1-0 アルゴリズムとは

1 人間向きのアルゴリズムとコンピュータ向きのアルゴリズム

問題を解くための論理または手順をアルゴリズム（algorithms：算法）という．問題を解くためのアルゴリズムは複数存在するが，人間向きのアルゴリズムが必ずしも（必ずといってもよいくらい）コンピュータ向きのアルゴリズムにはならない．

たとえば，225と105の最大公約数を求めるには，

```
5)225  105
3) 45   21      Ans=5×3=15
   15    7
```

とする．この方法をコンピュータのアルゴリズムにするのは難しい．というのは，2つの数の約数である5や3を見つけることは，人間の経験的直感によるところが大きいので，これをコンピュータに行わせるとなると，論理が複雑になってしまうのである．

最大公約数を求めるコンピュータ向きのアルゴリズムとして，ユークリッドの互除法という機械的な方法がある．これはこの章の**1-6**で説明する．

2 アルゴリズムの評価

ある問題を解くためのアルゴリズムは複数存在するが，それらの中からよいアルゴリズムを見つけることが大切である．よいアルゴリズムの要件として次のようなものが考えられる．

1. 信頼性が高いこと
   精度のよい，正しい結果が得られなければならない．
2. 処理効率がよいこと
   計算回数が少なくて済み，処理スピードが速くなければならない．計算量の目安としてO記法（big O-notation）を用いる．これについては3章3-0参照．
3. 一般性があること
   特定の状況だけに通用するのではなく，多くの状況においても通用しなければならない．

4. 拡張性があること
   仕様変更に対し簡単に修正が行えなければならない．
5. わかりやすいこと
   誰が見てもわかりやすくなければならない．わかりにくいアルゴリズムはプログラムの保守（メンテナンス）性を阻害する．
6. 移植性（Portability：ポータビリティ）が高いこと
   有用なプログラムは他機種でも使用される可能性が高い．このため，プログラムの移植性を高めておかなければならない．

学問的なアルゴリズムの研究では**1**と**2**に重点が置かれているが，実際的な運用面も考慮すると**3**～**6**も重要である．

3 アルゴリズムとデータ構造

コンピュータを使った処理では多量のデータを扱うことが多い．この場合，取り扱うデータをどのようなデータ構造（data structure）にするかで，問題解決のアルゴリズムが異なってくる．

『Algorithms + Data Structures = Programs（アルゴリズム＋データ構造＝プログラム）』（N. Wirth著）という書名にもなっているように，データ構造とアルゴリズムは密接な関係にあり，よいデータ構造を選ぶことがよいプログラムを作ることにつながる．

データ構造として，リスト，木，グラフなどがあり，**5章**，**6章**，**7章**で詳しく説明する．

3-0 ソートとサーチとは

1 ソート（sort：整列）

ソート（sort）は，データ列をある規則に従って並べ替えることをいう．1，5，11，20…というように小さい順に並べることを昇順（正順）といい，逆に，…20，11，5，1と大きい順に並べることを降順（逆順）という．

コンピュータでは文字列も大きいか小さいか判定でき，それはいわゆる辞書式順に大小関係が決まっている．たとえば，

a＜aaa＜ab＜b…

という順序は，小さい順（辞書順）に並んでいる．

ソートは大きく分けると，

・内部ソート
・外部ソート

になる．コンピュータの主記憶（メモリ）上の配列にデータが与えられていて，これを扱うのが内部ソート，外部記憶装置（ディスク，磁気テープ）上のデータを扱うのが外部ソートである．

本書では内部ソートだけを扱う．内部ソートは**表3.1**に示す6種類に大別できる．

ソートに要する時間は，比較回数と交換回数によりだいたい決まる．この回数は，ソートする数列のデータがどのように並んでいる（正順に近い，逆順に近い，でたらめ）かにより異なる．したがって，ソート時間を一概に論ずることはできないが，基本形と改良形では，データ数が多く（100以上）なったときに圧倒的な差が出る．

数列の長さがn倍になると，基本整列法では所要時間はほぼn^2倍になるのに対し，シェル・ソートでは$n^{1.2}$倍，クイック・ソートやヒープ・ソートでは$n\log_2 n$倍になる．たとえば，nが10^6ならn^2は10^{12}，$n^{1.2} \approx 2 \times 10^7$，$n\log_2 n = 2 \times 10^7$となり，基本形は改良型に比べ50000倍もの差が出る．

計算量のオーダー（order）を表すのに，$O(n^2)$，$O(n\log_2 n)$，$O(n^{1.2})$という表現を用いる．これをビッグO記法（big O-notation）と呼ぶ．$O(n^2)$はデータ数がn倍になれば計算量はn^2倍になることを示している．

