計算機プログラミング I (4)

問題解決とアルゴリズム

2002 年 10 月 31 日　増原 英彦

アルゴリズム (algorithm) とは、問題を解くための手順のことである。例えば、掛け算を筆算で行う方法や、逆行列を求めるガウスの消去法などは数学的なアルゴリズムである。(ただし、一般に計算機が扱う問題は数学的なものに限らない。)

一つの問題に対するアルゴリズムは複数あり得る。同じ問題を解く場合でも、アルゴリズムによっては繰返しや計算の回数が大きく違うことがある。プログラムは、アルゴリズムに従って計算機が処理をする手順を書き下したものなので、アルゴリズムの差はプログラムの実行時間や使用するデータ量などに反映することになる。

ある問題に対して、それを解くアルゴリズムがあれば、それに従ってプログラムを書くことはそれほど難しくない。しかし、問題を解くアルゴリズムを見つけること自体は簡単ではない。多くの問題は、より小さな部分問題に分けて考えることができる。また、典型的な問題については定番となるアルゴリズムが知られている。そこで普通は、(1) 問題を小さな問題に分割する (2) 典型的な問題について定番アルゴリズムを当てはめる、といったことをする。もちろん、分割もできずアルゴリズムも知られていない問題もあるので、その場合は知恵を働かせて新しいアルゴリズムを考え出すことになる。

ここではいくつかの例題について、それをどのように分割し、アルゴリズムを見つけるかを見てゆく。

1　最大公約数

問題: 2 つの整数 a, b の最大公約数を求める (ただし $a < b$ とする)

1.1　アルゴリズム 1

この問題は「ある数が a, b の公約数かを調べる」ことと「公約数のうち最大のものを見つける」ことに分けると簡単になる。

アルゴリズム 1(最大公約数): i を $1, 2, \ldots, a$ の順に変化させ、i が a, b の公約数であるかを調べる。最後に公約数であると分かったときの i の値が最大公約数である。 ∎

例えば、$a = 6, b = 21$ の場合であれば、

- $i = 1$:　i は a, b の公約数である
- $i = 2$:　i は a, b の公約数でない
- $i = 3$:　i は a, b の公約数である
- $i = 4$:　i は a, b の公約数でない
- $i = 5$:　i は a, b の公約数でない
- $i = 6$:　i は a, b の公約数でない

となるので、最後に見つけた公約数である 3 が答えとなる。これをプログラムにすると次のようになる:

```
1  public class GCD1 {
2    public static void main(String[] args){
3        int a = 6, b = 21;
4        int lastCommonDivisor = 0; // 一番最近に見つかった公約数
5        for (int i = 1; i <= a; i++) {
6            if ( i は a,b の公約数? ) {
7                lastCommonDivisor = i; // i は公約数なので覚える
8            }
9        }
10       System.out.println("最大公約数は" + lastCommonDivisor);
11   }
12 }
```

変数lastCommonDivisor は、一番最近に公約数となったi の値を覚えておくためのものである。if 文によって、i が公約数であると分かる度に、そのときのi の値が代入される。

for 文は i を 1 からa まで順に変化させる。結果として for 文が終了したときのlastCommonDivisor には、一番最後に公約数となった値、つまり最大公約数が入っていることになる。

分割した残りの問題は次のようにして解くことができる:

- i は a,b の公約数であるとは、i がa の約数、かつ、i がb の約数である。つまり次のような式になる:

  $$\boxed{\texttt{i が a の約数}} \;\texttt{\&\&}\; \boxed{\texttt{i が b の約数}}$$

- i が a の約数であるとは、a をi で割った余りが 0 であること。つまり次のように書ける (a % i はa をi で割った余りを計算する式。==は値が等しいかどうかを比較する演算子):

  ```
  a % i == 0
  ```

練習 4-1: (最大公約数 1) 上の説明を読んで、実際に最大公約数を求めるプログラムを作れ。(資料 web ページから GCD1.java をダウンロードし、修正するとよい。)

1.2 アルゴリズム 2

アルゴリズム 1 の変型として、大きい順に公約数を探すこともできる。

アルゴリズム 2: i を $a, a-1, \ldots, 2, 1$ の順に変化させ、i が a, b の公約数であるかを調べる。最初に公約数であると分かったときの i の値が最大公約数である。 ∎

