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と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

2. データ構造

3. フローチャート

4. サブルーチンと再帰

5. いろいろなアルゴリズム

6. 探索

7. 整列

8. 文字列照合

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であろうか。

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

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

準備に大きくかかわ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。

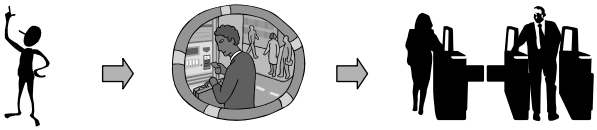

図 *1.1*　電車に乗るまでの手順

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ずつ買いましたか。

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。

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

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。

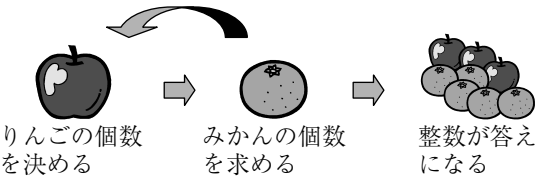

図 *1.2*　りんごとみかんの個数を求める手順

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。

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が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，妥当なアルゴリズムを選ぶことが望ましい。

以上に挙げた7項目はあくまでおもな選択基準であり，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

—— 著 者 略 歴 ——

大谷　紀子（おおたに　のりこ）
1993 年　東京工業大学工学部情報工学科卒業
1995 年　東京工業大学大学院修士課程修了(情報工学専攻)
1995 年　キヤノン株式会社入社
2000 年　東京理科大学助手
2002 年　武蔵工業大学講師
2006 年　博士（情報理工学）（東京大学）
2007 年　武蔵工業大学（現 東京都市大学) 准教授
現在に至る

志村　正道（しむら　まさみち）
1960 年　東京大学工学部応用物理学科卒業
1965 年　東京大学大学院博士課程修了（応用物理学専攻）工学博士
1965 年　大阪大学講師
1967 年　大阪大学助教授
1976 年　東京工業大学助教授
1980 年　東京工業大学教授
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)

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