作だけである。これらは対応する連立一次方程式における操作を考えれば連立一次方程式の形は変えてもその解は変えないことはすぐに納得できる。また、これらの操作をプログラムにすることはプログラミングを学習し始めたばかりのものにとってもそれほど難しいことではないであろう。問題はこれらをどう組み合わせて連立一次方程式の解を求めるかということであり、これがはきだし法の本質的な部分である。

はきだし法そのものはこの行列の行の基本変形を繰り返して行列を「対応する連立一次方程式が簡単にとけるような形」の行列に変形するアルゴリズムである。適当な日本語訳がみあたらないので英語のまま用いるが上の「対応する連立一次方程式が簡単に解けるような形」を reduced row echelon form という。これは、

(1) ゼロベクトルでない行ベクトルの一番左のゼロでない成分は1である
(2) 下のゼロベクトルでない行ベクトルの一番左の1は上のゼロベクトルでない行ベクトルの一番左の1よりも右にある
(3) ゼロベクトルである行ベクトルはそうでない行ベクトルよりも必ず下にある
(4) ゼロベクトルでない行ベクトルの一番左の1のある列の他の成分はすべてゼロである

という条件を満たすものである。この条件の中に出てくる各行の一番左の1をピボット（pivot）という。この reduced row echelon form の行列に対応する連立一次方程式では、ピボットに対応する変数 x[s] はそれ以降のピボットに対応しない変数 x[t] の式を用いて

あらわすことができるので、この連立一次方程式の解は「後ろからの代入」の形で簡単に求めることができる。

実際、はきだし法をきちんとした形で提示することは結構面倒である。私自身が線形代数を講義する場合は具体的な計算例を示して説明するだけで済ませてしまうことも多い。たとえば変数の数が4で一次方程式の数が3ぐらいの連立一次方程式を実際に説明を加えながらはきだし法で解いてみる。それから演習問題をたくさんやってみることでなんとなく連立一次方程式は解けるようになる。このはきだし法を、ここではそれに基づいてプログラミングを行うことを前提にアルゴリズムの一部分を書き下してみる。

[前提条件] m 行 n 列まで reduced row echelon form になっている M 行 N 列の行列 A が与えられているものとする。これを n+1 列目まで reduced row echelon form になっている行列に上の行の基本変形を用いて変形する。ただし、m 行 n 列まで reduced row echelon form になっているというのは

(1) 1 行目から m 行目まではゼロベクトルでない行ベクトルで一番左の 0 でない成分は 1
(2) 1 行目から m 行目までの行ベクトルで一番左の 1 のある列のほかの成分はすべて 0
(3) m 行目より下で n 列目から左の成分もすべて 0

であるものとする。

(1) n+1 列目の m 行目より下の成分でゼロでないものが見つかるまで探す。もしも 0 でない成分が見つかればその成分を a とし、その成分の行を r とする。(M 行目まで探して見つけられなければこの行列 A は m 行 n+1 列まで reduced row echelon form になっている。 ここで n+1 が N に等しいならば、行列 A は reduced row echelon form である。)
(2) r が m+1 より大きければ m+1 行目の行ベクトルと r 行目の行ベクトルを入れ替える。
(3) m+1 行目の行ベクトルに 1/a をかける。
(4) m+1 以外のすべて s ($0<s<M+1$) に対して s 行 n+1 列の成分を b とし、s 行目の行ベクトルから m+1 行目の行ベクトルを 1/b 倍してひく。(この操作により行列 A は m+1 行 n+1 列まで reduced row echelon form になる。さきほどと同様に n+1 が N に等しければ行列 A は reduced row echelon form である。また、 m+1 が M と等しければ行列 A は reduced row echelon form である。)

この操作を行列 A が reduced row echelon form になるまで繰り返す。

このアルゴリズムはこのように書き下すと面倒なように見えるが、いざ自分でプログラムを書こうとするとどうしてもこの程度のことは書かなければならないということが理解できるし、上のアルゴリズムは、ほかの問題を解くためのアルゴリズムに比べてもとくに複雑なわけではない。

さきほど数回練習すればこのはきだし法は使えるようになるということを書いたが、それはこのアルゴリズムを書き下すことができるようになるということを意味しない。アルゴリズムをアルゴリズムと意識せずに実行することとアルゴリズムをきちんと書き下すことはべつの問題である。いってみればアルゴリズムを書き下すことができるというのはその

アルゴリズムを理解しているということであり、アルゴリズムが書き下せないということはそのアルゴリズムをたとえ実行できたとしても理解しているとは言いがたい。

はじめにこのアルゴリズムを上のような形に書き下さずに具体例の計算を通して学んでおいて、そして連立一次方程式がある程度解けるようになってから連立一次方程式を解くプログラムを作成しよう。そうするとそれまでアルゴリズムというものを意識せずに連立一次方程式をといていたときよりもより深い理解が要求されることがわかる。そして試行錯誤を繰り返せば、アルゴリズムだけではなく連立一次方程式についてもより深く理解できるようになる。このようにプログラミングという作業を意識することにより注意深く状況を分析しなければならない状況が生まれる。

