原稿

１１章 問題解決 データ構造 アルゴリズム

1. 問題をコンピュータに解かせるということ

問題を解くことを考える前に，問題とは何であるかを考えよう．例えば「未来の宝クジの当選番号はなにか」という問いには正解がないので，問題にはなり得ない．一方「２×３はいくつになるか」に対しては，１０進数を仮定している限り正解は一つであり，電卓を使って計算することができる.「明日は何センチの雪が積もるか」や「読みたい本が近所の図書館にあるか」といった，実世界に直接関わる問題は，簡単には解けないように思える．ところが，最近では，インターネットに接続して検索を行うと，明日の天気予報や図書館蔵書情報を入手することができる．コンピュータを使って誰かが生成した情報をコンピュータを使って引き出しているのだから，全体として見ると，実は，コンピュータを使って，先の問題を解いていることに気づくであろう．天気予報や蔵書情報を作り出すためには，まず，実世界から情報を得なければならない．一方，これまでに説明した通り，コンピュータが直接扱うことのできるデータ型は，極めて単純な形式である．したがって，得られた情報をコンピュータが直接処理可能な形，すなわち，記号により表現し，最終的には，基本データ型に置き換えなければならない．うまく，実世界の対象物をデータに置き換えることができれば，情報を作り出すという漠然とした問題は，データ間の計算に単純化できる可能性がある．もちろん，コンピュータは単純な計算しかできないため，データには，実世界の対象物をわかりやすく少ない量で表現でき，かつ，現実のコンピュータの上で効率良く処理できることが求められる．

1.1 プログラム＝データ構造＋アルゴリズム

実世界の対象物をデータに置き換えるためには，基本データ型以外に，データどうしを関連づける何かが必要である．例えば一つの書籍に関する情報について考えると，著者および題名は日本語を含む文字列，発行年や価格は整数，そして，ISBN は英数字からなる文字列により表現できる．しかし，これらの複数のデータを一つの書籍データとして扱うためには，ばらばらのデータをまとめる構造が必要である．さらに，複数の書籍データを関連づける場合には，構造間の関係を表現する手段も必要である．このように，基本データ型だけではできない，情報の本質的な構造を反映する仕掛けが「データ構造」である．コンピュータが扱うデータ型に基本形があるのと同様に，データ構造にも，コンピュータが処理

しやすい基本構造がある．さて，データ構造が決まれば，コンピュータは仕事をしてくれるであろうか．答えは否である．データ構造は，図書館に置いてある図書カードに例えることができる．図書カードの様式を決めても，図書館の貸出カウンタに誰もいなければ，図書館は機能しない．本を借りるためには，図書カードの取り扱い方や本の貸し出し方といった仕事の手順を熟知したスタッフが必要である．この仕事の手順が「アルゴリズム」と呼ばれるものである．データ構造とアルゴリズムを伝えることにより，コンピュータに仕事をさせることが可能になる．伝えるための言葉がプログラミング言語であり，プログラミング言語で記述したものが「プログラム」である．同じ問題を解くプログラムであっても，データ構造とアルゴリズムが異なれば，プログラムも異なってくることは言うまでもない．

1.2　時間的コストと空間的コスト

プログラムの計算結果が同じであっても，プログラム自身が異なるということには，単なるプログラミング言語の違いやわかりやすさといった外見上の差も含まれる．しかし，より重要なのは，同じ問題を解くのに必要なコストが異なるという点である．コストと言うと経済的負担を思い浮かべる人が多いけれども，コンピュータが問題を解くコストとは，処理時間（時間的コスト）と必要なメモリ空間（空間的コスト）であり，データ構造とアルゴリズムによって決まる．さて，この２つのコストには，一般的に，一方を減らすと他方が増えるという困った関係があり，うまくバランスをとる必要がある．もちろん，極めて短時間に解かなければならない場合や，極めて限られたメモリしか使えない状況では，一方に極端な犠牲を強いることが正当化されるため，どのようなプログラムが最も優れているかを一概に判断することはできない．ただし，処理時間が同じであれば，より少ないメモリを使うプログラムが，また，メモリ量が同じであれば，より速く問題を解くプログラムが優れていると言うことはできる．また，コストを考える場合には，最良値，平均値，最悪値のそれぞれを考慮する必要がある．いくら最良値と平均値が良くても，特定の条件下で発生する最悪値が許容範囲を越える場合には，受け入れられないこともある．例えば，明日の天気予報を計算する２種類のプログラムの計算時間が，一方は最良１時間，平均２時間，最悪５０時間，他方は最良５時間，平均６時間，最悪８時間である場合，どちらが優れているだろうか．