これをプログラムにすると次のようになる:

```
1  public class GCD2 {
2    public static void main(String[] args){
3        int a = 6, b = 21;
4        int i = a;
5        while (! i は a と b の公約数? ) { //! 式 は 式 の否定
6            i--;                          // i を 1 減らす
7        }
8        System.out.println("最大公約数は" + i + ".");
9    }
10 }
```

このプログラムでは、i が公約数でない間、i を 1 ずつ減らしている。i が公約数となると while 文の条件が false になり、繰り返しが終了する。結果として while 文が終了したときのi は最初に見つかった公約数になる。

練習 4-2: (最大公約数 2) 上のプログラムを完成させよ。(参考: 資料 web ページ GCD2.java)

1.3 ユークリッドの互除法

ここまでのアルゴリズムは単純に 1 つずつ数を調べてゆくものであったが、次の定理を用いるとより少い計算回数で最大公約数を求めることができる。

定理 b を a で割った余りを r とする。a と b の最大公約数と r と a の最大公約数は等しい。

この定理 (証明は省略するが、難しくない) と「r が 0 のとき a は b の約数である」ことに注目すると、以下のようなアルゴリズムが得られる。

アルゴリズム 3(ユークリッドの互除法): b を a で割った余りを r とする。r が 0 でなければ、a, b をそれぞれ r, a に置き換えてその最大公約数を求める。r で 0 であれば、a が (最初の a, b の) 最大公約数。 ∎

例えば、143 と 1469 の最大公約数を求める場合であれば、以下のようにな計算になる:

- $b = 1469$ を $a = 143$ で割った余りは $r = 39$ なので 39, 143 を次の a, b とする
- $b = 143$ を $a = 39$ で割った余りは $r = 26$ なので 26, 39 を次の a, b とする
- $b = 39$ を $a = 26$ で割った余りは $r = 13$ なので 13, 26 を次の a, b とする
- $b = 26$ を $a = 13$ で割った余りは $r = 0$ なので 13 が最大公約数

これをプログラムにすると以下のようになる。

```
1  public class GCD3 {
2    public static void main(String[] args){
3        int a = 143, b = 1469;
4        while ( b を a で割り切れない? ) {
5            次回の a を今回の b/a の余りにする。
             次回の b を今回の a にする。
6
7        }
8        System.out.println("最大公約数は" + a + ".");
9    }
10 }
```

練習 4-3: (最大公約数 3) 上のプログラムを完成させよ。

注意: 5 行目を単純に (1) a=[b/a の余り]; (2) b=a; のように書くと、(2) は今回のa の値ではなく次回のa(つまり今回のb/a の余り) を代入してしまう。正しくは、もう 1 つの変数r を用意して、(1) r=[b/a の余り]; (2) b=a; (3) a=r; のようにしなければいけない。(当然、変数r を宣言する必要もある。)

(参考: 資料 web ページ GCD3.java)

2 素因数分解

問題: 整数 x の素因数分解、つまり $x = p_1 \cdot p_2 \cdots p_k$ となるような素数の組 $p_1, p_2, \ldots p_k$ を求める

この問題は、「素因数を 1 つ見つける」問題と、「素因数を全て見つける」問題に分割することができ、後者は次のようなアルゴリズムで解ける:

アルゴリズム (素因数分解): x が 2 以上のとき、x の素因数を 1 つ見けて p とする。次回の x を今回の x/p として、残りの素因数を見つける。x が 1 になれば全ての素因数が求められたことになる。 ∎

例えば、$x = 84$ の場合は次のようになる。

- $x = 84$ の素因数として 2 が見つかる。次回の x を $84/2 = 42$ とする
- $x = 42$ の素因数として 2 が見つかる。次回の x を $42/2 = 21$ とする

- $x = 21$ の素因数として 3 が見つかる。次回の x を $21/3 = 7$ とする
- $x = 7$ の素因数として 7 が見つかる。次回の x を $7/7 = 1$ とする
- $x = 1$ なので素因数はもうない

