アルゴリズム 43−10
（1995. 1. 23）

対象の代数構造を重視したアルゴリズム記述法

飯田卓郎, 岩澤京子, 萩原洋一, 中森眞理雄
東京農工大学 工学部 情報工学大講座

概要

リストに関して2項演算とそれに基づく代数系を定義することにより, リストに関するアルゴリズムを一般的に記述し, その特殊な場合として多くのアルゴリズムを導くことができることを示す. 取り上げるアルゴリズムは, n個の数値の和, 積, 最大値, 最小値, 整列, 小さい方からk番目の要素などである. これにより, アルゴリズムを, 曖昧さがなく, 短く, 構造上の特徴を捉えやすく記述することが可能となる. 本研究はアルゴリズムのデータベースに応用することができる.

Algebraic Structure Based Description of Algorithms

Takuro IIDA, Kyoko IWASAWA
Yoichi HAGIWARA, and Mario NAKAMORI
Department of Computer Science
Tokyo University of Agriculture and Technology

Abstract

By defining binary operations and algebraic structures on lists, we describe a general algorithm on lists. We obtain various algorithms on lists as special cases of the general algorithm. Examples are sum, product, maximum value, minimum valiue, sort (order), and the kth smallest valuc of n numbers. Such a way of description makes it easy to describe algorithms without ambiguity, in a precise way. It also makes it easy to characterize thc structure of algorithms. Our research is applicable to the design of a database system on algorithms.

1 はじめに

近年, アルゴリズムに関する入門書, 研究書, 事典, ハンドブック類が多数出版されている. アルゴリズムはプログラムを抽象化した概念であり, その記述に標準的な方法が定まっているわけではない. 上記の書物類でも, 個々のアルゴリズムに応じて, さまざまな記述のしかたがされている.

一般に, アルゴリズムを記述する方法としては, 次のものが考えられる.

(1) プログラム言語による方法

実際, プログラム言語 ALGOL60 は ALGOrithmic Language として提案されたものであった [1]. また, プログラム言語 PASCAL は教育用言語として提案されたものであり [2], アルゴリズムの書物に広く用いられている.

プログラム言語を用いる方法は, 厳密さの点で優れているが, 記述が長くなる傾向がある. また, データ構造が特定のものに限定され, 自由度が少ない (ただし, この問題は, 抽象データ型を用いれば解決される).

(2) 自然言語による方法

プログラムそのものより抽象的に記述できるので, 著者の意図を表現しやすいという利点があるが, 自然言語の本質として, 曖昧さが避けられないのが欠点である.

(3) プログラム言語と自然言語を併用する方法

上記 (1), (2) の長所を取り入れたように見えるが, 使い方によっては, 両者の欠点を兼ね備えたものにもなる.

制御構造は既存のプログラム言語のものを用い, 問題固有の処理は自然言語や数式などを用いる, というのが多くの書物におけるアルゴリズムの記述方法である (例えば, [3] の Pidgin ALGOL).

他に, 論理回路レベルで記述する方法もあるが, そのようなレベルの問題は本論文では, 一応, 考察の対象外とする.

本論文では,

アルゴリズムの標準的な記述形式
アルゴリズムの抽象的な記述形式
アルゴリズムの特徴を捉えるのに適した記述形式
アルゴリズムを短く表現できる記述形式
曖昧さのないアルゴリズム記述形式

を目標として, 記述を試みる. 具体的には, リストに関するいくつかのアルゴリズムを取り上げ, リスト同士の2項演算に基づく代数系を定義し, 種々のアルゴリズムを共通の代数系における一般的・抽象的アルゴリズムの特殊な場合として導く.

本研究は, アルゴリズムの構造上の特徴をキーとする検索を想定したアルゴリズムベース (アルゴリズムのデータベース) に応用できる.

