
ま え が き

コンピュータが普及し、社会でも家庭でも種々な場面で利用されるようになってきた。さらに2002年度からの新しい学習指導要領では、小中高に一貫したカリキュラムが用意され、情報教育が脚光を浴びるようになった。情報教育とは情報活用をの能力を育成することであり、その中の「情報の科学的理解」は、中学校での「情報とコンピュータ」や高校での必修教科「情報」など専門の教科として取り上げられるようになった。とはいえ、高校までは、コンピュータを授業科目の支援ツールとして用いることが主たる内容であろうが、近い将来、コンピュータに興味を持つ大学生が急激に増加することは間違いないであろう。特にコンピュータサイエンスの中でもソフトウェアの比重は大きくなり、社会の複雑化や産業界のIT化に伴い、今後ソフトウェア技術の発展は一段と高まっていくであろう。

コンピュータはソフトウェアによって動き、与えられた問題を解くためにはプログラムを作成しなければならない。同じ性能のコンピュータを用いてもプログラムの良し悪しによってコンピュータの働き具合が違ってくる。良いプログラムであるかどうかは、その記述が明瞭でかつそこで用いられているアルゴリズムの効率が良いかどうかで決まる。プログラミングは与えられた問題の解法手順の記述と言語への翻訳の二つの工程からなる。

アルゴリズムとは問題に対する解法の筋道あるいは手順のことであり、プログラミングにおける基本的な考え方を提供するものである。この意味でアルゴリズムは情報工学のコアであるばかりでなく、多くの分野で問題を解いていく考え方を学ぶことに役立つ。

アルゴリズムやデータ構造に関する教科書や参考書は、いままでに数多く出

版されている。しかしながら、その多くは理工系、特に情報工学科や情報科学科の学生向きに企画されたものであって、高度な知識や技法の学習が目的とされている。あるいは大学生レベル以下の幼稚とも思える内容のものに二分されている感がないでもない。これらの中には、C 言語や Java を用いてその例題的な構成でアルゴリズムを説明し、むしろプログラミング言語の学習書となっているものもある。

本書で取り上げたアルゴリズムの多くは一般に良く知られている基本的なものである。このようなアルゴリズムを紹介しながら、本書ではプログラミングあるいはアルゴリズムの初心者を対象に、効率の良いアルゴリズムの設計の基本的な考え方と技法について丁寧な説明を心がけたつもりである。著者の一人は、1981年に初めてわが国で情報工学科が設立されたときから情報工学の教育にプログラミング言語などの授業担当として携わってきた。その経験からすれば、プログラミングほど個人の能力の差が顕著に表れる科目はない。しかし、基本的なアルゴリズムについては経験や知識の差であるし、考え方に慣れていないためである。本書は、著者らが武蔵工業大学で文系の学生を対象とするアルゴリズムの講義録を基に執筆したものである。対象が文系の1年次学生であるため、できるかぎりわかりやすく丁寧な説明を心がけたつもりであるが、一方では、ある程度アルゴリズムに慣れた読者のために種々なアルゴリズムを紹介するとともに、章末問題には解答を付して独学で学習できるようにもなっている。さらに自分でプログラムを作成して実際にコンピュータ上で動作確認を行いたい読者のために、代表的なアルゴリズムについてはフローチャートだけでなく、C 言語で記述したプログラムを巻末に添付しておいた。

本書がプログラミングおよびアルゴリズムの学習を目指す読者に少しでも役立てば幸いである。

2004年8月

著者しるす

改訂版の出版にあたり

先のまえがきで述べたように、本書はもともと文系の学生を対象として執筆された。しかし、情報システム分野を専門とする学生には、一つの目的を達成するための手段として多様なアルゴリズムを習得し、状況に応じて適切なアルゴリズムを選択できるようにすることが肝要である。特に、データ構造の異なるアルゴリズムを知ることで、他の処理への応用力をも身につけることができる。

そこで、改訂版では、木構造を用いて探索を行う二分探索木について加筆した。二分探索木のアルゴリズムでは再帰呼出しが多用されるため、再帰の理解を深めるためにも役立つことが期待される。他の探索アルゴリズムとの特徴の違いを比較しつつ、各処理の手順と必要性を理解することで、読者の知識の引出しの一つを埋めていただけると幸いである。

