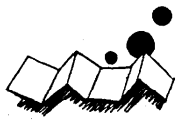
解　説



計算の複雑さについて†

野崎　昭弘††

この解説の目的は，計算の複雑さについての最近の体系化されつつある理論の根本のところを，計算機についてすでにかなりの予備知識をお持ちの方々に十分わかりやすい形で提示することである．したがって理論の位置づけと基本概念の解説が主眼であって，そのあとに多項式の計算と表の検索等についていくらか立ち入った説明が与えられているが，それらは我々の理論の「具体例」を示し，「感じ」(flavor)をつかんで頂くための「付録」のようなものである．限られた紙数で網羅的な解説を述べることはできないことをあらかじめお断りしておきたい．

1. 計算の複雑さ

1.1 計算の複雑さの理論とは

計算の複雑さの理論とは，簡単にいえば，ある計算を実行するのに「どれくらいの手間がかかるか」を研究する理論である．手間というのは所要時間，所要メモリ量，プログラムのステップ数，ハードウェアの素子（あるいは装置）数など，いろいろな観点から測られる．

この理論の実用上の意義は自明であろうが，ほかの理論との位置づけを見るために，「よいプログラムとはどんなものか」を考えてみよう．あるプログラムが「よいプログラムである」と見なされるためには，どのような条件を満足すべきであろうか．第一にそれは，正しいプログラムでなければならない．正しいというのは「目的に叶う」ということであり，また文法的にも正しく，さらに（数値的な問題であれば）十分に精度の高いものでなければならない．またそのプログラムは，許されるメモリ量および時間の枠内で，実行可能でなければならない．もちろん同じ時間の枠内でも，早く終わるものの方が望ましいことはいうまでもない．その上，データの形式や内容の変更や問題の一部変更など，環境の変化に追随して修正しやすいものであることが望ましい．

このような要求あるいは要望に対して，どのような理論が有用であるかを考えると，次のような対応表ができるであろう．

プログラムの正当性
- 目的に叶う……正当性理論を含むプログラム理論，およびソフトウェア工学
- 文法的に正しい……文法的誤りの検出と修正法の指示にかかわる，コンパイラ工学
- 計算精度が高い……数値解析学

プログラムの実行可能性
- 所要時間，所要メモリが少ない……**計算の複雑さの理論**
- 計算の原理的可能性……計算可能性の理論

プログラムの保守について
- 読みやすさ，修正の容易さ……ソフトウェア工学

このように見ると，計算の複雑さの理論が，よいプログラムを作るための理論として，ソフトウェア工学などに並ぶ基礎理論のひとつであることが容易に理解されるであろう．

本稿では，計算の複雑さの理論のうち，特に所要時間にかかわる部分をとりあげて紹介する．所要メモリ量についても随時触れるつもりであるが，プログラムの長さとかハードウェア的な複雑さについては論じない．それらについては，たとえば次の書物でお調べ頂きたい．

伊理正夫，野崎昭弘，野下浩平編著「計算の効率化とその限界」入門・現代の数学 [13]，日本評論社 (1980)

1.2 計算の複雑さの理論の出発点

我々の理論の出発点は，計算の複雑さ（特に，所要時間）をどのように測るか，というところにある．そこでひとつの具体例として，配列

$A(1), A(2), \cdots\cdots, A(N)$

に記録されている実数値を，小さい順に並べかえると

† On the Complexity of Computation by Akihiro NOZAKI (Natural Science Division, International Christian University).

†† 国際基督教大学理学科

```
FACOM 230 OS2/VS   FORTRAN.S                    V-03 L-66
                                                  30
SOURCE   LIST
0001            SUBROUTINE BUBBLE (A, II, JJ)
                C
                C  BUBBLE SORT,
                C  SORT ARRAY A INTO NONDECREASING ORDER,
                C  FROM A(II) TO A(JJ),
                C
0002            DIMENSION A(5010)
0003            DOUBLE PRECISION A, T
                C
0004            NA = JJ - II + 1
0005            JJ1 = NA - 1
                C
0006            DO 20 J = 1, JJ1
0007            JA = JJ - J
0008            IIT = 0
                C
0009            DO 15 I = II, JA
0010            IF(A(I) - A(I+1)) 15, 15, 10
0011         10 T = A(I)
0012            A(I) = A(I+1)
0013            A(I+1) = T
0014            IIT = 1
0015         15 CONTINUE
                C
0016            IF(IIT .EQ. 0) GO TO 25
0017         20 CONTINUE
                C
0018         25 RETURN
0019            END
```

図-1

表-1　順序づけの所要時間（I）

データ数 N	併合法	隣接交換法
500	0.306	5.371
1,500	1.105	48.456
2,500	2.002	134.542

単位：秒，一様乱数データ使用（王義孝氏による）
使用計算機：FACOM 230-45S

いう順序づけ（ソーティング）の問題について考えてみよう．入門書によくみられるのは隣接要素

$A(I)$, $A(I+1)$

の比較・（必要なら）交換をくり返す方法で，かりに**隣接交換法**と呼ぶことにする．細かいところまで含めるといろいろな形が考えられるが，ひとつのプログラム例を**図-1**に示した．