(a)構造体　　(b)共用体

図 1　複合データ型

2. 複合データ型

書籍情報の例に示したように，基本データ型により表現されたばらばらの情報を一つにまとめて新しいデータ型とする考え方が，構造体と共用体であり，これらはまとめて複合データ型と呼ばれる．

2.1 構　造　体

複数の基本データ型を一つにまとめたものである．ただし，どの部分がどのような基本データ型であるかは，あらかじめ決められている．例えば書籍情報は，**図 1**（a）のような複合データ型により表現することができる．

2.2 共　用　体

構造体と同様，複数の基本データ型を一つにまとめたものである．ただし，各部分に複数の解釈が許される点が構造体とは異なる．例えば図 1（b）のような複合データ型を用意することにより，複合データが書籍情報またはＣＤ情報のどちらも表現することができる．もちろん，各々の複合データがどの情報を表現しているかを把握する必要がある場合には，データ内の共通部分に種別を格納しておかなければならない．例では，種別が 1 である場合は書籍情報，2 である場合はＣＤ情報であることを示している．

2.3 参照（リンク，ポインタ）

複数の基本データ型をまとめることができたら，次に必要なものは，複合データ間の関係を表現する手段である．後続章でとりあげるように，コンピュータがメモリ上のデータを扱う際には，メモリの内容とその番地の両方が必要である．実は，コンピュータが直接扱うことのできるデータには，これまでにとりあげたデータ型（内容に相当）の他に，データの場所（番地に相当）を表現することが

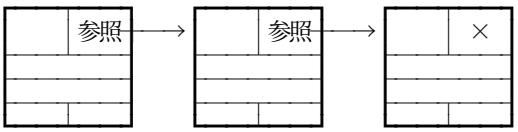


図 2　参照

できる「参照」がある．複合データ型の一部に参照（リンク，ポインタと呼ばれる場合もある）を含めることにより，データ間の上下関係やつながりといった，複雑な構造を表現することが可能になる．なお，**図 2** の最後のデータの参照欄が×（終端，空，Null と呼ぶ）となっている．最後のデータであることを表現するために，特別な値（たとえば 0）をもって，これ以上参照先がないことを表現するのが一般的である．

3. ならび（リスト）

データ構造には，いくつかの基本形がある．最も簡単な構造は，同じ型の基本データや複合データが複数個ならんだものであり，ならび（リスト）と呼ばれる．さらに，ならんでいる状態をコンピュータ上に表現する方法に「配列」と「つなぎ」がある．

3.1 配　　列

配列とは，ならびに属する各データに番号がつけられ，番号とデータの場所が，互いに簡単な計算により，相互に変換できる構造である．長所は，次のデータは必ず隣にあることがわかっているため，各データに参照が含まれる必要がない点，また，データを参照するのに要する時間的コストが，すべてのデータについて同じである点である．一方，短所は，配列全体の大きさを自由に変えることができない点，また，ならびの途中にデータを追加したり途中のデータを削除すると，後続データ全体をずらすために，大きな時間的コストを要する点である．

3.1.1 一次元配列

データが一方向にならんでいる配列を一次元配列と呼ぶ．仮に配列Aが 8 個のデータから構成され，先頭データが 0 番目，次のデータが 1 番目という具合に番号付けされている場合，それぞれのデータは，A [0]，A [1]，… ，A [7] と表現される．ｊ番目のデータにおける，ｊと場所の関係は，**図 3** のようになる．ｊ

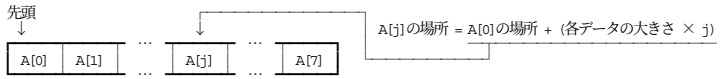


図 3　一次元配列

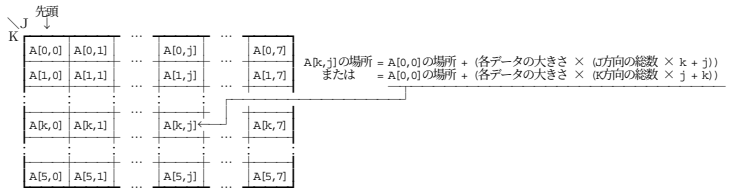


図 4　二次元配列

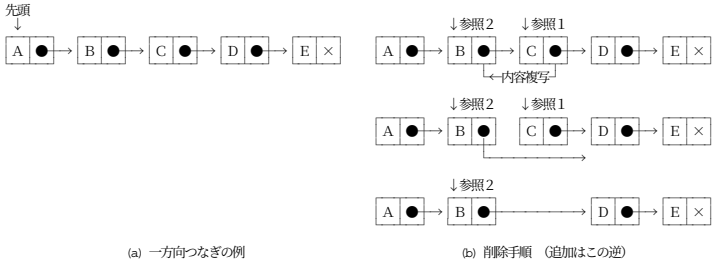
の値に関わらず，この計算方法が同じであることが，データを参照するのに要する時間的コストが一定であることの証明である．