また、変数の数の大きい連立一次方程式を解くのが難しいことを納得することも実際に手で計算しようとすればすぐにわかるので、プログラミングをおこなってコンピュータにアルゴリズムを実行させればコンピュータの有効性についても一定の理解が得られる。さらに、自力で上のアルゴリズムが書き下せなかったとしても上のアルゴリズムを一歩一歩なにが行われているか確認してゆくことにより、複雑そうに見えるアルゴリズムも理解できることを体験することができれば、アルゴリズムとはどのようなものなのか、連立一次方程式とはどのようなものかということについてより深く考える機会になる。

さらに上のはきだし法とは多少異なるように見えるアルゴリズムでグレブナー基底を計算するためのブックバーガーのアルゴリズム風のものを以下に述べておく。ここでは連立一次方程式を行列に変換することなくそのまま一次式のリストとして取り扱う。変数は x[1], ... , x[N] とする。

(1) リスト内のすべての一次式に対して、その一次式の一番左 0 でない項の係数の逆数をその一次式にかけて一番左の 0 でない項の係数を 1 にしておく。
(2) n を 1 から N まで順番に動かしながら、つぎの操作を行う。一番左の項が x[n] であるリスト内の一次式のすべての対 f, g に対して、 f−g を計算し、0 にならないならばこの一次式に一番左の 0 でない項の係数の逆数をかけて一番左の 0 でない項の係数を 1 にしてから、これを一次式のリストに加える。
(3) n を 1 から M まで順番に動かしながら、つぎの操作を行う。 x[n] が一番左の項である一次式が複数あるならば、ひとつを残して残りのリスト内の一次式をすべて削除する。
(4) n を 1 から M まで順番に動かしながら、つぎの操作を行う。 x[n] の係数が a で、x[n] が一番左の項でないような一次式 f がリスト内にあれば、この一次式 f から x[n] が一番左の 0 でない項であるリスト内の一次式 g を a 倍してからひいて、 f の x[n] の係数を 0 にしたもの、つまり f−a*g に置き換える。

これにより対応する行列が reduced row echelon form になっているような連立一次方程式がえられる。ちょっと見たところではこのアルゴリズムのほうがはきだし法よりも簡単にみえるし、プログラミングも楽そうにみえる。またちょっと手で計算してみればこのアルゴリズムには改良の余地があることもすぐにわかる。

ここで問題になるのは、本質的な問題としてはこのアルゴリズムで得られる解は正しい解なのか、もしそうならば果たしてこのアルゴリズムははきだし法のアルゴリズムと同じ答えを与えてくれるのか、という問題がある。これに答えるためにはこのアルゴリズムでどのような操作が行われているのかが理解できなければならない。これは、アルゴリズムが実行できる、このアルゴリズムが書き下せる、このアルゴリズムに基づいてプログラミン

グができるということとはまた別の話である。その意味ではこれは狭い意味のプログラミング教育からは外れているが、プログラミングというものが本来問題解決のためにあるというのであればこの種の問題も取り上げるべきであろう。一応上のアルゴリズムが正しいアルゴリズムであることの証明をつけておく。

[上のアルゴリズムが連立一次方程式の正しい解法であることの証明]

上の (3) 以外の操作では連立一次方程式の解は変わらないことは明らかであるので (3) の操作でも連立一次方程式の解が変わらないことを示せばよい。一番左の 0 でない項が x[n] である一次方程式が f[n,1], ..., f[n, r] であったとする。一次式 f[n, s]−f[n, 1] はある f[t, u] の定数倍になる。この操作を繰り返せば f[n, s] は f[n, 1], f[t_1, 1], ... , f[t_u, 1] (n<t_1<t_2<...<t_u) の線形結合でかける。したがって、f[n, 1]=0 だけを集めてきて得られる連立一次方程式の解ははじめの連立一次方程式の解と同じである。証明終。

ここですでに述べたもの、まだ述べていないものを取り混ぜて、この小論で連立一次方程式の解法を取り上げた理由をまとめておく。

(1) 連立一次方程式は変数の数の小さいものについては手で計算できる。
(2) 連立一次方程式の変数の数が大きい場合には手で計算するのが困難であるのでコンピュータプログラミングの有効性がある程度認識できる。
(3) 連立一次方程式の応用として線形計画法などの応用数学の話題につなぐこともできる。
(4) 連立方程式の解法のアルゴリズム自体を工夫する余地が充分にある。
(5) はきだし法はグレブナー基底の特殊な場合と見ることができるのでグレブナー基底とその応用に発展させることもできる。(ある程度の数学的知識が必要になってくるが、それでもグレブナー基底の概念は大学院生レベルの内容を学部学生レベルで教えられるようになっている。)
(6) プログラミングを前提にして連立一次方程式の解法を考え直すことにより、アルゴリズムとはなにか、連立一次方程式とはどのようなものかについてより深い洞察がえられる。