ほかによく知られているのは併合法(merging)であろう．ひとつのプログラム例を**図-2**に示した．

これらの方法の**所要時間**を比較する最も直接的な方法は，これらの**プログラム**を実際の計算機にかけて，所要時間を実測することである．すると**表-1**のような実測値が得られ，隣接交換法より併合法の方がずっと速い（らしい）ことがよくわかる．

このように絶対時間を実測することは，特定の計算機の上での能率を正確に知るためには必ず行わなければならない．しかしこのような実測値には次のような欠点がある．

（1） 2.002とか134.542などという数値は，その計算機については正しくても，ほかの計算機には必ずしもあてはまらない．また

$$\frac{134.542}{2.002}=67.2\cdots$$

という比率も，ほかの計算機でどの程度あてはまるのか，実測値だけからはわからない．

（2） 実測されなかった場合，たとえば $N=10$ とか $N=10000$ の場合にどうなるかは，実測値だけからは確実なことはいえない．

（3） 実測値には，基本となる方法の差だけでなく，プログラミング技法上の差（たとえばDO文を使うかIF 文を使うか，など）も関係する．

このような欠点を避けるためには，計算の途中で実行されるある基本ステップに注目して，そのステップが延べ何回実行されるかを考えてみるとよい．我々の問題については，たとえば要素どうしの比較

$IF(A(I).LE.A(J))$ ……

を基本ステップとすると，それぞれの方法での要素の比較回数はおよそ次のようになる．

隣接交換法では　約 $N^2/4$ 回，

併合法では　約 $N\log_2 N$ 回．

この回数は計算機によらず，またプログラム技法の細かいところにも依存しない．何々ミリセカンドというような絶対的な時間でないかわり，一般性のある目安

```
FACOM 230 OS2/VS    FORTRAN S                    V-03 L-66
                                                   27

SOURCE    LIST

0001            SUBROUTINE SMERGE (A, II, JJ)
                C
                C  MERGE SORT.
                C  SORT ARRAY A INTO NONDECREASING ORDER,
                C  FROM A(II) TO A(JJ), WHERE II IS 1.
                C
0002            DIMENSION A(5010)
0003            DOUBLE PRECISION A
                C
0004            IIS = 0
0005            IP = 1
0006            NUM = JJ - II + 1
0007            ICHK = 1
                C
0008         10 I = II
0009            J = JJ
0010            K = JJ
0011            L = JJ + JJ + 1
0012            IF(IIS .EQ. 0) GO TO 15
0013            I = JJ + 1
0014            J = JJ + JJ
0015            K = 0
0016            L = JJ + 1
0017         15 IE = 1
0018            IIQ = IP
0019            IR = IP
0020            ICK = IP + IP
0021            ICL = NUM / ICK
0022            IF(ICL .EQ. 0) IR = NUM - IIQ
0023            IZ = 0
                C
0024         20 IF(A(I) .GT. A(J)) GO TO 30
0025            K = K + IE
0026            A(K) = A(I)
0027            I = I + 1
0028            IIQ = IIQ - 1
0029            IF(IIQ .GT. 0) GO TO 20
                C
0030         25 K = K + IE
0031            IF(K .EQ. L) GO TO 45
0032            A(K) = A(J)
0033            J = J - 1
0034            IR = IR - 1
0035            IF(IR .GT. 0) GO TO 25
0036            GO TO 40
                C
0037         30 K = K + IE
0038            A(K) = A(J)
0039            J = J - 1
0040            IR = IR - 1
0041            IF(IR .GT. 0) GO TO 20

0042         35 K = K + IE
0043            IF(K .EQ. L) GO TO 45
0044            A(K) = A(I)
FACOM 230 OS2/VS    FORTRAN S        SMERGE      V-03 L-6
                                                   28

SOURCE    LIST

0045            I = I + 1
0046            IIQ = IIQ - 1
0047            IF(IIQ .GT. 0) GO TO 35
                C
0048         40 IZ = IZ + 1
0049            IIQ = IP
0050            IR = IP
0051            IE = (-1) * IE
0052            IK = K
0053            K = L
0054            L = IK
0055            IF(IZ - ICL) 20, 41, 44
0056         41 MREM = MOD(NUM,ICK)
0057            IF(MREM .EQ. 0) GO TO 44
0058            IR = MREM
0059            IF(IR .LE. IP) GO TO 42
0060            IR = IR - IP
0061            GO TO 20
0062         42 IIQ = MREM
0063            IF(ICHK .GT. 0) GO TO 35
0064            GO TO 25
0065         44 IE = (-1) * IE

                C
0066         45 IP = IP + IP
0067            IF(IP .GE. JJ) GO TO 50
0068            IIS = 1 - IIS
0069            ICHK = IE
0070            GO TO 10
0071         50 IF(IIS .NE. 0) GO TO 60
0072            DO 55 JKL = II, JJ
0073         55 A(JKL) = A(JKL+JJ)
0074         60 RETURN
0075            END
```