2 リストに関するアルゴリズム記述の一般形

2.1 諸定義

以下では, **リスト** (list) とは数値や文字などの有限列と定義する. 書物によっては, "リスト" を "リンクされたリスト (linked list)" の意味に使う場合があるが, 本論文におけるリストとは別物である. リストは, 一般に, $a_1, a_2, \ldots, a_n$ と記し, 各 $a_i (i = 1, 2, \ldots, n)$ を**要素** (element) とよぶ.

本論文では, 要素1個だけのリストとその要素を同一視することにする. すなわち, 要素a_1だけから成るリストLに対して$L = a_1$と書く. リスト$L = a_1, a_2, \ldots, a_n$の**長さ** (length) とは要素の個数$n$のことであり, $|L|$と書く.

リスト同士の**2項演算** (binary operation) を一般に$\bigcirc$と書くことにする. 二つのリストL, L'に対して2項演算$\bigcirc$の結果$L \bigcirc L'$はリストである. 本論文では, いかなるリストL, L'に対しても$|L \bigcirc L'| \leq |L| + |L'|$と仮定する.

いかなるリストL, L', L''に対しても,

$$(L \bigcirc L') \bigcirc L'' = L \bigcirc (L' \bigcirc L'') \tag{1}$$

が成り立つとき, 2項演算$\bigcirc$は**結合的** (associative) であるという. 2項演算$\bigcirc$が**結合的** (associative) であるとき, $L \bigcirc L' \bigcirc \cdots L^{(n)}$のような記述は一意的な意味をもつ. 本論文では, 結合的な2項演算だけを対象とする.

いかなるリストL, L'に対しても,

$$L \bigcirc L' = L' \bigcirc L \tag{2}$$

が成り立つとき, 2項演算$\bigcirc$は**可換** (commutative) であるという. 本論文に現れる2項演算はすべて可換であるが, 本論文の記述において可換性は本質ではない.

リスト同士の演算$L \bigcirc L'$に伴う**時間複雑度** (time complexity), **空間複雑度** (space complexity), **誤差** (error) などを**コスト** (cost) と総称する.

2項演算$\bigcirc$は結合的であるが, コストは必ずしも結合的であるとは限らない.

2.2 リストに関する縦型アルゴリズム

リスト$a_1, a_2, \ldots, a_n$に対して2項演算$\bigcirc$を$n-1$回行った結果$a_1 \bigcirc a_2 \bigcirc \cdots \bigcirc a_n$を求めるアルゴリズムを考える.

おそらく, もっとも素朴なアルゴリズムは, 次の計算式によるものであろう.

$$(\cdots(a_1 \bigcirc a_2) \bigcirc \cdots) \bigcirc a_n \tag{3}$$

この式の構文木は図1のとおりである.

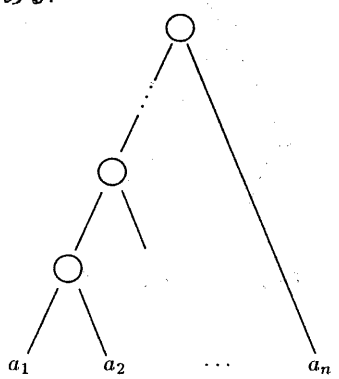

図1 縦型アルゴリズム

'ALGOL風に書くならば,

$S :=$ **nil** ;

for $i := 1$ **to** n **do** $S := S \bigcirc a[i]$

ただし, 配列aの第i要素を$a[i]$とし, $a[i]$の中身はa_iとする. また, **nil**は演算$\bigcirc$に関する単位元(unit element)である.

(2)においてはn回の演算$\bigcirc$が行われる. 次の手順

$S := a[1]$;

for $i := 2$ **to** n **do** $S := S \bigcirc a[i]$

によれば, 演算$\bigcirc$の回数は$n-1$となる. ただし, $n = 0$のときSには$a[1]$の代わりに**nil**が入るものとする. また, $n < 2$のとき,

for $i := 2$ **to** n **do**

は何もしないものとする.