3.1.2　二次元配列/多次元配列

同様に，データが二方向にならんでいる配列を二次元配列と呼ぶ．横方向に８，縦方向に６の合計４８個のデータは，Ａ[０, ０]，…，Ａ[５, ７]と表現される．[ｋ, ｊ] 番目のデータにおける，ｋ, ｊと場所の関係は，**図 4** のように二通りが考えられる．Ａ[ｋ, ｊ] に隣り合うデータがＡ[ｋ, ｊ＋１] とＡ[ｋ＋１, ｊ] のいずれであるかは，プログラミング言語により異なっており，例えばＣ言語では前者，ＦＯＲＴＲＡＮでは後者である．さらに添字を増やすことにより，多次元配列を構成することができる．

3.2　つ　な　ぎ

つなぎでは，各データに固有の番号はなく，次のデータが隣にある保証がない．次のデータを参照するためには，すでに場所がわかっているデータに含まれる参照を用いる．このことは一見不便に思えるが，配列ではできないことを可能にしている．長所は，つなぎ全体の大きさを自由に変えることができる点，また，後続データ全体をずらすことなく，ならびの途中にデータを追加したり途中のデータを削除することができる点である．一方，短所は，各データに参照を含めるために，空間的コストがかかる点，また，特定のデータを参照する際に必ず先頭から順に参照をたどる必要があるため，データを参照するのに要する時間的コスト



(a) 一方向つなぎの例　　(b) 削除手順 (追加はこの逆)

図 5　一方向つなぎ

が，データのつながり具合によって大きく異なる点である．

3.2.1　一方向つなぎ

データが一方向につながれている構造を一方向つなぎと呼ぶ．**図 5** は，５個のデータ（Ａ～Ｅ）が参照（●）によりつながれている様子を示している．特定のデータを参照するためには，必ず先頭から順に参照をたどらなければならず，１つの参照をたどる時間的コストを１とすると，Ｎ個のデータのうちの１つを参照するための時間的コストは，最良１，平均Ｎ／２，最悪Ｎである．さて，図５において，データＣを削除するには，まず，先頭から参照をたどりＣを見つける．この時重要なのは，現在注目しているデータの場所を保持する「参照１」とは別に，１つ手前のデータの場所を保持する「参照２」が必要な点である．参照１がＣを発見した時点で，いきなりＣ内の参照を消してしまうと，その先につながれているＤおよびＥが行方不明となってしまうため，Ｃの内容を消す前に，参照２が指しているＢ内の参照にＣ内の参照を複写する必要がある．複写が終われば，ＤおよびＥはＢにつながれるので，Ｃを消去することができる．このように，つなぎにおいては，つなぎかえの手順が極めて重要である．削除の逆の手順により追加を行うことができる．

3.2.2　双方向つなぎ

データが双方向につながれている構造を双方向つなぎと呼ぶ．データを参照するためのコストは一方向リストと同じであるが，データの追加や削除の際に参照２が不要であり時間的コストが多少改善される．ただし，一つのデータあたり右手と左手という二つの参照が必要となるため，空間的コストは上昇する．**図 6** においてデータＣを削除するには，まず，先頭から参照をたどりＣを見つける．