2013年5月

著者しるす

目 次

1. プログラミングとアルゴリズム

1.1	プログラミング	1
1.1.1	ソフトウェアとプログラミング	1
1.1.2	プログラミング言語	2
1.2	アルゴリズム	3
1.2.1	アルゴリズムの必須条件	5
1.2.2	アルゴリズムの選択基準	6
1.3	計 算 量	9
1.3.1	時間計算量	9
1.3.2	O 記 法	12
1.3.3	最大計算量と平均計算量	13
1.3.4	領域計算量	14
	章末問題	15

2. データ構造

2.1	データ構造	16
2.2	変数と定数	17
2.3	基本データ型	20
2.3.1	単 純 型	20
2.3.2	ポインタ型	21
2.4	構 造 型	23
2.4.1	配 列 型	23
2.4.2	レコード型	26
2.5	抽象データ型	27
2.6	リ ス ト	28
2.7	スタック	30
2.8	キュー	32
2.9	木	33

2.10 ハ ッ シ ュ	36
章末問題	38

3. フローチャート

3.1 フローチャート	39
3.2 流れ図記号	39
3.2.1 端子・処理・データ・準備	40
3.2.2 判 断	42
3.2.3 ループ端	43
3.2.4 並列処理	45
3.2.5 定義済み処理	46
3.2.6 注 釈	47
3.2.7 結 合 子	47
章末問題	48

4. サブルーチンと再帰

4.1 サブルーチン	49
4.1.1 階 乗	49
4.1.2 実数の整数部分と整数除算の剰余	52
4.1.3 検 査 数 字	54
4.1.4 配列の要素の交換	56
4.2 再 帰	57
4.2.1 階乗の再帰的定義	57
4.2.2 自然数の和	59
4.2.3 アッカーマン関数	61
4.2.4 フィボナッチ数列	63
4.2.5 ハノイの塔	64
4.2.6 エイトクイーン	67
章末問題	73

5. いろいろなアルゴリズム

5.1 最大値と整列	75
5.1.1 配列要素の最大値	76
5.1.2 選択法による整列	77
5.2 10進数から2進数への変換	80
5.3 素 数	82
5.3.1 逐 次 法	82

5.3.2	エラトステネスのふるい	84
5.3.3	N 個の素数	92
5.3.4	アルゴリズムの比較	93
5.4	最大公約数	94
5.4.1	手計算と同じ方法	94
5.4.2	差を用いる方法	97
5.4.3	ユークリッドの互除法	99
5.4.4	アルゴリズムの比較	101
章末問題		102

6. 探 索

6.1	線形探索	103
6.1.1	配列の線形探索	104
6.1.2	単方向リストの線形探索	108
6.2	二分探索	111
6.3	二分探索木	120
6.4	ハッシュ探索	124
章末問題		129

7. 整 列

7.1	バブルソート	130
7.2	クイックソート	133
7.3	ヒープソート	141
7.3.1	整 列	141
7.3.2	追加と削除	146
章末問題		148

8. 文字列照合

8.1	基本の文字列照合	149
8.2	番兵を用いた文字列照合	151

付録 C	C 言語プログラムソース	154
章末問題	解答	170
索 引		181

1

プログラミングとアルゴリズム

1.1 プログラミング

近年のコンピュータの普及には目を見張るものがあり、企業や大学だけではなく、一般家庭や小中学校に至るまで、あらゆるところでコンピュータが使用されるようになった。その用途は、文書作成、画像処理、音声処理、データ処理、ゲーム、電子メール送受信、Web ページ閲覧など、多岐にわたっている。最近のコンピュータは高性能で機能も多いため、これまで人間が多大な労力を使って行ってきた作業を一瞬のうちにやりとげることができる。さらに、一昔前までは想像も及ばなかったような処理が可能となった。さまざまな処理を高速に行うコンピュータを目の前にすると、われわれは「コンピュータはなんと賢いのだろう！」と感心しがちであるが、本当にコンピュータは賢いのだろうか。

1.1.1 ソフトウェアとプログラミング

コンピュータは、おもにハードウェアとソフトウェアから構成されている。ハードウェアとはコンピュータを構成している物理的な機器のことであり、ソフトウェアとはハードウェアを制御するための手順や命令をまとめたものである。一般にコンピュータの性能としては、中央処理装置（central processing unit; CPU）の動作速度やメモリ容量など、ハードウェアの仕様が注目される。しかし、いくら高性能のチップが組み込まれていても、自分が望んでいる