図-2

であるといえる．またNの大きさにもよらないので，$N=1,500$ とか $N=2,500$ の場合だけでなく，

Nが大きくなればなるほど，併合法が有利であるという全体的な傾向をある程度定量的に見ることもできる．

一般に，ある基本ステップに注目したときのその実行回数は，その基本ステップに関する**時間計算量** time complexity と呼ばれる．隣接交換法の時間計算量は約 $N^2/4$，併合法の時間計算量は約 $N\log_2 N$ ということになる．時間計算量に影響を及ぼすパラメータ（この例では N）は，問題の**サイズ**と呼ばれる．

ところで時間計算量は，サイズをきめてもぴったり確定するとは限らない．運・不運によって，多かれ少なかれバラツキが出るのがふつうである．併合法はバラツキが少ないが，隣接交換法はバラツキが大きいので，実は次のように大きな差が出る．

最も運がよい場合（データがすでに小さい順に並んでいる場合）の要素の比較回数：$N-1$ 回．

最も運が悪い場合（データが大きい順に並んでいる場合）の要素の比較回数：約 $N^2/2$ 回

前に述べた $N^2/4$ 回というのは，いろいろな場合が一様な頻度で起るとして計算した平均回数である．一般に基本ステップの平均実行回数を**平均時間計算量**といい，最悪の場合の実行回数を**最大時間計算量**という．計算センターなどでいろいろな場合を反復計算するときの目安としては平均時間計算量が役にたつ．またリアルタイム・オンライン処理などで，安全な目安がほしいときには最大時間計算量を使えばよい．最良の場合の基本ステップの実行回数は，**最小時間計算量**と呼ぶべきであろうが，応用上の意義は乏しい．

基本ステップとして何を選ぶかは，なかなか微妙な問題である．上記の問題については，要素どうしの比較のほかに，次のようなステップを加えてもよい．

要素の移動：$X=A(I)$，$A(I)=A(J)$ など．

添字に関する比較：$IF(I.LE.N)$……

添字の値の変更：$I=0$，$I=I+1$ など．

基本ステップをふやせば，当然それらの実行回数（の合計，時間計算量）もふえるが，現実の所要時間のよりよい目安になる．しかし今考えている例では，それらを全部あわせても平均時間計算量は次のとおりで，あまり大きな変化は起らない．

隣接交換法では　$c\cdot N^2$ 程度，

併合法では　$d\cdot N\log_2 N$ 程度．

ここで c, d はある定数係数である．要するに定数係数の差を無視するなら，要素の比較だけに注目した時間計算量に比べて，本質的な違いはないということである．

隣接交換法と併合法のように，能率に大差のある方法を比較するときには，ごく基本的なステップに注目するだけでよい．しかしヒープ・ソートとかクィック・ソートなど，高級な手法の比較のためには，要素の移動なども考えた方がよい．一方ポケット電卓による計算のための目安としては，「キーのどれかを押す」ことを基本ステップとする（つまりキーを押す回数をかぞえる）べきであろう．このように，基本ステップというものは問題に応じ，また状況に応じて選ばなければならないので，ここでは特に限定しないでおく．

なおメモリの使用量についても，同じように考える．すなわち，配列要素(および添字)の記憶に必要なメモリ量を，一時的に使用するのも含めて，適当な単位(ビット，バイト，ワード等)でかぞえるのである．そしてその単位数を，**領域計算量** space complexity という．たとえばワード単位でかぞえると，隣接交換法の領域計算量は $N+1$，併合法の領域計算量は約 $2N$ となる．計算の複雑さの理論は，時間計算量と領域計算量にかかわるものである限り，**計算量の理論**と呼んでもさしつかえない．本稿で扱うのは計算量の理論である．

1.3 計算量のオーダについて

計算量の理論では，便宜上，定数係数を無視した大ざっぱな議論をすることがある．前に現われた

$$cN^2,\quad dN\log_2 N$$

などはその一例である．これはずいぶん荒っぽいようであるが，それでもNがひじょうに大きいところでの比較（$N\to\infty$ のときの漸近的なふるまい）を知ることができる．実際，

$$c\neq 0,\quad d\neq 0$$

でありさえすれば，$N\to\infty$ のとき明らかに

$$dN\log_2 N/cN^2\to 0$$

となり，c, d の値は極限値 0 には関係しない．要するに $c, d(\neq 0)$ の大きさによらず，N が十分大きいところでは，

$$dN\log_2 N \ll cN^2$$

と考えてさしつかえない．

注意　Nが小さいところでは話は別である．実際 $N\leqq 10$ ぐらいだと，併合法より隣接交換法の方がたいてい速い．こういう細かいところは，絶対時間を実測してみないとわからない．