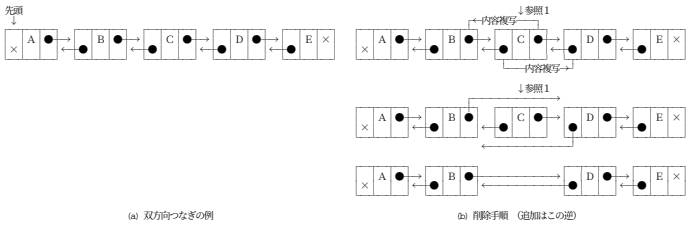


図 6　双方向つなぎ

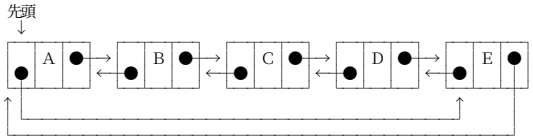


図 7　循環つなぎ

参照１がＣを発見した時点で，Ｃの左手が指しているＢ内の参照に，Ｃの右手の内容を複写し，同様に，Ｃの右手が指しているＤ内の参照に，Ｃの左手の内容を複写する．複写が終われば，ＤおよびＥはＢの右手に，また，ＡおよびＢはＤの左手につながれるので，Ｃを消去することができる．

3.2.3　循環つなぎ

一方向つなぎや双方向つなぎでは，両端部分の参照は終端である．実は，これまでに説明した追加や削除の手順には，さらに，終端部分の特別扱いが必要である．この特別扱いをなくすために，**図 7** に示すように，双方向つなぎの両端部分をつないで全体を環状にしたデータ構造が循環つなぎである．ただし，双方向つなぎの場合，終端に到達することによりデータ構造の終わりを知ることができるのに対し，循環つなぎの場合には，同じ方法ではデータ構造の終わりを知ることができないため，常に先頭の場所と比較するなど，一巡したかどうかを確認する手段が新たに必要となる．

4.　キューとスタック

以上に説明したならびは，時間的コストや空間的コストは様々であるけれども，任意の位置のデータを参照・追加・削除することが許されるという共通点が

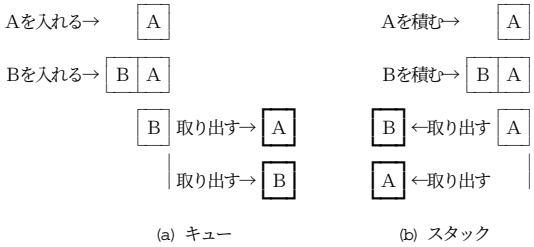


図 8　キューとスタック

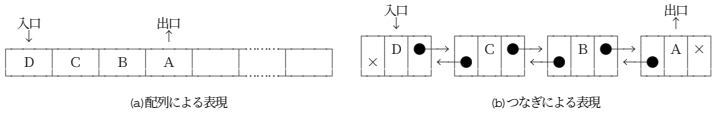
あった．これに対し，参照・追加・削除できるデータの位置を制限したデータ構造が，キューおよびスタックと呼ばれる構造である．

4.1　キュー

図 8（a）の例では，左端が入口，右端が出口である．左からＡを入れ，次にＢを入れると，最初に取り出されるのはＡである．最初に入ったデータが最初に取り出される構造をＦＩＦＯ（First In First Out）またはキュー（Queue）と呼ぶ．この構造は，実世界における「待ち行列」，すなわち，ならんだ順にサービスを受ける構造の表現に適している．

4.2　ス　タック

一方，図 8（b）の例では，出入口は左端に限られる．左からＡを積み，次にＢを積むと，最初に取り出されるのはＢである．このように，最後に入ったデータが最初に取り出される構造をＬＩＦＯ（Last In First Out）またはスタック（Stack）と呼ぶ．データの出入口をならびの一端に制限した構造は，実世界における「寄り道」の表現に適している．Ａさんと話している状態が，スタックにＡを積んだ状態であるとすると，途中でＢさんから緊急の電話がかかってきた状態が，スタックにＢを積んだ状態である．Ｂさんの用件が終わり，スタックからＢを取り除くと，Ａさんと会話していたことを思い出すことができ，Ａさんの用件も終わると，スタックが空になる．なお，特にスタックの場合，データを積むことをプッシュ（Push），取り出すことをポップ（Pop）と呼ぶ．さて，以前に，ならびをコンピュータ上に表現する方法として配列とつなぎを紹介した．キューおよびスタックもならびの一種であり，配列やつなぎにより表現することができる．**図 9** は，キューを配列および双方向つなぎにより表現したものである．