2 1. プログラミングとアルゴリズム

処理を高速かつ正確に行えなければ、「賢いコンピュータだ！」と思うことはないだろう。ソフトウェアによってハードウェアの性能を最大限に引き出してこそ、「賢い」と思えるような処理が行える。すなわち、ハードウェアを制御するソフトウェアがなければ、「コンピュータはただの箱」といっても過言ではない。

ソフトウェアには、大別してシステムソフトウェアとアプリケーションソフトウェアの2種類がある。システムソフトウェアは、ソフトウェアの中でもハードウェアの管理や制御をダイレクトに行うもので、WindowsやMac OS, UNIX, Linuxなどコンピュータ動作の基盤を提供するオペレーティングシステム (operating system; **OS**)、コンパイラやインタプリタ、データベースソフトなどユーザの入力とコンピュータの仲介役を果たすミドルウェアなどがこれにあたる。アプリケーションソフトウェアは、ワープロソフトや表計算ソフト、プレゼンテーションツール、電子メールソフト、Webブラウザなど、ユーザが目的の作業を行うために用いるソフトウェアである。

ソフトウェアには、各ハードウェアを動作させる手順が書かれている。これをプログラムといい、プログラムを書くことをプログラミングという。通常、OSや市販のソフトウェアのプログラムを変更することはないが、既存のソフトウェアを使った高度な処理や、既存のソフトウェアでは行えない独自の処理が必要な場合には、目的に合うプログラムを自分で書かなければならない。

1.1.2 プログラミング言語

プログラムを記述する言語をプログラミング言語という。コンピュータは人間の言葉を理解することができないため、人間はプログラミング言語を用いてコンピュータに命令を伝達する。人間とコンピュータの橋渡しをするのがプログラミング言語なのである。

コンピュータが理解できる唯一の言語は機械語であり、その実体は2進法で書かれた0と1の並びとなっている。機械語はそのままコンピュータを動作させることができる反面、人間にとっては非常にわかりにくい。機械語の命令を表す数値を人間が理解しやすい文字列に1対1対応させた言語がアセンブリ言

語である。アセンブリ言語で書かれたプログラムは、アセンブラで機械語に変換してから実行される。機械語やアセンブリ言語のように、コンピュータの各部への命令を記述するプログラミング言語を低水準言語という。低水準言語は、コンピュータ内部を直接操作するため非常に効率が良いが、コンピュータ内部に関する詳細な知識が必要であり、ごく基本的な命令しか用意されていないこともあって、複雑な処理の記述には不向きである。

低水準言語に対し、人間の言葉に近いプログラミング言語を高水準言語という。高水準言語の第1号は、IBMによって開発されたFortranであり、現在でも幅広い分野で科学技術計算を中心に利用されている。Fortranのプログラムは、処理手順を順番に記述していくタイプであり、手続き型あるいは命令型と呼ばれている。Fortranから派生した手続き型言語には、事務処理分野を中心に普及したCOBOLや、プログラミング教育用に開発されたBASICなどがある。また、構造化プログラミングの概念を取り入れたPascalやC言語も手続き型言語であり、さらにオブジェクト指向の概念を組み込んだC++やJavaが現在のプログラミングの主流となっている。

手続き型言語以外には、数学の関数の形式でプログラムを記述する関数型のLISPや、論理型のPrologなどがある。これらはおもに人工知能の研究分野で使われた。また、小規模な簡易プログラム向けのスクリプト言語として、sh, awk, Perl, JavaScript, Rubyなどが使われている。

以上に示したように、プログラミング言語には数多くの種類が存在する。これは、人間が話す言葉にも日本語や英語、フランス語、ドイツ語、中国語など、非常に多くの種類があるのと同じである。プログラムを組むときには、目的や状況に応じて適切なプログラミング言語を選択することが重要な課題となる。

1.2 アルゴリズム

何かあることをしようとする場合には、何を行うか、そしてどのような手順で進めていくかを考えることが大切であり、目的を達成できるかどうかはこの

4 1. プログラミングとアルゴリズム

準備に大きくかかっているといってもよいだろう。プログラミングについても例外ではなく、目的を達成するための処理手順を考えることがプログラミングにおける作業の中核をなしているといえる。一般に、コンピュータを使ってある目的を達成したり、問題を解決したりするための処理手順のことをアルゴリズム (algorithm)[†]という。このアルゴリズムをプログラミング言語によって記述したものをプログラムと呼んでいる。