これをプログラムにすると次のようになる:

```
1  public class Factorize {
2    public static void main(String[] args){
3        int x = 84;
4        while (x > 1) {
5            int p;
6            x の素因数を 1 つ求め p に代入する
7            System.out.println("素因数:"+p);
8            x = x/p;                          //次回の x を x/p とする
9        }
10   }
11 }
```

x の素因数を 1 つ求め p に代入する 部分は、p を $2, 3, 4, \ldots$ と変化させ、x を割り切る最初のp を見つけることで解ける。つまり最大公約数の「アルゴリズム 2」とほとんど同じ形をしている。

練習 4-4: (素因数分解)　上のプログラムを完成させよ。また、自分の学生証番号を素因数分解してみよ。
(参考: 資料ページ Factorize.java)

3　曜日の計算

問題: 2002 年 10 月 31 日が何曜日であるかを計算する。
(1900 年 1 月 1 日が月曜日であった事実を利用する。)

この問題は次のように分割して解くことができる。

- 1900 年 1 月 1 日から目的の日 (2002 年 10 月 31 日) の間の日数を days 日とする。(両端の日も 1 日と数える。) days を 7 で割った余りが 0 であれば日曜日、1 であれば月曜日、2 であれば火曜日... ... (以下略) である。

```
1  public class Weekday {
2      public static void main(String[] args) {
3          int days = 0;
4          1900 年 1 月 1 日から目的の日までの日数を計算し、days に代入
5          int dayOfWeek = days%7; // 0:日曜日 ..  6:土曜日
6          System.out.println("曜日は"+dayOfWeek);
7      }
8  }
```

- 1900 年 1 月 1 日から目的の日のまでの日数を計算し days に代入 する簡単な方法は、(年, 月, 日) を $(1900, 1, 1), (1900, 1, 2), (1900, 1, 3), \ldots$ のように 1 日ずつ変化させてゆき、目的の日 (2002,10,31) になるまで繰返すことである。プログラムにすると次のようになる:

```
1  loop:                                                    // 4.5 節参照
2      for (int year = 1900; year <= 2002; year++) {
3          for (int month = 1; month <= 12; month++) {
4              int daysOfMonth;
5              year 年 month 月の日数を計算し daysOfMonth に代入
6              for (int date = 1; date <= daysOfMonth; date++) {
7                  days++;                                  // 日数を 1 増やす
8                  if (year 年 month 月 date 日は目的の日か?)
9                      break loop;                          // 4.5 節参照
10             }
11         }
12     }
```

ここでは、1900 年 1 月 1 日から 2002 年 12 月 31 日までの繰返しを行い、途中で目的の日に達したら三重の繰返しを中断する break 文を使っている。(教科書 4.5 節参照)

- year 年 month 月の日数を計算し daysOfMonth に代入するには、月の日数が
  - 1,3,5,7,8,10,12 月は 31 日
  - 4,6,9,11 月は 30 日,
  - 2 月は、閏年でなければ 28 日, 閏年であれば 29 日

  であることから if 文を使って下のように書ける:

```
1  if (month 月は大の月か?)
2      daysOfMonth = 31;
3  else if (month 月は (2 月を除く) 小の月か?)
4      daysOfMonth = 30;
5  else if (year 年は閏年か?)
6      daysOfMonth = 29;
7  else
8      daysOfMonth = 28;
```

練習 4-5: (曜日の計算)　教科書の練習問題 3.6 などを参考にして残りの空欄を埋め、曜日の計算プログラムを完成させよ。また、目的日を自分の生まれた日や、自分の来年の誕生日として曜日を計算せよ。
(参考: 資料ページ Weekday.java)

4　クラスメソッドによる手続きの抽象化

曜日の計算で見たように、複雑な問題は分割して解くことが有効である。しかし、完成したプログラムを見ると、分割した問題がどこに対応するかが分かりにくい。

プログラム言語には、一連の処理をまとめて書き、それに名前を付ける仕組みがある。Java 言語ではメソッドという仕組みがそれである。これらの仕組みを使うことで、次のようなことが可能になる:

- 一まとまりの処理に名前をつけて抽象化する (*cf.* 変数は値に名前をつけて抽象化する機能があった)、
- 分割した問題がプログラムとしても分割して書く、
- 同じ処理が何回も現われる場合に、一回だけ定義をして、複数箇所から共有して使う。