(a)配列による表現　　(b)つなぎによる表現

図 9　キューの表現

4.3　抽象データ型

キューやスタックは，ならびに制限を加えたものであると言っても，配列やつなぎにより表現されている限り，出入口以外のデータの参照や変更を禁止することはできない．具体的には，キューが配列Aにより表現されている場合，A [j] により，出入口以外の任意のデータを読み出すことができる．ところが，キューには，入口にデータを追加（Enqueue）し，出口からデータを取り出す（Dequeue）機能だけがあればよく，内部構造が配列かどうかは本質的な問題ではない．A.Enqueue またはA.Dequeue という，データ名と操作種別の組み合わせにより，キューを使うことができればよい．このように,「データ構造」と「データ構造に対して許される操作」を一体のものとして考える新しい型を「抽象データ型」と呼ぶ．抽象データ型の導入による内部構造の隠蔽は，データ構造を使う側にとっては，内部構造の破壊が防止される利点があり，データ構造を作る側にとっては，ユーザに影響をおよぼすことなく，配列からつなぎに変更するなど内部構造を変更できる利点がある．

5.　木

ならびとは異なり，木はデータ間の親子関係を表現する構造である．ならびの場合，一方向に参照をたどることにより，所属する全てのデータを参照することができたが，木の場合にはデータをたどる手順（走査，Traversal と呼ぶ）が複雑である．ならびと同様，木にも様々な構造があるなかで，コンピュータが最も扱いやすいのは，二分木と呼ばれる構造である．

5.1　二　分　木

二分木は，二つの参照を用いてデータ間の親子関係を表現する構造である．なお，木構造ではデータと参照をまとめて，特にノード（Node）と呼ぶ．**図 10** は，先頭（根，Root と呼ぶ）がA，親Aに子Bおよび子C，親Bに子Dおよび子E，親Cに子Fがある関係を示している．

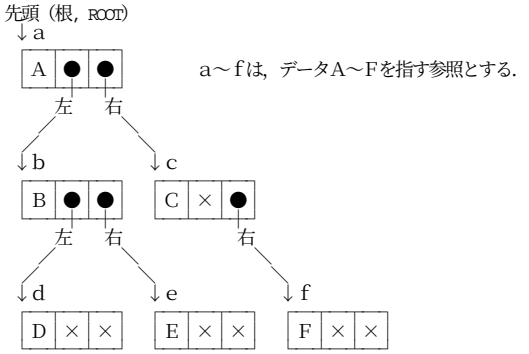


図 10　二分木

二分木に所属するデータをもれなく走査する基本的方法は次の通りである．

(1)　左手が終端でなければ左手へ進む
(2)　左手が終端ならば右手へ進む
(3)　右手が終端ならば親へ戻る

ただし，自ノードのデータをいつ参照するかにより，行きがけ順，通りがけ順，帰りがけ順の三種類に分類される．以下では，行きがけ順について詳述する．

行きがけ順，Pre-Order　自ノードのデータを参照した後に，子ノードに進む方法である．深さ方向への参照が優先されるため，深さ優先走査と呼ばれることもある．さて，親から子に向かっている参照をたどれば，左手や右手に進むことができる．しかし，子から親へ戻る参照はない．親へ戻るにはどうすればよいであろうか．実は，以前に説明したスタックを併用することにより，親へ戻ることができる．

(1)　自ノードのデータを参照する
(2)　左手が終端でなければ現ノード（参照）をプッシュし左手へ進む
(3)　左手が終端ならば右手を調べる
(4)　右手が終端ならばポップした参照へ戻り右手を調べる
(5)　右手が終端かつスタックが空であれば走査終了

図 11 は，図 10 の二分木を行きがけ順に走査する様子である．この場合，データの参照順はＡＢＤＥＣＦとなる．

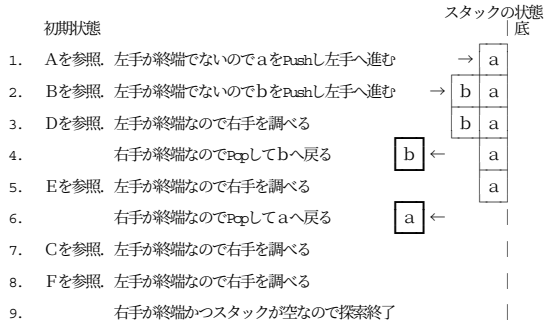


図 11　スタックを用いた二分木の走査

通りがけ順，In-Order　左手につながれた子ノードの走査を終了し，右手に進む前に自ノードのデータを参照する方法である．

帰りがけ順，Post-Order　子ノードの走査を全て終了した後に自ノードのデータを参照する方法である．

5.2　多　分　木

多分木は，三つ以上の参照を用いてデータ間の親子関係を表現する構造であり，**図 12**（a）に示すように，二分木よりも複雑な構造を表現することができる．ただし，ノードの大きさや参照の個数が一定ではないため，データを扱う立場からは，あまり歓迎されない．このため，多分木は，図 12（b）に示すように，親子関係を損なうことなく二分木に変換することが可能であることを知っておくと便利である．変換後の二分木では，左手が親子関係を表し，右手が兄弟関係を表している．