	ソート法	特徴	計算量
基本形	基本交換法（バブル・ソート）	隣接する2項を逐次交換する．原理は簡単だが，交換回数が多い．	$O(n^2)$
基本形	基本選択法（直接選択法）	数列から最大（最小）を探すことを繰り返す．比較回数は多いが交換回数が少ない．	
基本形	基本挿入法	整列された部分数列に対し該当項を適切な位置に挿入することを繰り返す．	
改良形	改良交換法（クイック・ソート）→4章 4-6	数列の要素を1つずつ取り出し，それが数列の中で何番目になるか，その位置を求める．	$O(n\log_2 n)$
改良形	改良選択法（ヒープ・ソート）→6章 6-7	数列をヒープ構造（一種の木構造）にしてソートを行う．	
改良形	改良挿入法（シェル・ソート）	数列をとび（gap）のあるいくつかの部分数列に分け，そのおのおのを基本挿入法でソートする．	$O(n^{1.2})$

これらのソート法については『プログラム技法』（二村良彦，オーム社）に，系統的でわかりやすく書かれているので．それを参考にするとよい．

表3.1 代表的な内部ソート法

2 サーチ（search：探索）

大量のデータの中から必要なデータを探し出す作業をサーチ（search：探索）という．サーチの方法は大きく分けて，逐次探索法と2分探索法がある．

データは一般に次のようなレコードで構成されている．

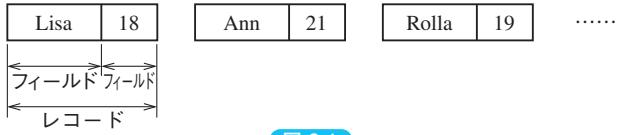


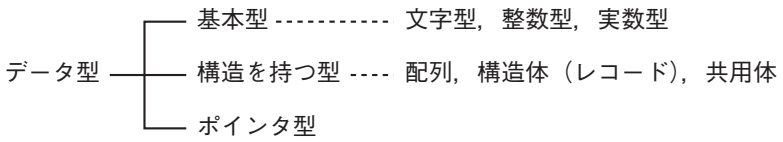
図3.1

ひとかたまりのデータをレコード（record）と呼ぶ．レコードは複数のフィールド（field：項目）で構成される．サーチする項目（フィールド）を特にキー（key）と呼ぶ．

逐次探索法は，表のデータを先頭から順に探索する方法で，アルゴリズムはきわめて単純であり，表がソートされていない場合に用いられる．表の大きさがnの場合，逐次探索法において最も効率よくデータが探索されるのは，目的のデータが先

5-0 データ構造とは

Cの持つデータ型を大きく分類すると次のようになる．



このようなコンピュータ言語が持つデータ型だけでは，大量のデータや複雑なデータを効率よく操作することはできない．そこでデータ群を都合よく組織化するための抽象的なデータ型をデータ構造（data structure）と呼ぶ．

代表的なデータ構造として次のようなものがある．

1. 表（table：テーブル）
2. 棚（stack：スタック）
3. 待ち行列（queue：キュー）
4. リスト（list）
5. 木（tree：トゥリー）
6. グラフ（graph）

これらのデータ構造は，コンピュータ言語が持つデータ型と組合わせてユーザが作る．

表，棚，待ち行列は配列を用いて表現できる．

FORTRAN，従来型BASICには，構造体（レコード）とポインタ型がないので，リスト，木，グラフといったデータ構造を実現するには不向きである．

表は，実社会で最も使われているデータ構造で，次のような成績表を思い浮かべればよい．このような表は配列（2次元配列とは限らない）を用いて簡単に表現できる．

	名前＼科目	国語	社会	数学	理科	英語
1	赤川一郎	75	80	90	73	81
2	岸　伸彦					
3	佐藤健一			75		
4	鈴木太郎					
5	三木良介					90

表 5.1 表（table）

表については，本書では特に説明しない．

スタック，キュー，リストについてはこの章で，木については**6章**で，グラフについては**7章**で説明する．

データ構造とアルゴリズムは密接な関係にあり，よいデータ構造を選ぶことがよいプログラムを作ることにつながる．一口によいデータ構造といってもなかなか難しいが，データの追加，削除，検索が効率よく行えるとか，複雑な構造が簡潔に表現できるといった観点が一つの判断基準となる．

7-0 グラフとは

グラフ（graph）は節点（node，頂点：vertex）を辺（edge，枝：branch）で結んだもので次のように表せる．このグラフの意味は，1→2への道や2→1への道はあるが，1→3への道がないことを示していると考えてもよい．