アルゴリズムの例を挙げてみよう。図 1.1 は電車に乗るまでの手順を示したものである。まず、運賃表で目的の駅までの運賃を調べ、券売機で乗車券を購入する。その後、購入した乗車券を自動改札機に通して駅構内に入り、電車に乗ることになる。これが「電車に乗る」という目的を達成するためのアルゴリズムである。



図 1.1 電車に乗るまでの手順

つぎに、算数の文章題を解く手順を考えてみよう。

1 個 110 円のりんごと 1 個 70 円のみかんを買い、680 円払いました。りんごとみかんをいくつつ買いましたか。

この問題では、りんごとみかんの個数が整数であることがヒントになる。みかんの個数はりんごの個数を用いて式 (1.1) により求めることができる。

$$\text{みかんの個数} = (680 \text{ 円} - 110 \text{ 円} \times \text{りんごの個数}) \div 70 \text{ 円} \quad (1.1)$$

りんごは最多で 6 個買えるので、りんごを 1~6 個買ったときのみかんの個数を調べればよい。

- りんごが 6 個のとき、みかんは 0.28 個
- りんごが 5 個のとき、みかんは 1.85 個

[†] イランの数学者アルフワリズミ (al-Khwarizmi) の名前が語源といわれている。

- りんごが4個のとき、みかんは3.42個
- りんごが3個のとき、みかんは5個
- りんごが2個のとき、みかんは6.57個
- りんごが1個のとき、みかんは8.14個

この結果、りんごが3個のときにみかんがちょうど5個になるので、これが答えであることがわかる。この手順を図示したものが図 1.2 である。このように、なんらかの目的を達成するための手順をアルゴリズムというのである。

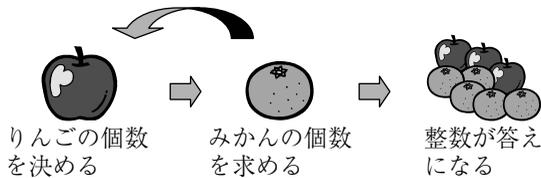


図 1.2 りんごとみかんの個数を求める手順

同じ目的を達成したり、同じ問題を解いたりする場合にも、いくつかの方法や手順があることが多い。例えば、駅までの道順をだれかに教える場面を考えてみよう。小さな路地を通る最短経路や、目印のわかりやすい経路、遠回りだが安全な経路など、いくつかの道順がある場合には、相手の土地勘や体力などを考慮した経路を選んで教えるだろう。また、相手がフランス人ならフランス語で表現しやすい経路を選択するかもしれない。これと同様に、プログラミングにおけるアルゴリズムも、どのような問題を対象とし、どのような環境で実行されるのか、どのプログラミング言語を使用するかなどを考慮し、与えられた状況の中で最良と思われるアルゴリズムを選択することが望ましい。

1.2.1 アルゴリズムの必須条件

アルゴリズムについてももう少し詳しく説明しておこう。処理手順を表すアルゴリズムは、つぎのような条件を満たしていなければならない。

条件 1 得られた処理結果はつねに正しい。

条件 2 有限時間内に処理が終了する。

6 1. プログラミングとアルゴリズム

どのアルゴリズムも目的の達成や問題の解決を目指しているため、処理結果に対する信頼性が最も重要といえる。誤った処理結果を出力する可能性の高いアルゴリズムでは、得られた結果への信頼性が低く、確実に目的を達成することはできない。初期条件やデータ、実行状況を変えるといろいろな処理結果が得られると思われるが、なんらかの結果が得られたならば、その結果はすべて正しいという保証が必須である。

正しい結果が得られるとしても、結果を出力するまでに膨大な時間がかかるのでは、結果が得られないのに等しい。また、解が得られるまで探索を続けようとしても、解が存在しなければ処理が終了することはない。

例えば、迷路の出口を探すことを考えてみよう。右側もしくは左側の壁に沿っていくと必ず出口を見つけることができる。しかし、非常に入り組んでいる迷路でこれを行うのは気が遠くなりそうな作業であり、非現実的な方法と思われる。また、出口のない迷路に迷い込んだ場合には、いつまで探しても出口は見つからないのだから、どこかであきらめて入口から出るしかない。

したがって、有限時間内に処理が終了することも必須条件となる。解が見つからない場合には、途中結果や近似解、「解なし」「処理不能」などのメッセージを出力し、終了させたほうがよい。