ここでは曜日を計算するプログラムを、クラスメソッドによって分割してゆく方法を見る。
まず、分割する前のプログラムは次のようになっていた:

```
1  public class Weekday {
2      public static void main(String[] args) {
3          int days = 0;
4      loop: //---------------------------------------- (ここから)1900 年 1 月 1 日から
5          for (int year = 1900; year <= 2002; year++) { // 目的の日までの日数を計算し、
6              for (int month = 1; month <= 12; month++) { // days に代入
7                  int daysOfMonth;
8                  year 年 month 月の日数を計算し daysOfMonth に代入
9                  for (int date = 1; date <= daysOfMonth; date++) {
10                     days++;    // 日数を 1 増やす
11                     if ( year 年 month 月 date 日は目的の日か? )
12                         break loop; // 4.5 節参照
13                 }
14             }
15         } //---------------------------------------- (ここまで)
16         int dayOfWeek = days%7; // 0:日曜日 ..  6:土曜日
17         System.out.println("曜日は"+dayOfWeek);
18     }
19 }
```

上のプログラムでは、4–15 行目が 1900 年 1 月 1 日から目的の日までの日数を計算 する部分問題になっている。この部分をクラスメソッドを定義して分割したのが下である:

```
1  public class Weekday {
2      public static void main(String[] args) {
3          int days = 0;
4          days = Weekday.countDays();       // クラスメソッドの呼出し
5          int dayOfWeek = days%7; // 0:日曜日 ..  6:土曜日
6          System.out.println("曜日は"+dayOfWeek);
7      }
8      /** 1900 年 1 月 1 日から目的の日までの日数を計算 */
9      public static int countDays() {
10         int d = 0;            // 日数
11     loop:
12         for (int year = 1900; year <= 2002; year++) {
13             for (int month = 1; month <= 12; month++) {
14                 int daysOfMonth;
15                 year 年 month 月の日数を計算し daysOfMonth に代入
16                 for (int date = 1; date <= daysOfMonth; date++) {
17                     d++;       // 日数を 1 増やす
18                     if ( year 年 month 月 date 日は目的の日か? )
19                         break loop;
20                 }
21             }
22         }
23         return d;             // 計算した日数を返す
24     }
25 }
```

下のプログラムの 9–24 行がcountDays という名前のクラスメソッドを定義している。9 行目のpublic static int countDays() は、定義されるメソッドの性質を決めている:

- static — クラスメソッドである
- int — 返り値が整数 (int) 型である
- countDays — メソッドの名前
- () — 引数が無い

4 行目は、定義したクラスメソッドを呼出している。このプログラムのクラス名が (1 行目にあるように)Weekday であるので、Weekday.countDays() のように書く。

11–22 行は、上のプログラムの 4–15 行とほとんど同じである。違うのは次の 2 点である:

- 日数を数えるのに別の変数d を宣言して使っている。これは、メソッドの中で宣言された変数を他のメソッドから使うことができないためである。つまり、3 行目で宣言された変数days はメソッドcountDays の中で使うことができないので、独自に変数を宣言して使っている。
- 23 行目の return 文で数えた日数を返している。つまり、この文が実行されたときの d の値が、4 行目のWeekday.countDays() の式の値となる。

同様にして year 年 month 月の日数を計算し daysOfMonth に代入 する部分もクラスメソッドによって分けることができる:

```
1  public class Weekday {
2      略
3      /** 1900 年 1 月 1 日から目的の日までの日数を計算 */
4      public static int countDays() {
5          int d = 0;              // 日数
6      loop:
7          for (int year = 1900; year <= 2002; year++) {
8              for (int month = 1; month <= 12; month++) {
9                  int daysOfMonth;
10                 daysOfMonth = daysOfMonth(year,month);
11                 for (int date = 1; date <= daysOfMonth; date++) {
12                     d++;         // 日数を 1 増やす
13                     if ( year 年 month 月 date 日は目的の日か? )
14                         break loop;
15                 }
16             }
17         }
18         return d;               // 計算した日数を返す
19     }
20     /** y 年 m 月の日数 */
21     public static int daysOfMonth(int y, int m) {
22         int days;
23         if ( m 月は大の月か? )
24             days = 31;          // 大の月
25         else if ( m 月は (2 月を除く) 小の月か? )
26             days = 30;          // 小の月
27         else if ( y 年は閏年か? )
28             days = 29;          // 閏年の 2 月
29         else
30             days = 28;          // 平年の 2 月
31         return days;            // 月の日数を返す
32     }
33 }
```