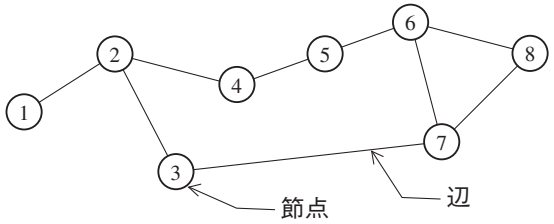


図 7.1 グラフ(graph)

このようなグラフをデータとして表現するためには次のような隣接行列（adjacency matrix）を用いる．

i \ j	1	2	3	4	5	6	7	8
1	0	1	0	0	0	0	0	0
2	1	0	1	1	0	0	0	0
3	0	1	0	0	0	0	1	0
4	0	1	0	0	1	0	0	0
5	0	0	0	1	0	1	0	0
6	0	0	0	0	1	0	1	1
7	0	0	1	0	0	1	0	1
8	0	0	0	0	0	1	1	0

図 7.2 隣接行列

この隣接行列の各要素は，節点i→節点jへの辺がある場合に1，なければ0になる．i=jの場合を0でなく1とする方法もあるが，本書では0とする．

なお，C言語の配列は基底要素が0から始まるので，0の要素は未使用とし，1の要素から使用するものとする．

辺に向きを持たせたものを有向グラフ（directed graph）といい，辺を矢線で表す．先に示した辺に向きのないものを無向グラフ（undirected graph）という．

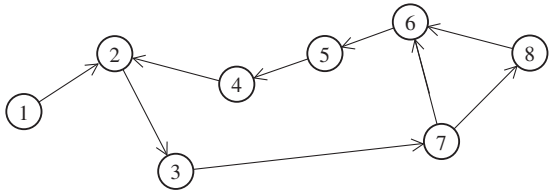


図 7.3 有向グラフ

この場合の隣接行列は次のようになる．有向グラフでは1→2に行けるが，2→1へは行けない．

	1	2	3	4	5	6	7	8
1	0	1	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	1	0
4	0	1	0	0	0	0	0	0
5	0	0	0	1	0	0	0	0
6	0	0	0	0	1	0	0	0
7	0	0	0	0	0	1	0	1
8	0	0	0	0	0	1	0	0

図 7.4 有向グラフの隣接行列

9-1 魔方陣

例題 64 奇数魔方陣

$n \times n$（nは奇数）の正方形のマスの中に$1 \sim n^2$までの数字を各行，各列，対角線のそれぞれの合計が，すべて同じ数になるように並べる．

図9.1に3×3の奇数魔方陣の答を示す．

8	1	6
3	5	7
4	9	2

図 9.1 3×3の奇数魔方陣

数が少ないうちは試行錯誤にマスを埋めていけば答が見つかるが，数が大きくなれば無理である．$n \times n$（$n = 3, 5, 7, 9, \cdots$）の奇数魔方陣を解くアルゴリズムは以下の通りである．

① 第1行の中央に1を入れる．

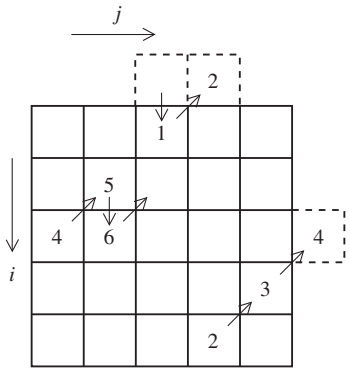


図 9.2 n×nの奇数魔方陣

② 入れる数を方陣の大きさnで割った余りが1であればすぐ下のマスへ進み，そうでなければ斜め上へ進む．

③ もし上へはみ出した場合は，同じ列の一番下へ移る．

④ もし右へはみ出した場合は，同じ行の一番左へ移る．

プログラム Rei64

```
 * --------------------
 *       奇数魔方陣     *
 * --------------------
 */
#include <stdio.h>

#define N 7     // n方陣(n=3,5,7,9,…)

void main(void)
{
    int hojin[N+1][N+1],i,j,k;

    j=(N+1)/2; i=0;
    for (k=1;k<=N*N;k++){
        if ((k%N)==1)
            i++;
        else {
            i--; j++;
        }
        if (i==0)
            i=N;
        if (j>N)
            j=1;
        hojin[i][j]=k;
    }

    printf("        奇数魔方陣 (N=%d)¥n",N);
    for (i=1;i<=N;i++){
        for (j=1;j<=N;j++)
            printf("%4d",hojin[i][j]);
        printf("¥n");
    }
}
```

①

①部は，まとめて次のように書くこともできる．

```
        else {
            i=N-(N-i+1)%N;
            j=j%N+1;
        }
```