1.2.2 アルゴリズムの選択基準

複数の異なるアルゴリズムの中から最良のアルゴリズムを選択する際に、つぎのようないくつかの基準を考えてみる。

- ① 実行時間 ② メモリ使用量 ③ わかりやすさ
- ④ 実現しやすさ ⑤ 使用頻度 ⑥ 拡張性
- ⑦ 対象となる問題の大きさ

以下、これらについて説明していこう。

- ① **実行時間** 同じ解が得られるならば、もちろん速く求められるほうがよい。最近のコンピュータは処理が高速で、簡単な処理にかかる時間はまったく気にならないほどであるが、大量データの処理ではちょっとした

アルゴリズムの工夫で実行時間を大幅に短縮できる。プログラムの実行時間は、CPU やメモリの性能、アーキテクチャの特性、OS やコンパイラの性能、実行のタイミング、メモリ容量などの環境要因による部分が大きい。そのため、プログラムの実行時間を単純に比較してアルゴリズムの優劣を決めることはできない。実際にアルゴリズムの性能を評価する際には、「時間」ではなく、各処理の実行回数である「ステップ数」を比較する。処理が終了するまでのステップ数によって実行時間を表現した指標を時間計算量 (time complexity) あるいは単に計算量 (complexity) と呼び、アルゴリズムの性能評価基準として最重要視することが多い。

- ② **メモリ使用量** メモリとは、データを記憶しておくための装置である。コンピュータの記憶装置には、CPU が直接に読み書きできる主記憶装置 (main memory) と、ハードディスクやフロッピーディスクなどの補助記憶装置 (auxiliary storage unit) がある。前者は読み書きが高速である反面、電源を切ると記憶内容が消えてしまうという欠点がある。逆に後者は、読み書きに時間がかかるが、電源を切っても記憶内容を保持することができる。単に「メモリ」という場合は主記憶装置を指し、アルゴリズムの評価には主記憶装置という意味でのメモリの使用量が重要になる。時間計算量に対して、アルゴリズムの実行に必要なメモリ使用量の指標を領域計算量 (space complexity) と呼んでいる。当然ながらメモリ容量は有限であるため、容量を超える作業領域が必要なアルゴリズムは適切ではない。あまり多くのメモリを使わないアルゴリズムのほうがよいといえる。
- ③ **わかりやすさ** 目標としている処理結果の獲得がアルゴリズムの最大の目的ではあるが、最低限の目的でもあり、その目的さえ達成されればどのようなアルゴリズムでもいいというわけではない。アルゴリズムのわかりやすさというのも重要な項目である。特に、グループで一つのプログラムを作成する場合や、作ったプログラムを別の人に引き継ぐ場合、あとからプログラムを変更する場合などには、わかりやすいアルゴリズムのほうが作業効率が良く、誤りの可能性も低い。先に述べた実行時間やメモリ使

用量を重視するために、わかりやすさを犠牲にする場合もあるが、アルゴリズムが複雑になればなるほど間違える危険性が高くなるので、細心の注意を払わなければならない。

- ④ **実現しやすさ** わかりやすさと実現しやすさは同じようなものと思われがちであるが、わかりやすいアルゴリズムがプログラムで実現しやすいとは限らない。用いるプログラミング言語によっては、簡単なアルゴリズムでも記述が難しいことがある。他の要因と絡めて実現しやすさを考慮することも、作業効率の向上と誤りの可能性の低下につながる。
- ⑤ **使用頻度** 実現しやすさと深く関連している基準に使用頻度がある。使用頻度の低いプログラムの場合は、実行する時間よりも作成する時間のほうが、そのプログラムにかかわる作業時間の大部分を占めることになる。したがって、多少遅いプログラムでも、プログラミングが簡単で短時間に行えるほうが、作業開始から結果を得るまでの時間を短縮し、労力を軽減することができる。
- ⑥ **拡張性** アルゴリズムを考えるにあたっては、もちろん当面の目的が達成されるものでなくてはならないが、あとに機能拡張が必要な場合も少なくない。したがって、つねに拡張性を考えながらアルゴリズムを選ぶことが重要である。
- ⑦ **対象となる問題の大きさ** 先に述べた実行時間は対象とする問題の大きさに依存するため、計算量を比較する際には、一般に問題が十分大きい場合を想定している。しかし、実際にどのような大きさの問題を扱うかわかっている場合には、対象となりうる大きさの範囲内で、計算量をはじめとするいろいろな項目について考え、適切なアルゴリズムを選ぶことが望ましい。