6.　グ　ラ　フ

ならびや木とは異なり，データ間に順序関係が定義できないような構造もある．例えば航空路線図がこの構造にあてはまる．このように，ノードと参照とで構成される一般的な構造をグラフと呼ぶ．データを捜し出す手順はさらに複雑であり，本書では，名前を紹介するにとどめる．

7.　検索（サーチ）

これまでデータ構造の基本を説明してきた．ここからはアルゴリズムについて

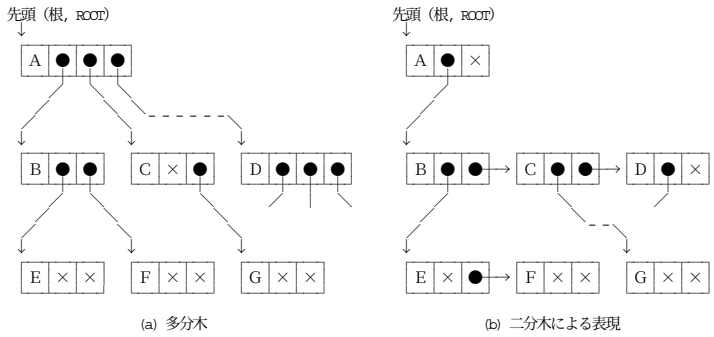


図 12　多分木

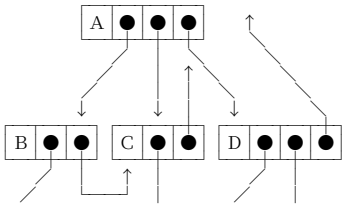


図 13　グラフ

述べていく．まず，データをとりあつかう際に重要なのは，データ構造の中から特定のデータを見つけ出す検索（サーチ，Search）である．以下では配列における検索方法について説明する．

7.1　順　検　索

順検索は，目標とするデータが見つかるまで，先頭から順に比較していく検索方法であり，シーケンシャルサーチ（Sequencial Search）とも呼ばれる．**図 14** のように配列Aに格納されているN個の地名（文字列）に OSAKA があるかどうかを捜す場合，0番目から順に文字列を比較していく．配列内の一つのデータを調べる時間的コストを1とすると，N個のデータから一つを捜すのに要する時間的コストは，最良1，平均N／2，最悪Nである．時間的コストの平均がNに比例する場合「計算量のオーダーがNである」と言い，$O(N)$ と表現する．

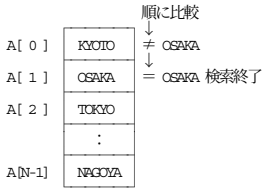


図 14　順検索

7.2　ハッシュ法

ハッシュ法は，同じく配列を使いながら検索時間の大幅削減を図る方法である．順検索では配列番号とデータに関連がないのに対し，ハッシュ法では配列番号自身がデータを捜すヒントとなっている．具体的には，各データはデータ自身から計算できる値（ハッシュ値と呼ぶ）を配列番号とする位置に格納しておく．ハッシュ値の計算（ハッシュ関数と呼ぶ）には基本形や推奨法というものはなく，データの特徴に合わせて，ハッシュ値が一様にちらばるようなハッシュ関数を考案する必要がある．例えば，図 14 について，各文字の ASCII コードを単純に合計した値の最下位８ビットをハッシュ値とし，データを再配置したものが**図 15** である．OSAKA を捜す場合は，まず OSAKA のハッシュ値（111）を求め，A [111] を参照し，内容が OSAKA であるかどうかを検査する．一方，TOKYO のハッシュ値（150）を使ってA [150] を参照しても，そこには KYOTO が格納されているため検索は失敗する．格納するデータがあらかじめ全てわかっている場合には，ハッシュ値が衝突しないハッシュ関数を考える余地があるものの，データの予測は一般的には困難である．したがって，ハッシュ値が衝突した場合の解決方法を用意しておく必要がある．**図 16** に，ハッシュ値を再計算して別の番号を割り当てる方法（クローズドハッシュ）と，同じ番号に複数のデータをつなぐ方法（オープンハッシュ）を示す．さて，ハッシュ法の時間的コストを考えてみよう．ハッシュ値が衝突しなければ，最良１，平均１，最悪１と順検索に比べて極めて高速である．この場合「計算量のオーダーが１である」と言い，$O(1)$ と表現する．一方，図 15 においてハッシュ値が０であるにも関わらず，配列の最後しか空いていないためにＮ－１番に格納されたデータは，検索にＮのコストを要する．データの削除や追加を繰り返した結果，全てのデータがこのような場所に格納された場合，最良Ｎ，平均Ｎ，最悪Ｎと，順検索よりも遅いデータ構造と