ここではどの月の日数を計算したいかを伝えるために引数を使っている。10 行目のメソッド呼出し

```
daysOfMonth(year,month)
```

では、引数として変数year, month のその時点の値をメソッドdaysOfMonth に渡している。

渡された引数は、呼出されたメソッドdaysOfMonth 側では、変数にしまわれた値として使う。21 行目の宣言

```
public static int daysOfMonth(int y, int m)
```

の(int y, int m) という部分が、以下のことを決めている:

- 引数は 2 つでなければいけなくて、
- 1 つ目の引数は整数 (int) 型でなければいけなくて、その値は変数y にしまうことを、
- 2 つ目の引数は整数 (int) 型でなければいけなくて、その値は変数m にしまう。

引数として渡されるのは、式を計算した結果の値であることに注意しよう。このプログラムでは一見、呼出した側の変数year, month が呼出された側の変数y, m に変わったように見えるが、これらの変数は全く別物である。そのため、もしメソッドdaysOfMonth の中でy やm の値を代入によって変更したとしても、呼出した側のyear, month に入っている値は変化しない。

同様にして、[y 年は閏年か?]の部分もメソッドとして定義できる:

```
1  public class Weekday {
2      [略]
3      /** y 年 m 月の日数 */
4      public static int daysOfMonth(int y, int m) {
5          int days;
6          if ([m 月は大の月か?])
7              days = 31;          // 大の月
8          else if ([m 月は (2 月を除く) 小の月か?])
9              days = 30;          // 小の月
10         else if (Weekday.isLeapYear(y))
11             days = 29;          // 閏年の 2 月
12         else
13             days = 28;          // 平年の 2 月
14         return days;            // 月の日数を返す
15     }
16     /** y 年は閏年か? */
17     public static boolean isLeapYear(int y) {
18         return (y%400 == 0 || (y%100 != 0 && y%4 == 0));
19     }
20 }
```

この場合「閏年かどうか」という計算結果の値は true か false という値であるので、メソッドの返り値の型は 17 行目のように boolean と宣言する。

練習 **4-6:** (クラスメソッドの定義版)　残りの空欄を埋め、クラスメソッドを用いた曜日の計算プログラムを完成させよ。(参考: 資料 web ページ Weekday.java(クラスメソッド版))

練習 **4-7:** (クラスメソッドの引数)　例では、「目的の日付」がプログラムの中の何箇所かに現われていた。そのため、別の日の曜日を計算するようにプログラムを変更することは簡単ではなかった。以下のように曜日 (の番号) を計算するメソッドを定義するようにプログラムを修正せよ。

```
1  public class Weekday {
2      public static void main(String[] args) {
3          int day = Weekday.dayOfWeek(2002,10,31);
4          System.out.println("曜日は"+day);
5      }
6      /** goalYear 年 goalMonth 月 goalDate 日の曜日を 0-6 の数で計算する
7          (ただし 0:日曜日 ..  6:土曜日) */
8      public static int dayOfWeek(int goalYear, int goalMonth, int goalDate) {
9          [1900 年 1 月 1 日から goalYear 年 goalMonth 月 goalDate 日までの日数を数え 7 で
           割った余りを返す]
10
11     }
12     [他のクラスメソッドの定義 (略)]
13 }
```

練習 **4-8:** (値を返さないクラスメソッド)　教科書のプログラム T45.java では、「8 角形を描く」部分と「5 角形を描く」部分が混在していた。5 角形を描く部分をクラスメソッドとして分けて定義し、それを 8 角形を描く部分から呼出すように変更せよ。

このメソッドは亀を移動させるだけで、計算結果を返さない。そのようなメソッドは、返り値の型がvoid(つまり「無し」) であるものとして以下のように定義する。また、値を返さないので return 文を書く必要はない。

```
1  public static void drawPentagon(Turtle m) {
2      [略]
3  }
```