定数係数を無視した議論をするときには，たとえば cN^2 のことを $O(N^2)$ と書くことがある．たとえば隣接交換法の平均時間計算量を $T(N)$ であらわすとき，

$$T(N)\sim O(N^2)$$

と書けば，これはある定数 $c, d(\neq 0)$ について

$$\lim_{N\to\infty}\frac{T(N)}{cN^2}=d$$

であることを意味している．このようなとき関数 $T(N)$ の**オーダ**は N^2 である，という．併合法の時間計算量を $S(N)$ とすると，

$$S(N)\sim O(N\log N),$$

すなわち $S(N)$ のオーダは $N\log N$ である．

注意 対数 log の底 $a(>1)$ は，どのようにとっても定数倍しか変わらない（同じオーダ）なので，書かなくてもよい．

なお最近は次のような略記法もよく使われる．

（1） $f(n)=O(g(n))$

意味：$f(n)$ はオーダ $g(n)$ の上界をもつ．つまりある定数 α について，十分大きなすべての n に対して

$$f(n)\leqq\alpha g(n).$$

（2） $f(n)=\Omega(g(n))$

意味：$f(n)$ はオーダ $g(n)$ の下界をもつ．つまりある定数 $\beta(>0)$ について，十分大きなすべての n に対して

$$f(n)\geqq\beta g(n).$$

（3） $f(n)=\Theta(g(n))$

意味：$f(n)$ のオーダは $g(n)$ である．つまりある定数 $\alpha, \beta(>0)$ について，十分大きなすべての n に対して

$$\beta g(n)\leqq f(n)\leqq\alpha g(n)$$

注意 やかましくいうと，これは前に述べたオーダの定義と厳密には一致しない．しかし実用上必要となる多くの場合一致するし，書物によっても多少異なるので，このままにしておく．

次に，オーダの評価によく使われる定理（の筆者による一変形）をひとつ紹介しておこう．

定理 正整数 n に対して定義されている関数 $T(n)$ を考える．p, q および a, b を非負実数とし，特に $q>1$ と仮定する．ある正整数 N が存在して，すべての

$$n\geqq N$$

に対して

$$T(n)\leqq an+b+pT(\lceil n/q\rceil)$$

が成りたつならば，T は次の不等式を満足する．

（ア） $p<q$ の場合：$T(n)\leqq cn$

（イ） $p=q$ の場合：$T(n)\leqq(a+\varepsilon)n\log_q n$

（ウ） $p>q$ の場合：$T(n)\leqq cn^{\log_q p}$

ここで c, ε は n と無関係な定数である．

注意 （イ）は ε を十分大きくとれば，すべての n に対して成りたつ．また $\varepsilon>0$ でありさえすれば，十分大きい n に対して成りたつ．

［略証］

（1） $n_0=n$, $n_i=\lceil n_{i-1}/q\rceil$ とおくと，容易にたしかめられるように

$$\begin{aligned}n_i&<n/q^i+1/q^{i-1}+\cdots+1/q+1\\&<n/q^i+q/(q-1)\end{aligned}$$

（2） $M=\mathrm{Max}\{N, 1+\lceil q/(q-1)\rceil\}$, $T=\lceil\log_q n\rceil$ とおくと，

$$n_T<n/q^T+q/(q-1)\leqq M.$$

したがって

$$n_i\leqq M$$

をみたす最小の i を t とおくと，$t\leqq T$.

（3） $K=\mathrm{Max}\{T(1), \cdots, T(M)\}$ とおく．T についての仮定から，$n\geqq M$ のとき

$$\begin{aligned}T(n)&\leqq an_0+b+p(an_1+b)+p^2(an_2+b)+\cdots\\&\quad+p^{t-1}(an_{t-1}+b+p\cdot K)\\&<an\sum_{i=0}^{t-1}(p/q)^i+c\sum_{i=0}^{t}p^i\\&\leqq an\sum_{i=0}^{T-1}(p/q)^i+c\sum_{i=0}^{T}p^i\quad\cdots\cdots\cdots\cdots\cdots\cdots(*)\end{aligned}$$

ここで $c=\mathrm{Max}\{b+q/(q-1), K\}$.

（4） もし $p<q$ ならば

$$\begin{aligned}(*)&\leqq an\sum(p/q)^i+cqn\sum p^i/q^T\\&<(a+cq)n\sum_{i=0}^{\infty}(p/q)^i=\Theta(n)\end{aligned}$$

（5） もし $p=q$ ならば

$$\begin{aligned}(*)&\leqq anT+cp^{T+1}/(p-1)\\&=\Theta(n\log_q n)\end{aligned}$$

（6） もし $p>q$ ならば

$$\begin{aligned}(*)&<an(p/q)^T/(p/q-1)+cp^{T+1}/(p-1)\\&=\Theta(p^T)=\Theta(n^{\log_q p})\end{aligned}$$

2. 多項式の計算

はじめにかなり詳しい性質が知られている多項式の計算法について述べよう．基本ステップとしては次の4種類を考える．

$u\leftarrow v+w$

$u\leftarrow v-w$

$u\leftarrow v\times w$

$u\leftarrow v$

ここで矢印←は右辺の値を左辺の変数に代入することをあらわす．uは変数名，vおよびwは変数または定数で，定数としては整数，小数だけでなく分数

(a/b)

の形をも許すことにする（a, b はもちろん定数である）．

2.1 x のべキ乗の計算

x の n 乗 x^n を乗算

$u \leftarrow v \times w$

のくり返しによって求めるとき，必要かつ十分な乗算回数を $M(n)$ であらわすことにしよう．たとえば x^4 は，次の2回の乗算によって求められる．

$u \leftarrow x \times x$,

$v \leftarrow u \times u$.