以上に挙げた7項目はあくまでもおこなった選択基準であり、これ以外にも基準とすべき項目は存在する。また、どの項目を最優先すべきかを断定できるものでもない。プログラミングにおけるすべての要因を考慮したうえで、目的と状況に応じた最良のアルゴリズムを考えなくてはならない。

索引

【あ】

アセンブラ	3
アセンブリ言語	2
アッカーマン関数	61
後入れ先出し	31
アドレス	21
アプリケーションソフト	
ウェア	2
アルゴリズム	4

【い】

入れ子状態	80
インタプリタ	2

【え】

エイトクイーン	67, 154
枝	33
エラトステネスのふるい	84, 93, 156, 157

【お】

オーダ	12
オブジェクト指向プログラ	
ミング	28
オペレーティングシステム	2
親	34, 142

【か】

階乗	49, 57
開番地法	124
外部節点	34
カウンタ変数	43
拡張性	8, 18
仮引き数	50

間順走査	36
環状リスト	30
関数	56
関数型	3
完全二分木	35, 141

【き】

木	33
キー	36, 103
機械語	2
利き筋	67
利きマス	67
木の高さ	34, 141
基本データ型	20
基本データ構造	17
キュー	32
兄弟	34
均一ハッシュ法	124

【く】

クイックソート	133, 166, 167
組合せ	49

【け】

計算量	7
経路	34
結合子	47
検査数字	54

【こ】

子	34, 141
降順	76, 77, 142
後順走査	36
高水準言語	3
構造型	23

構造体	27
コンパイラ	2

【さ】

再帰	58, 134, 138
再帰関数	58
再帰的定義	58
再帰呼出し	58
最大計算量	14
最大公約数	94, 157, 158
最大値	75, 77
先入れ先出し	32
削除	103-105, 108, 109, 117, 120, 128, 146
サブルーチン	46, 49
サブルーチン INT	53
サブルーチン REM	53
サブルーチン SWAP	57, 78, 132
差を用いる方法	97, 158

【し】

時間計算量	7, 9
システムソフトウェア	2
システム流れ図	39
自然数の和	59
子孫	34
実行時間	6
実数型	20
実数型ポインタ	22
実数の整数部分	52
実引き数	50
終端節点	33, 34, 143
終了条件	43
主記憶装置	7

主要項	12	多分木	35	二分探索木	120
順序木	33	ダミー要素	108		
準備記号	40	探 索	103, 128	【ぬ】	
昇 順	112, 130, 142	端子記号	40	スルポイント	22
衝突問題	124	単純型	20		
情報処理流れ図	39	単方向リスト	28	【ね】	
初期値	43	——の線形探索	108	根	34, 142
処理記号	40			【は】	
【す】		【ち】		葉	34, 141
スカラ型	20	チェーン法	124	ハードウェア	1
スクリプト言語	3	チェス	67	配列型	24, 26, 28
スタック	30	逐次法	82, 93, 156	配列の線形探索	104, 151,
スタックポインタ	31	中央処理装置	1	159	
ステップ数	9	注釈記号	47	配列要素の最大値	76
		抽象データ型	27	破 線	47
【せ】		長方形の面積	41	バックトラッキング	69
整数型	20			ハッシュ	36
整数型ポインタ	22	【つ】		ハッシュ関数	37, 124
整数除算の商	41	追 加	103-105, 108, 109,	ハッシュ探索	124, 165
整数除算の剰余	52	116, 120, 126, 146		ハッシュ値	37, 124
整 列	77	【て】		ハッシュ表	37, 124
節 点	33	定義済み処理記号	46	ハノイの塔	64, 154
——の深さ	34	低水準言語	3	バブルソート	130, 166
——のレベル	34	定 数	17	半順序木	34, 141
線	40	データ型	19	判断記号	42
漸化式	63	データ記号	40	番 兵	107, 108, 151, 159
線形探索	103	データ構造	16		
線形リスト	28	データ流れ図	39	【ひ】	
前順走査	36	データ部	28	ヒープ	141
選択法	79, 155	手計算と同じ方法	94, 157	ヒープソート	141, 167
		手続き	56	引き数	50
【そ】		手続き型	3	非終端節点	33, 34, 141
走 査	36			左部分木	34, 144
双方向環状リスト	30	【と】		標準 2 分木	34
双方向リスト	29, 110	等比数列の和	9, 13, 14, 18		
添 字	24			【ふ】	
素 数	82, 156, 157	【な】		フィールド値	27
祖 先	34	内部節点	34	フィールド名	27
ソフトウェア	1			フィボナッチ数	63
		【に】		フィボナッチ数列	63
【た】		二次元配列	25, 68, 125	プッシュ	31
代 入	18	二分探索	111, 161	部分木	33