次のように再帰関数fを用いて縦型アルゴリズムを記述することも可能である.

$$f(La) = \textbf{ if } L = \textbf{empty } \textbf{ then } a \textbf{ else } f(L) \bigcirc a \tag{4}$$

2.3 リストに関する横型アルゴリズム

リスト$a_1, a_2, \ldots, a_n$に対して$a_1 \bigcirc a_2 \bigcirc \cdots \bigcirc a_n$を求めるアルゴリズムであるが, 前節とは計算の順序が異なる.

$$a_1 \bigcirc \cdots \bigcirc a_{2n} = (a_1 \bigcirc \cdots \bigcirc a_n) \bigcirc (a_{n+1} \bigcirc \cdots \bigcirc a_{2n}) \tag{5}$$

すなわち,

$$(\cdots((a_1 \bigcirc a_2) \bigcirc (a_3 \bigcirc a_4)) \bigcirc ((a_5 \bigcirc a_6) \bigcirc (a_7 \bigcirc a_8))\cdots)$$

のように計算するアルゴリズムである. この式の構文木は図2のとおりである.

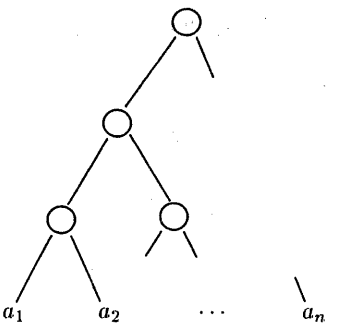

図2 横型アルゴリズム

ALGOL風に記述するならば,

$imax := n \div 2$;

while $imax > 1$ **do**

begin

for $i := 1$ **to** $imax$ **do**

$a[i] := a[2*i-1] \bigcirc a[2*i]$;
$imax := imax \div 2$
end

ただし, $\div$は変数間の除算(端数は切り捨て)である. また, 存在しない要素を参照したときは**nil** ($\bigcirc$に関する単位元)を返すことは, 前述のとおりである. 次のように再帰関数gを用いて横型アルゴリズムを記述することも可能である.

$$g(L) = \textbf{ if } |L| = 1 \textbf{ then } L \text{ bf else } g(left(L)) \bigcirc g(right(L)) \tag{6}$$

ただし, $left(L)$はリストLの左半分, $right(L)$はリストLの右半分である(リストLの長さ$|L|$が奇数のときは中央の要素は$left(L)$に含まれ, $right(L)$には含まれないものとする).

3 リストに関するアルゴリズムの記述例

3.1 数値の和, 積

2項演算$\bigcirc$が加算$+$の場合は,

$$a_1 \bigcirc a_2 \bigcirc \cdots \bigcirc a_n = a_1 + a_2 + \cdots + a_n = \sum_{i=1}^{n} a_i \tag{7}$$

となる. 単位元は0である.

縦型アルゴリズムは, 式では,

$$(\cdots(a_1 + a_2) + \cdots) + a_n \tag{8}$$

ALGOL風に記述すると,

$S := 0$;
for $i := 1$ **to** n **do** $S := S + a[i]$

再帰関数を用いて記述すると,

$$f(La) = \textbf{ if } L = \textbf{empty then } a \textbf{ else } f(L) + a \tag{9}$$

2項演算$\bigcirc$が乗算$\times$の場合は,

$$a_1 \bigcirc a_2 \bigcirc \cdots \bigcirc a_n = a_1 \times a_2 \times \cdots \times a_n = \prod_{i=1}^{n} a_i \tag{10}$$

となる. 単位元は1である. 具体的な記述例は上記と同様である.

縦型アルゴリズムを**2.2**の再帰関数fを用いて定義した場合, コストを$c(f, L)$と書くことにすると, 時間複雑度は, $c_T(f, La) = c_T(f, L) + 1$. これから, 一般に, $c_T(f, L) = O(|L|)$が導かれる. 空間複雑度の場合, $c_S(f, La) = 1$. これから, 一般に, $c_T(f, L) = O(1)$が導かれる.