すぐわかるのは，次のような性質である．

（1） 一般に

$M(n) \geqq \lceil \log_2 n \rceil$

ここで記号「 」は端数切上げをあらわす．

（2） 特に $n=2^k$ の場合，

$M(n)=\log_2 n$.

（3） 一般に

$M(n) \leqq 2\lfloor \log_2 n \rfloor$

ここで記号⌊ ⌋は端数切捨てをあらわす．これは次の関係から導びかれる．

(3′) $M(n) \leqq M(\lfloor n/2 \rfloor)+2$

[証明] $x^{2m}=x^m \times x^m$,

$x^{2m+1}=x^m \times x^m \times x$

である．ゆえに $n=2m$ ならば

$M(n) \leqq M(m)+1$,

また $n=2m+1$ ならば

$M(n) \leqq M(m)+2$.

いずれにしても

$M(n) \leqq M(\lfloor n/2 \rfloor)+2$.

以上のことから，次の結果が得られる．

$M(n)=\Theta(\log n)$

実はさらに詳しく，次の事実が知られている．

定理 (Brauer)

$$\lim_{N\to\infty} \frac{M(n)}{\log_2 n}=1$$

定理 (Erdös) はほとんどすべての n について，

$M(n) \sim \log_2 n+\log_2 n/\log_2 \log_2 n$

2.2 多項式の時間計算量の上界

次に1変数の n 次多項式

$p(x)=a_n x^n+\cdots+a_1 x+a_0$

の計算法について考えよう．よく知られているのは次のホーナーの方法である．

$(\cdots(a_n x+a_{n-1})\times x+\cdots+a_1)\times x+a_0$

この方法によれば任意の n 次多項式が，n 回の乗算と n 回の加算で求められる．

ではより少ない乗算で任意の n 次式の値を求めることはできないだろうか．それはある意味では不可能であるが，式変形を許すなら可能である．

[Todd の方法] 任意の4次式 ($a_4 \neq 0$) は次のように変形できる．

$a_4x^4+\cdots+a_1x+a_0$

$=a_4(x(x+\alpha)+\beta)(x(x+\alpha)+x+r)+\delta$

このように変形しておけば，もとの4次式の値が次のように3回の乗算で求められる．(加算は5回になる)．

$u \leftarrow x+\alpha$

$v \leftarrow x \times u$

$w \leftarrow v+\beta$

$p \leftarrow v+x$

$q \leftarrow p+r$

$w \leftarrow w \times q$

$w \leftarrow a_4 \times w$

$w \leftarrow w+\delta$

α, β, r, δ を求めるのはこれより複雑な計算を要する（しかし未定係数法で，加減乗除だけで求まる）が，同じ式を何回もくり返し計算したいときは，式変形は1回だけやっておけばよいので，無視できるとしてよいであろう．

このような式変形は，一般の n 次式にも適用できる．

[Knuth の方法] n 次多項式 $p(x)$ の偶数次の項の和は，$y=x^2$ の多項式と考えることができる．これを $Q(y)$ とおく．また奇数次の項だけの和も，xをひとつくくり出せば，残りは y についての多項式となるから，それを $R(y)$ とおく．結局

$p(x)=Q(y)+x \cdot R(y)$

ということである．この R が

$R(y)=(y-\alpha_1)\cdots(y-\alpha_t)$

と因数分解できたと仮定して，次のような式変形を行う．

$Q_0(y)=Q(y)+xR(y)$

$Q_{i-1}(y)=(y-\alpha_i)Q_i(y)+b_i$

要するに，Q_{i-1} を $(y-\alpha_i)$ で割った商を Q_i，余りを b_i とするのである．$xR(y)$ の側はいつでも割り切れるので，余り b_i はいつでも定数になる．また $Q_t(y)$

はyについてのある多項式にxを加えたものになる．多項式Q_iの係数と各b_iの値を求めておけば，与えられたxに対する$p(x)$の値が次のように計算できる．

（1） $y \leftarrow x \times x$

（2） $Q_i(y)$の値を計算する（q_iとおく）．

（3） $q_{i-1}=(y-\alpha_i)\cdot q_i+b_i$

を$i=s, s-1, \cdots, 1$の順で計算する．

これらの計算に，準備の手間を無視すれば，

$\lfloor n/2 \rfloor+2$回の乗算とn回の加算

でできる（加算を1回ふやせば，乗算を$\lfloor (n+3)/2 \rfloor$回にすることもできる）．つまり，乗算を約半分に減らすことができるのである．

この方法の欠点は，$R(y)$の因数分解に代数方程式を解かなければならないこと（手間はともかく，精度の問題が出てくる），各α_iが複素数になるかもしれないことである．後者については，Eveが次のことを示した．