部分範囲型 21 フラグ変数 21 フローチャート 39 プログラミング 2 プログラミング言語 2 プログラム 2, 4 プログラム流れ図 39 【へ】 平均計算量 14, 124, 133 平衡木 35 並列処理記号 45 変 数 17 【ほ】 ポインタ型 21 ポインタ部 28 補助記憶装置 7 ポップ 31 【ま】 増 分 43 待ち行列 32	【み】 右部分木 34, 144 ミドルウェア 2 【め】 命令型 3 メインルーチン 49 メモリ使用量 7, 14, 21, 93 【も】 文字型 20 文字型ポインタ 21 文字列照合 149, 151, 168, 169 戻り値 50 問題向きデータ構造 17 【ゆ】 ユークリッドの互除法 99, 158 優先度付き待ち行列 146 【よ】 要 素 24	【ら】 ラベル 34, 141 【り】 リスト 28 ——の線形探索 160 領域計算量 7, 14, 93 【る】 ループ開始記号 43 ループ終了記号 43 ループ端記号 43 【れ】 レコード型 27, 29, 37 列挙型 21 レベル 141 連想配列 36 【ろ】 論理型 3, 21
--	---	--



AVL 木 35 awk 3 B 木 35 BASIC 3 C 3 C++ 3 COBOL 3 CPU 1 false 21 FIFO 32 Fortran 3 ISBN 54 Java 3 JavaScript 3 LIFO 31	Linux 2 Lisp 3 Mac OS 2 N 個の素数 92 n 次元配列 25 N までの自然数を表示 43 NULL 22, 109 O 記法 12 OS 2 Pascal 3 Perl 3 Prolog 3 Ruby 3 sh 3 true 21	UNIX 2 Windows 2 ~~~~~ 2 進数 80 2 分木 34 10 進数 80 10 進数から 2 進数への変換 80 ~~~~~ = 18 ← 19 ↑ 22 & 22
--	---	--

— 著者略歴 —

大谷 紀子 (おおたに のりこ)	志村 正道 (しむら まさみち)
1993年 東京工業大学工学部情報工学科卒業	1960年 東京大学工学部応用物理学科卒業
1995年 東京工業大学大学院修士課程修了(情報工学専攻)	1965年 東京大学大学院博士課程修了(応用物理学専攻)
1995年 キヤノン株式会社入社	工学博士
2000年 東京理科大学助手	1965年 大阪大学講師
2002年 武蔵工業大学講師	1967年 大阪大学助教授
2006年 博士(情報理工学)(東京大学)	1976年 東京工業大学助教授
2007年 武蔵工業大学(現 東京都市大学) 准教授	1980年 東京工業大学教授
現在に至る	1997年 東京理科大学教授
	東京工業大学名誉教授
	2001年 武蔵工業大学教授
	2008年 武蔵工業大学名誉教授

アルゴリズム入門 (改訂版)

Introduction to Algorithms (Revised Edition)

© Noriko Otani, Masamichi Shimura 2004, 2013

2004年10月28日 初版第1刷発行

2013年7月24日 初版第7刷発行 (改訂版)

検印省略

著者 大谷 紀子
志村 正道
発行者 株式会社 コロナ社
代表者 牛来真也
印刷所 三美印刷株式会社

112-0011 東京都文京区千石 4-46-10

発行所 株式会社 コロナ社

CORONA PUBLISHING CO., LTD.

Tokyo Japan

振替 00140-8-14844・電話(03)3941-3131(代)

ホームページ <http://www.coronasha.co.jp>

ISBN 978-4-339-02474-6 (大井) (製本:愛千製本所)

Printed in Japan



本書のコピー、スキャン、デジタル化等の無断複製・転載は著作権法上での例外を除き禁じられております。購入者以外の第三者による本書の電子データ化及び電子書籍化は、いかなる場合も認めておりません。

落丁・乱丁本はお取替えいたします