文字列	ASCIIコード	合計	下位8bit	10進数
KYOTO	4B 59 4F 54 4F	196(16)	96(16)	150(10)
OSASA	4F 53 41 4B 41	16F(16)	6F(16)	111(10)
TOKYO	54 4F 4B 59 4F	196(16)	96(16)	150(10)
NAGOYA	4E 41 47 4F 59 41	1BF(16)	BF(16)	191(10)

図 15　ハッシュを用いた検索

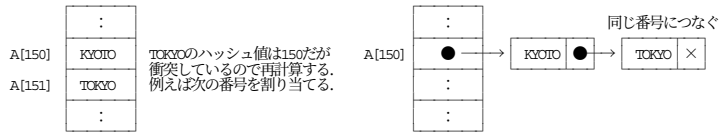


図 16　衝突の解決

なりうることに注意しなければならない．このような状況に陥らないために，クローズドハッシュでは，格納すべきデータ数の二倍以上の大きさの配列を確保することが望ましい．

8. 整列（ソート）

次に，計算量に劇的な違いが現れる整列（ソート，Sort）をとりあげる．N個のデータを整列するためには，N回の大小比較では足りない．N－１回の大小比較により，最も大きなデータを一つ発見することができ，N－２回の大小比較により，残りのN－１個のデータから二番目に大きなデータを発見することができる．この考え方に基づく整列手法をバブルソートと呼び，必要な計算量は次の通りとなる．計算量のオーダーは N^2 であり，$O(N^2)$ と表現する．

$$(N-1)+(N-2)+\cdots+(1)=N\times(N-1)/2$$

Nが４，１６，６４倍になると，計算量は１６，２５６，４０９６倍となり，データ数が多くなるほど計算量が急激に増えることがわかる．ところが，別の考え方に基づくクイックソートを用いると，計算量は $O(NlogN)$ となり，Nが４，１６，６４倍になっても，計算量は８，６４，３８４倍にしかならず，データ量の増加に対する計算量の伸びを劇的に減らすことができる．**図 17** は整列の様子である．各横棒の長さをデータと見なし，大きい順に下から整列することをめざ

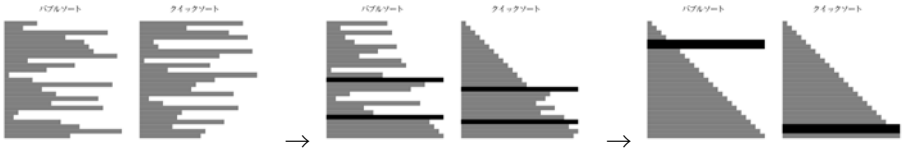


図 17　バブルソートとクイックソート

している．バブルソートが下から順に大きいものを詰めていく間に，部分的な区間の整列を繰り返すクイックソートは，すでに整列を終えようとしている．

8.1　アルゴリズムの記述

以前にとりあげた順検索やハッシュ法は単純なアルゴリズムであったため，言葉による簡潔な説明が可能であった．しかし，複雑なアルゴリズムは，最終的な生産物であるプログラムに近い形式のほうが簡潔かつ正確に記述することができる．以下では，次の基本要素からなる疑似的なプログラミング言語を用いてアルゴリズムを記述する．

定数　整数をそのまま表したもの．

変数　整列対象配列を A[番号]，配列の大きさを N，他の単純変数を I,J,K とする．

演算と代入　四則演算を行い，←により変数へ代入する．⇔は交換とする．

条件分岐　変数間の大小関係に応じた処理を行う．

```
if（大小関係）{ 成立する場合の処理 }
else        { 成立しない場合の処理 }
```