定理 適当に定数cを選んで$x=z-c$という変数変換を施してから，$y=z^2$に対して

$$p(z-c)=Q(y)+q\cdot R(y)$$

とおけば，$R(y)=0$の根はすべて実根になる．

また前者については，Rabinと Winogradによる次の結果がある．

定理 次数が$m(m+1)+1$以下である多項式に，次の条件をみたす$m(m+1)+2$個のパラメータで表現できる．

（1） その多項式は，パラメータおよび変数xの値から，$m(m+1)/2+O(m)$回の乗算で計算できる．

（2） どのパラメータも，もとの多項式の係数の有理式であらわせる．

2.3 多項式の時間計算量の下界

多項式の乗算回数の下界を示すには，次の定理が役にたつ．

定義 実係数多変数多項式を成分とするp次元列ベクトル$e_1, \cdots, e_t$がR独立であるとは，次の条件が成り立つことをいう．

もしある実数$\lambda_1, \cdots, \lambda_t$に対して

$$\lambda_1 e_1+\cdots+\lambda_t e_t \in R^r$$

ならば実は

$$\lambda_1=\cdots=\lambda_t=0.$$

定理（Winograd） 変数$u_1, \cdots, u_n$に関する多項式を要素とするp行q列の行列Aがs個のR独立な列を含むならば，次の行列ベクトル積の計算には少なくともs回の乗算が必要である．

$$A\cdot[v_1, \cdots, v_q]^T$$

ここで$v_1, \cdots, v_q$は$u_1, \cdots, u_n$とは異なる任意の変数とする．

［応用］ $A=[x^n x^{n-1}\cdots x1]$の最初のn列はR独立である．したがって積

$$[x^n x^{n-1}\cdots x1]\cdot[a_n a_{n-1}\cdots a_1 a_0]^T$$

の計算には少なくともn回の乗算が必要である．

これでn次多項式の計算には，n回の乗算が必要であること（したがってホーナーの方法が最良であること）がたしかめられた．

式変形の手間を除いて乗算回数を評価することもできる．

定理 任意のn次多項式の値を適当な式変形（あるいは一般に，xの値には関係しない予備計算）ののちにm回の乗算で求める一般的な方法があるとしたら，

$$m \geqq \lfloor n/2 \rfloor+1.$$

この結果と比較すると Knuthの方法はほとんど最適といってよいことがわかる．

ところで実用上は，多項式$p(x)$の値をある範囲のx，たとえば$c \leqq x \leqq d (c<d)$に対して，ある許容誤差$\varepsilon(>0)$以内で計算すればよいことが多い．その場合，許容誤差を越えないように係数をわずかに修正することによって，計算量を大幅におさえられないものだろうか．この間に対する答は半ばイエス，半ばノーであって，次の結果が知られている．

定理（Paterson, Stockmeyer）係数がすべて有理数の多項式の値は，約$\sqrt{2n}$回の乗算で求めることができる．

しかしそのやり方では，加減算の回数がどこまでふえるかわからない．そこで加減算の回数にある上限Nを定めると，次のことが証明できる．

定理（Nozaki） εがある程度以上小さいと，N回以下の加減算と$\lfloor n/2 \rfloor$回以下の乗算では近似値も求められない多項式が無数に存在する．

要するに一般的・実用的な計算法としては，およそ$n/2$回の乗算が必要十分と考えてよいわけである．

3. 単純な索表と順序づけ

ここでは配列

$$X(1), \cdots, X(N)$$

等に対する検索や順序づけ（ソーティング）の複雑さを概観する．基本ステップとしては前章で考えた4種類に，次の4種類をつけ加える．

$$w \leftarrow u/v$$

if $u=v$　**go to** L,
if $u\leqq v$　**go to** L',
go to L''.

L, L' および L'' はあるステップにつけられたラベルをあらわす（意味は明らかと思うから，説明は省略する）．

3.1 索　表

逐次検索法　配列 $X(1), \cdots, X(N)$ とデータ A とが与えられたとして，

$A=X(I)$

をみたす最小の I をみつけるという問題を考えてみよう．すぐ思いつくのは次のような方法であろう．

$I\leftarrow 1$
L_1: **if** $A=X(I)$　**go to** L_2
$I\leftarrow I+1$
if $I\leqq N$　**go to** L_1
go to L_3

ここで L_2 は「I がみつかった」ときのゆくさき，L_3 は「A が X の中になかった」ときのゆくさきである．

この手順の時間計算量は次のように評価できる．

最良の場合（$A=X(1)$ のとき）：2
最悪の場合（A が X の中になかったとき）：
$3N+2$.

$A=X(1), \cdots, A=X(N)$ の各場合，およびどれでもない場合が等確率で起るとしたら，平均時間計算量は約 $1.5N$ になる．これは次のようにプログラムを修正することによって，大幅に改良できる．

$X(N+1)\leftarrow A$
$I\leftarrow 1$
L_1: **if** $A=X(I)$　**go to** L
$I\leftarrow I+1$
go to L_1
L: **if** $I\leqq N$　**go to** L_2
go to L_3

このようにすると $I\leqq N$ の判定がループの外に出るので，平均時間計算量は約 $1.0N$ になる．3割以上のスピード・アップということである．

このように $X(1), X(2), \cdots$ をその順に検査する方法は**逐次検索法**と呼ばれる．この方法はよく現われる項目が前の方におかれているときにはもっと速くなりうる．一般に