5．アルゴリズムとプログラミング

ここでアルゴリズムとプログラミングの違いについて考えておきたい。アルゴリズム自体はコンピュータの内部構造、ハードウェアの動作原理などと切り離して考えることもできる。実際にアルゴリズムの記述は特定のプログラミング言語ではなく擬似コード（ pseudocode ）を用いて行われることも多い。たとえば前述の Cox 他著の “Ideals, Variables, and Algorithms” などでも Pascal 類似の擬似コードが用いられている。しかし、擬似コードは知識のある人間にとっては明確なアルゴリズムであってもコンピュータはそれを解釈してくれないのである。それは特定のプログラミング言語でアルゴリズムを記述するにはそのプログラミング言語の文法規則に従わなければならないからである。そしてそれは時として人間にとってはアルゴリズムを書き下す以上に不自然な作業になる。本来ならば擬似コードを解釈してこれを実行してくれるようなコンピュータが望ましいのであるが、現実問題としてはそのようなコンピュータは存在していない。

アセンブリ言語などのコンピュータよりのプログラミング言語から、C 言語などのいわゆる汎用の高級言語が利用できるようになってきてはいるもののいまだに人間とコンピュータとの間の壁は厚い。コンピュータが有限の機械であることからくる問題や、実行効率を求めることからくる問題があるからである。たとえば、C言語には「整数」と呼ばれる変数の型(種類)があるがこれは数学で「整数」と呼ばれるものとは違う。また、有理数などの

計算もC言語での取り扱いはプログラムをする人間が注意深く行わなければならない。この際に、アルゴリズムそのものではなく、コンピュータの動作原理にたいして注意を払わなければならない。これでは、コンピュータをアルゴリズム実行機械として抽象化して使うことはできない。現実の問題としては、プログラミングはアルゴリズムをさらにコンピュータにわかる言葉に書き直す作業でもある。そしてそれは時としてコンピュータの内部構造、ハードウェアの動作原理を意識することを要求する。このようにアルゴリズムの学習とプログラミングの学習の間には現実には違いがある。

連立一次方程式の解法と、プログラミングの演習と同時に取り上げることを考えてみようと提案したわけであるが、現実問題としては上のようなアルゴリズムとプログラミングの間のギャップなどもあり実際に行うことはそれほど容易なことではない。数学的な内容としてのはきだし法は中学校を卒業していれば充分であろう。また、上にあげたようなアルゴリズムを理解することも中学校を卒業していれば充分であるように思える。しかし、現実問題としてはコンピューターリテラシー教育の不徹底、数学の学力低下などの深刻な問題もあるがそれに加えてさらに、適切なプログラミング言語はなにかという問題がある。これについて考えてみる。

まず、数式処理ソフトウェアについて考えてみたい。数式を取り扱うためのソフトウェアで抽象代数を取り扱えるものは多い。商用の Mathematica, MapleV などいろとある。さらに連立一次方程式とプログラミング言語についての適切な教科書があったほうが望ましいが、現状では、コンピュータプログラミングと同時に数学を学ぶ、コンピュータプログラミングを通して数学を学ぶという類の本はそれなりの数存在する。しかし、これら数式処理ソフトウェアを使って連立一次方程式を解くことはできるが、これらはどちらかというと連立一次方程式とその解法を理解している人を対象にしたソフトウェアである。すでに連立一次方程式を解くための機能が組み込まれている。連立一次方程式を解けばどのような事がわかるか、連立一次方程式の解さえわかればよい、という場合には問題ないが、一から連立一次方程式の解法をプログラミングしようという立場では利用しにくい。

もうひとつの選択肢は汎用のプログラミング言語である。汎用のプログラミング言語の教科書、参考書は豊富にそろっている。数学を同時に学ぶというものもある。しかし、たとえば人気のあるプログラミング言語 C 言語などは、前述のようにコンピュータの内部構造を意識させるし、はじめての人間にはやや不自然な点がある（たとえば main 関数の存在）。また、いわゆる手続き型のプログラミング言語を用いれば有理数、複素数などの取り扱いが不自然で複雑になってしまう。

これに対して、同じ汎用のプログラミング言語でも Ruby などのオブジェクト指向プログラミング言語は数学的な構造を取り扱うときに魅力的な選択肢を提供しているように思われる。数学を考えるときには自然に構造を考えるのであるが、この考え方はオブジェクト指向プログラミングの考え方に共通するものがある。配列に定数倍と足し算を定義すれば自然にベクトルになる。さらに掛け算を定義すれば一変数の多項式にすることもできる。このように抽象代数における数学的な対象はオブジェクト指向プログラミングの中でクラスの定義を考えるときに格好の材料を提供しているように思える。それゆえにオブジェクト指向プログラミングと数学を同時に学ぶようなコースも考えられるのではないかと考えている。

6．結び

情報教育の第一段階としてコンピュータの内部構造に触れずに単に便利な道具としてその

使い方を使ってみる、使ってみせることで教えるという「コンピュータリテラシー」は必要である。しかし、その次に来るものとして、コンピュータの内部構造について触れずにコンピュータをアルゴリズムの実行機械としてとらえ、アルゴリズムの教育と数学の教育を結びつけることにより相乗効果が期待できるのではないだろうか。