横型アルゴリズムを**2.3**の再帰関数gを用いて定義した場合, コストを$c(g, L)$と書くことにすると, 時間複雑度は, $c_T(g, L) = c_T(f, left(L)) + c_T(f, right(L)) + 1$. これから, 一般に, $c_T(g, L) = O(|L|)$が導かれる. 空間複雑度は, $c_S(g, L) = c_S(f, left(L)) + c_S(f, right(L))$. これから, 一般に, $c_T(f, L) = O(|L|)$が導かれる.

3.2 数値の最大値, 最小値

2項演算◯が“2数の大きい方”maxの場合は,

$$a_1 \bigcirc a_2 \bigcirc \cdots \bigcirc a_n = \max(a_1, a_2, \ldots, a_n) \tag{11}$$

となる. 単位元は∞である. 具体的な記述例は自明であろう.

同様に, 2項演算◯が“2数の小さい方”minの場合は,

$$a_1 \bigcirc a_2 \bigcirc \cdots \bigcirc a_n = \min(a_1, a_2, \ldots, a_n) \tag{12}$$

となる. 単位元は$-\infty$である.

3.3 整列

数値が昇順に並べられた二つのリストを併合(マージ)して一つの昇順リストを作る演算を基本とするアルゴリズムである. したがって, $a_1 \bigcirc a_2 \bigcirc \cdots \bigcirc a_n$の結果は, $a_1, a_2, \ldots, a_n$を昇順に整列したリストである.

縦型アルゴリズム$(\cdots(a_1 \bigcirc a_2) \bigcirc \cdots) \bigcirc a_n$は, 通常の整列のことばで述べれば,

ステップ1 a_1とa_2を比較し, 小さい方を1番目, 大きい方を2番目に並べる.
ステップ2 a_1, a_2, a_3が昇順になるように, a_3をステップ1の結果のリスト中の該当する箇所に挿入する.
ステップ3 a_1, a_2, a_3, a_4が昇順になるように, a_4をステップ2の結果のリスト中の該当する箇所に挿入する.
.....

というもので, いわゆる挿入法(insertion sort)である.

縦型アルゴリズムを再帰関数fを用いて定義した場合, コストを$c(f, L)$と書くことにすると, 時間複雑度は, $c_T(f, La) = c_T(f, L) + |L| + 1$. これから, 一般に, $c_T(f, L) = O(|L|^2)$が導かれる. 空間複雑度は, $c_S(f, La) = c_S(f, L) + 1$. これから, 一般に, $c_T(f, L) = O(|L|)$が導かれる.

横型アルゴリズムは,

ステップ1 $a_1a_2, a_3a_4, a_5a_6, \ldots$のように長さ2ずつの部分リストに区切り, それぞれの区切りを昇順に整列する.
ステップ2 左端から1番目と2番目, 3番目と4番目, 5番目と6番目, ...の部分リストをマージする.
ステップ3 左端から1番目と2番目, 3番目と4番目, 5番目と6番目, ...の部分リストをマージする.
.....

というマージソート(merge sort)である.

横型アルゴリズムを再帰関数gを用いて定義した場合, コストを$c(g, L)$と書くことにすると, 時間複雑度は, $c_T(g, L) = c_T(f, left(L)) + c_T(f, right(L)) + |L|$. これから, 一般に, $c_T(g, L) = O(|L| \log |L|)$が導かれる. 空間複雑度は, $c_S(g, L) = c_S(f, left(L)) + c_S(f, right(L)) + |L|$. これから, 一般に, $c_T(f, L) = O(|L|)$が導かれる.