$A=X(I)$

となる確率を p_I, A が X の中にない確率を p_{N+1} とし，p_I が

$$p_I \fallingdotseq \frac{a}{I^n}\quad (a \text{ は定数})$$

という形で近似的にあらわされるとすると，平均時間計算量は n の値によって次のように変化する．

$n=1$ のとき……約 $2aN$
$n=2$ のとき……約 $2a\log_2 N$
$n=3$ のとき……約 $2a$

英単語を使用頻度の順に並べると，Zipf の法則によれば $n=1$, $a=0.1$ となる．したがって平均時間計算量はおよそ $0.2N$ となり，かなりのスピード・アップが期待できる．

2分検索法　各 $X(I)$ が実数値で，しかも

$X(1)\leqq X(2)\leqq\cdots\cdots\leqq X(N)$

のように順序づけられている場合には，次の方法が使える．

$I\leftarrow 1$
$J\leftarrow N$
L_1: $K\leftarrow I+J$
$K\leftarrow K/2$（端数切捨てとする）
if $A=X(K)$ **go to** L_2
if $A\leqq X(K)$ **go to** L_4
$I\leftarrow K+1$
if $I\leqq J$ **go to** L_1
go to L_3
L_4: $J\leftarrow K-1$
if $I\leqq J$ **go to** L_1
go to L_3

L_2, L_3 の意味は前の通りである．

この方法によれば，最大時間計算量は約 $6\log_2 N$，平均時間計算量も（p_I が一様と仮定して）ほとんど変わらない．たとえば $N=1000$ のとき

$0.2N=200$,
$6\log_2 N \fallingdotseq 60$

であるから，これはひじょうに速い方法である．

ハッシュ法　$X(I)$ がある限られた範囲の相異なる整数値である場合には，次のようにして補助の配列 Y を作るとよい．便宜上

$1\leqq X(I)\leqq M$

と仮定する．

（1）　まずすべての $Y(K)(1\leqq K\leqq M)$ を 0 にする．
（2）　次に各 $I(1\leqq I\leqq N)$ に対し，次の操作を行う：$Y(X(I))\leftarrow I$

すると $A=X(I)$ をみたす I は，次のように簡単に求められる．

$I \leftarrow Y(A)$
if $I=0$ **go to** L_3
go to L_2

このようにすればNに関係なく3ステップ以内で検索が終了する．これを**分配法**と呼ぶ．

$X(I)$ の値の範囲Dが十分狭くないときは，Dを

$$E=\{1, 2, \cdots, M\}$$

に変換する関数 $\varphi: D \rightarrow E$ を使うとよい（$M>N$ とする）．そして次のような配列Zを作る．

（1） すべての $Z(K)$ $(1 \leqq K \leqq M)$ にある値αを入れておく．ここでαはDに属さない任意の要素である．

（2） 各 $I(1 \leqq I \leqq N)$ に対して，次の操作を行う．

$K \leftarrow \varphi(X(I))$
L: **if** $Z(K)=\alpha$ **go to** R
$K \leftarrow K+1$
if $K \leqq M$ **go to** L
$K \leftarrow 1$
go to L
R: $Z(K)=X(I)$

このようにしておくと，$A=Z(K)$ をみたすKは，次のように求められる．

$K=\varphi(A)$
P: **if** $A=Z(K)$ **go to** L_2
if $Z(K)=\alpha$ **go to** L_3
$K \leftarrow K+1$
if $K \leqq M$ **go to** P
$L \leftarrow 1$
go to P

配列Zを**ハッシュ表**，関数φを**ハッシュ関数**という．この方法の平均時間計算量は，比較的ゆるい仮定のもとで，$O(1)$（つまりNに無関係な定数）であることが証明できる．しかし“占有率”$\xi=N/M$ が1に近いと能率は著しく落ちるので，$\xi \leqq 0.7$ ぐらいが安全といわれている．詳しい分析が次の文献に含まれているので，ご覧頂きたい．

D. E. Knuth, the Art of the Computer Programming, Vol. III, “Sorting and Searching”, Addison-Wesley (1973)

3.2 順序づけ

順序づけについては前にも触れているし，解説書も多いのでここでは要点だけ述べることにする（詳細は，たとえば Knuth の上記の文献がよい参考になる）．

順序づけは，対象となるデータが主記憶装置内のひとつの配列である場合（**内部ソート**）と，外部記録装置に記録されている場合（**外部ソート**）とにわけられる．

表-2 順序づけの所要時間（II）

データの個数 N	クィック法*)	併合法	ヒープ法
500	0.189	0.306	0.349
1,500	0.655	1.105	1.243
2,500	1.158	2.002	2.194

単位：秒，一様乱数データ使用（王義孝氏による）
使用計算機：FACOM 230-45S
*) 王義孝氏の改良によるもの

内部ソート 対象となるデータの個数をNとする．

（1） **隣接交換法** 最大時間計算量，平均時間計算量ともに $\theta(N^2)$ で，Nが大きいときは非常に遅く，実用的でない．しかし $N \leqq 10$ ぐらいだとかえって速く，領域計算量が $N+1$ ですむのが長所といえる．

（2） **併合法** 最大，平均時間計算量がともに $\theta(N \log N)$ であるが，領域計算量は$2N$かかる．

（3） **ヒープ法**（整列2分木法，tree sort） 最大，平均時間計算量がともに $\theta(N \log N)$ で，領域計算量も $N+1$ ですむ．しかし所要時間を実測してみると，併合法にやや劣る（**表-2 参照**）．