ループ　変数間の大小関係に応じて処理を繰り返す．

```
while（大小関係）{ 成立する場合の処理 }
```

呼び出し　別のプログラムを呼び出す．

```
call プログラム名
```

復帰　元のプログラムへ戻る．

```
return
```

8.2　バブルソート

バブルソートは，隣り合う二つのデータ（A[j] と A[j+1]）を順に比較し，大きい方を下に移動することにより整列を行うアルゴリズムである．

8.3　クイックソート

次に，クイックソートについて説明する．クイックソートは，ある区間のデー

```
BUBBLE_SORT
{
  i ← N-1                    … iの初期値は配列の最終番号
  while (i ≧ 0) {            … iの範囲はN-1〜0
    j ← 0                    … jの初期値は0（配列の先頭から比較）
    while (j < i) {          … jの範囲は0〜i-1
      if (A[j] > A[j+1]) {   … 隣どうしを比較し，
        A[j] ⇔ A[j+1]        … 大きい方を下へ移動
      }
      j ← j+1                … jを1だけ増やす
    }
    i ← i-1                  … iを1だけ減らす
  }
}
```

ルール j

ルール i

図 18　バブルソート

タを基準値以下のグループと基準値以上のグループに分け，各グループにおいて，同じ操作を繰り返すことにより整列を行う手順である．

手順 A.　ならびの中央付近の値を基準値とする．

手順 B.　上から基準値以上，下から基準値以下のデータを捜し，両方そろえばデータを交換する．

手順 C.　手順 B. を繰り返すと，上部に基準値以下，下部に基準値以上のデータが集まる．

手順 D.　上部，下部の各々について手順 A. から D. を繰り返す．

call QUICK_SORT(0,N-1) として，**図 19** を呼び出すことにより，配列 A 全体を整列することができる．なお，最後の部分で，call QUICK_SORT(lo, j) および call QUICK_SORT(i, hi) により，自分自身を呼び出している．このように，自分自身を呼び出して処理を進める方法を「再帰呼び出し」と呼ぶ．さて，計算量について再考してみよう．N が 2 の倍数であり，かつ，再帰呼び出しごとに整列区間がちょうど半分になっていく場合，比較回数は，全体（0 から $N-1$ の範囲）について N 回，$\frac{1}{2}$ の範囲各々について $\frac{N}{2}$ 回，$\frac{1}{4}$ の範囲各々について $\frac{N}{4}$ 回，…，$\frac{1}{N/2}$ の範囲各々について $\frac{N}{N/2}$ 回である．合計すると，以前に説明した通り計算量は $O(NlogN)$ となる．

$$N + (\frac{N}{2} \times 2) + (\frac{N}{4} \times 4) + \cdots + (\frac{N}{N/2} \times (N/2)) = N \times log_2 N$$

```
QUICK_SORT(lo, hi)    与えられた上端（lo）と下端（hi）の間をソートする
{
  i ← lo                                  … iは上端から下方向に変化
  j ← hi                                  … jは下端から上方向に変化
  k ← A[(i+j)/2]                          … 中間位置のデータが基準値k

  while (i ≦ j) {                         … iとjが交差するまで繰り返す
    while ((i < hi) かつ (A[i] < k)) {    … 上から基準値以上の最初のデータを捜す
      i ← i+1
    }
    while ((j > lo) かつ (A[j] > k)) {    … 下から基準値以下の最初のデータを捜す
      j ← j-1
    }
    if (i ≦ j) {                          … 両方見つかったら,
      A[i] ⇔ A[j]                         … データを交換する
      i ← i+1
      j ← j-1
    }
  }

  if (lo < j) {
    call QUICK_SORT(lo, j)                … 基準値以下のみを含む上部分をソート
  }
  if (i < hi) {
    call QUICK_SORT(i, hi)                … 基準値以上のみを含む下部分をソート
  }

  return
}
```

ループ

図 19　クイックソート

9. 課　　題

(1)　一方向つなぎにおいて，先頭および末尾のデータを削除する方法を説明せよ．

(2)　双方向つなぎにおいて，先頭および末尾のデータを削除する方法を説明せよ．

(3)　循環つなぎにおいて，最後のデータを削除する方法を説明せよ．

(4)　通りがけ順により走査する場合のデータ参照順序を示せ．

(5)　帰りがけ順による走査する場合のデータ参照順序を示せ．

(6)　走査方法には，スタックを用いる深さ優先走査の他に，キューを用いる幅優先走査がある．幅優先走査について調べてみよ．

(7)　疑似プログラミング言語を用いて，キューの機能を有する抽象データ型を記述せよ．

(8)　疑似プログラミング言語を用いて，スタックの機能を有する抽象データ型を記述せよ．

(9)　疑似プログラミング言語を用いて，階乗（N!）を計算するためのアルゴリズムを記述せよ．