通常のマージソートは，リストを極大な部分昇順リスト(連(run)という)に区切り，“隣り合う二つの連同士をマージする”という操作を繰り返すものであり，上記のアルゴリズムとは少々異なる．

リストL中の左端から1番目の連を$run1(L)$と書くことにする．左端から1番目と2番目，3番目と4番目，5番目と6番目，...の連をマージする操作$halfsort(L)$は，

$$halfsort(L) = (run1(L) \bigcirc run1(L - run1(L))halfsort(L - run1(L - run1(L))) \qquad (13)$$

と記述される．ただし，$L - run1(L)$はリストLから最初の連を除いた残りのリストである．通常のマージソートは$halfsort(L)$を使って

$$\textbf{while } L \neq run1(L) \textbf{ do } halfmerge(L)$$

と記述される．

なお，マージの定義は，本論文の記法によるならば，

$$L \bigcirc L' = \textbf{if } L = \textbf{nil } \textbf{then } L' \textbf{ else if } head(L) < head(L') \textbf{ then } head(L)((L - head(L)) \bigcirc L') \textbf{ else } head(L')(L \bigcirc (L' - head(L')))$$

ただし，$L - head(L)$はリストLから最初の要素を除いた残りのリストである．

3.4 k番目に小さな値

1番目からk番目までの要素が昇順に並んでいるリストに対する演算によって定義されるアルゴリズムである．このようなリストL, L'に対して，$L \bigcirc L'$は，長さkの昇順リストであり，その要素は両者のリスト中の要素で小さい方からk番目までのものである．(LとL'の要素の個数の和lがk未満のときは$L \bigcirc L'$の長さはその個数lとする)．

このリストに対する$a_1 \bigcirc a_2 \bigcirc \cdots \bigcirc a_n$の結果は，$a_1, a_2, \ldots, a_n$の中で小さい方から$k$個が1番目から$k$番目までに昇順に並んだリストである．これによって，小さい方からk番目の要素を求めるアルゴリズムが導かれる．

コストを時間複雑度について評価すると，縦型アルゴリズムの場合，$c_T(f, La) = c_T(f, L) + \min(|L|, k)$であるから，一般に，$c_T(f, L) = 1 + 2 + \cdots + k + (n - k)k = O(nk)$である．横型アルゴリズムの場合，$c_T(f, L) = c_T(f, left(L)) + c_T(f, right(L)) + \min(|L|, k)$であるから，一般に，$c_T(f, L) = O(n \log k)$である．

3.5 ヒープに関するアルゴリズム

リスト$a_1, a_2, \ldots, a_n$が

$$a_i \leq a_{2i}, \qquad a_i \leq a_{2i+1}$$

を満たすとき，このリストを**ヒープ**(heap)と定義することにする．ヒープに対しても，上記と同様の2項演算と代数系を定義することにより，各種のアルゴリズムが導かれる．

4 おわりに

リストに関して2項演算とそれに基づく代数系を定義することにより，アルゴリズムを一般的に記述し，その特殊な場合として多くのアルゴリズムを導くことができることを示した．

今後は，リストに関する探索についても同様のことを試みたい．また，グラフやネットワークに関する多くのアルゴリズムについても試みたい．また，本論文のような記述形式の下で個々のアルゴリズムの構造上の特徴を捉える方法について研究を進めてみたいと考えている．

謝辞

本研究に関して討論いただいた奈良先端科学技術大学院大学情報科学研究科植村・吉川研究室の諸氏，東京農工大学工学部情報工学大講座アルゴリズム設計工学研究室の諸氏に深謝申し上げます．

本研究は，平成5，6年度文部省科学研究費補助金一般研究(C)05680267の援助を受けた．

参考文献

[1] Naur, P. (ed.), Report on the algorithmic language ALGOL60, *Comm. ACM*, **3**, 299-314 (1960).

[2] Jensen, K. and Wirth, N., *PASCAL User Manual and Report*, Springer, 1976.

[3] Aho, A.V., Hopcroft, J.E., and Ullman, J.D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.