（4） **クィック法** (quicksort) 平均時間計算量は $\theta(N \log N)$ で，所要時間を実測してみるとヒープ法のおよそ半分であり，一般的な方法の中では最も速い．平均領域計算量は $N+C \cdot \log_2 N$（Cはある定数）で，少し工夫すると最大領域計算量を $N+2\log_2 N$ におさえることもできる．欠点は最大時間計算量が $\theta(N^2)$ に達することであるが，これもヒープ法と組み合わせることによって $\theta(N \log N)$ におさえることができる．

（5） **分配法** (distribution sort) これはデータの値がある範囲の整数値の場合に有効な方法である．一例として

$$0 \leqq L(I) \leqq M \quad (1 \leqq I \leqq N)$$

をみたす整数配列Lの順序づけを考えてみよう（入試の成績処理などがよい実例である）．

まず次のようにして配列 LRANK を作る．
（以下便宜上 FORTRAN で書く）．

```
      DO 1 J=0,M
    1 LRANK (J+1)=0
      DO 2 I=N,1,-1
        J=L(I)+1
        L(I)=LRANK (J)
    2 LRANK (J)=I
```

こういう表を作っておけば，$L(I)$ の値の大きい順に，その値と I の値，それに順位を印刷することは次のような手順でできる（順位 IRANK, I, $L(I)$ の順に印刷する）．

```
    IRANK=L
    DO 3 J=M,0,-1
        IRGAP=0
        L1=J-1
        I=LRANK (J+1)
      5 IF (I. EQ. 0) GO TO 3
        IRGAP=IRGAP+1
        WRITE (6,600) IRANK, I, L 1
(FORMAT 文 略)
        I=L(I)
        GO TO 5
      3 IRANK=IRANK+IRGAP
```

この方法によれば，最大時間計算量は $\Theta(N+M)$，領域計算量は $N+M+8$ ですむ．M が大きくなければ，これは抜群に速い方法である（たとえば 500 点満点の入学試験を 4,000 人が受験するなら，$N=4,000$, $M=500$ である）．

外部ソート 磁気テープあるいは磁気ディスクを使用してソーティングを行うには，併合法がほとんど唯一の方法である．その基本は次のような二進併合法 (two-way merge) である．

磁気テープ装置 A, B, C, D の 4 台を使用するものとし，最初 N 個のデータが全部 A に記録されているものとする．

(準備) A の中のテープの半分を C に，残り半分を D に移す．それぞれ $N/2$ 個のデータは，それぞれ「すでに順序がつけられている，長さ 1 の列」とみなされる．

(併合 1) C, D の中の列をひとつずつ併合して，ひとつの新しい順序列を作る．新しい順序列は，A と B に交互に 1 列ずつ書きこまれる．

(併合 2) A, B の中の列をひとつずつ併合して，ひとつの新しい順序列を作る．新しい順序列は，C と D に交互に 1 列ずつ書きこまれる．

(併合 1) と (併合 2) を交互にくり返せば，$\lceil \log_2 N \rceil$ 回の反復の後に A または C に完全な順序列が得られる．

この方法には次のような改良法が考えられる．

(1) 自然併合法 準備段階で最初のデータ．たとえば

3, 1, 4, 1, 5, 9

を，

3/1/4/1/5/9

のように区切って 6 個の「長さ 1 の順序列」と考えるかわりに，自然にできている順序列を利用して，

3/1, 4/1, 5, 9

という 3 個の順序列として扱う．全体が M 個の順序列になるなら，C と D に $M/2$ 本ずつの列を分配すればよい．なおこの段階で，なるべくたくさんのデータを内部ソートによって順序づけておくのもよい．

(2) 多重併合法 $2k$ 台の磁気テープ装置が利用できる場合には，最初にデータを k 台のテープを分配しておき，それから「k 本の順序列を 1 本に併合する」操作をくり返すとよい．併合の反復回数は $\lceil \log_k N \rceil$ ですむから，ずっと速くなる．

(3) 多層併合法 k 台の磁気テープ装置が利用できる場合，最初にデータを $k-1$ 台のテープに分配しておき，それから「$k-1$ 本の順序列を 1 本に併合する」操作をくり返す．最初に分配するデータ数を適正にしておくと，ある 1 台のデータがなくなったときに (ほとんど) いつもほかの $k-1$ 台にはデータが残っていて，能率のよい併合を続けることができる．

4. おわりに

いくつか基本的な文献はすでに挙げたが，一般的な入門書を示しておこう．

Aho, A. V., Hopcroft, J. E. and Ullman, J. D.: The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974 (野崎，野下他訳，アルゴリズムの設計と解析 I, II，サイエンス社)．

Aho, A. V.: Currents in the Theory of Computing, Prentice-Hall, 1973 (守屋悦朗訳，最近の計算理論，近代科学社)．

Wirth, N.: Algorithms+Data Structure=Programs, Prentice-Hall, 19 (片山卓也訳，アルゴリズム＋データ構造＝プログラム，日本コンピュータ協会)．

(昭和 55 年 7 月 24 日受付)