計算機プログラミング I (6)

アルゴリズムと計算量

2002 年 11 月 14 日　増原 英彦

プログラムが問題を解くのにかかるおよその時間を予測することは重要である。例えばデータを 100 個扱う問題を、あるプログラムが 1 秒で解いたとしよう。データが 1,000 個に増えたとき、そのプログラムの実行時間は何秒になるだろうか? この答えは、そのプログラムの性質によって異なり、10 秒かも知れないし、1 万秒かも知れない。

実際のプログラムの実行時間は CPU の性能などの要因で変わるため、正確に予測することは難しい。しかし、およその計算時間は「計算の回数」に比例している。計算の回数はアルゴリズムの性質を考えれば分かるので、問題の大きさが n 倍になったときに、およその時間が n 倍、n^2 倍、あるいは 2^n 倍になるのか、といった見積りをすることは可能である。

このように、アルゴリズムから見積られたな大まかな計算時間を**計算量**という。

ここではアルゴリズムによって計算量が変わることを見るため、まず「ある数の平方根になっている値を見つける」という探索アルゴリズムを 2 つ紹介し、それの計算量を議論する。

1　探索

問題:　ある数 x の平方根を δ 以内の誤差で求める。(ただし、ライブラリを使わずに求める。また、δ は 0.000001 のような小さな数とする。)

1.1　線形探索

x の平方根は「2 乗すると x になる数」なので、小さい順に数を調べてゆき、そのような数を見つければよい。このようなアルゴリズムは線形探索 (linear search) と呼ばれる。

アルゴリズム 1(線形探索):　$a = 0, \delta, 2\delta, 3\delta, \ldots$ のように変化させながら、 $a \cdot a$ と x を比較する。$x \leq a \cdot a$ となるような最小の a を見つければその a が x の平方根 (の近似値) 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 Sqrt {
2      public static void main(String[] args){
3          System.out.println(Sqrt.linear(2));
4      }
5      /** 線形探索により x の平方根を探す */
6      public static double linear(double x) {
7          double a = 0;          // 平方根の推定値
8          while (a の自乗が x 未満か) {
9              a をδだけ増やす
10         }
11         return a;
12     }
13 }
```

平方根 (の近似値) は小数であるので、変数は`double` 型を用いてることに注意せよ。また、ここでは「平方根の計算」をしている部分をクラスメソッドとして定義している。

練習 6-1: (線形探索)*　上のプログラムを完成させよ。

練習 6-2: (大きな数の平方根)　さらにプログラムを拡張して

- 1 から 100 までの数の平方根をそれぞれ計算・表示させる場合
- 学生証番号の平方根を計算する場合
- (学生証番号の 100 倍) の平方根を計算する場合

にかかる時間を調べよ。

1.2 二分探索

大きな数 (例えば 1 億) の平方根を探す際に、小さな a (例えば 1 付近) を高い精度で探すのは無駄であろう。直感的には、「初めはおおまかに探し、序々に細く探してゆく」方が効率的であると思われる。

これをアルゴリズムにしたものの 1 つは「探す範囲を半分ずつに狭めてゆく」やり方で、**二分探索**(binary search) と呼ばれている。

例えば 2 の平方根を求めるのに、最初は区間 (0,2) に $\sqrt{2}$ があることが分かっている。区間の中央値 $(0+2)/2 = 1$ は $1 \cdot 1 < 2$ なので $\sqrt{2}$ は 1 よりも大きい。つまり、 区間 (1,2) に $\sqrt{2}$ があることが分かる。このような処理を繰り返してゆくと区間は次の表のように狭くなってゆく:

回数	区間	中央値	中央値の 2 乗(大小)	次の区間
1	(0 ,2)	1.0	1 (<2)	(1 ,2)
2	(1 ,2)	1.5	2.25 (>2)	(1.5 ,2)
3	(1.5 ,2)	1.25	1.5625 (<2)	(1.25 ,1.5)
4	(1.25 ,1.5)	1.375	1.890625 (<2)	(1.375,1.5)
5	(1.375,1.5)	1.4375	2.06640625 (>2)	(1.375,1.4375)
		⋮		

このようにしてゆけば、区間の幅はいつか δ と以下となるので、そのときの区間の上限 (あるいは下限) が求める平方根の近似となる。

アルゴリズム 2(二分法): 区間の下限・上限を表わす変数 min, max を用意し

- はじめは min=0, max=x とする
- 中央値 mid = (min+max)/2 の 2 乗と x を比較し、
  - $mid^2 < x$ のときは次回の min をいまの mid にする
  - $mid^2 > x$ のときは次回の max をいまの mid にする

  この処理を区間の幅 $max - min$ が δ 以下になるまで繰返す。
- 区間の幅 $max - min$ が δ 以下になったときの max が $\sqrt{x}$ の近似値である。(min でもよい)

これをプログラムにすると次のようになる。

```
1  /** 二分法によって平方根を求める */
2  public static double binary(double x) {
3    double min = 0, max = x; // 区間の下限・上限
4    while (区間の幅はδ以上か) {
5      if (区間の中央値の 2 乗が x 未満か) {
6        次の区間の下限を今の中央値にする
7      } else {
8        次の区間の上限を今の中央値にする
9      }
10   }
11   return max;
12 }
```

練習 6-3: (二分法)* 練習 6-1 のプログラムにならって上のメソッド`binary`を用いて平方根を計算するプログラムを作れ。

練習 6-4: (線形探索と二分法の比較) 線形探索・二分法の 2 つのメソッドを使った場合に

- 答えが誤差の範囲で一致していることを確めよ
- 大きな数の平方根を求める場合の計算時間を比較せよ

練習 6-5: (他の関数) 二分法は平方根だけでなく、単調で逆関数が分かっているような関数であればそのまま適用できる。練習 6-3 のプログラムをもとに、逆数 $(1/x)$ や 3 乗根 $\sqrt[3]{x}$ を求めるプログラムを作れ。

2 計算量

線形探索と二分法では、数が大きくなるにつれ、また精度を高くするにつれ計算時間が大きく違ってくる。このことは、2 つのアルゴリズムの**計算量**を考えることで明らかにできる。

定義 (計算量): n を問題の大きさとする。あるプログラムが大きさ n の問題を解くのにかかる時間のうち、n に関係する部分を (時間) 計算量と呼び $O(f(n))$ のように書く。 ∎

例:

- n 回の繰返しで、1〜n までの和を求めるプログラムの計算量は $O(n)$ である。
- n 回の繰返しで、1〜n までの積を求めるプログラムの計算量も $O(n)$ である。足し算と掛け算では計算時間が違うかも知れないが、それは n とは関係ない定数項なので無視される。
- n が素数かどうかを判定する単純なアルゴリズム、つまり、2〜n で n が割り切れるかどうかを調べるアルゴリズムの計算量は $O(n)$ である。繰返しの中で、剰余を求めたり条件分岐を行っているが、それらは全て n に依らない定数項なので無視される。

  (また、n が素数でない場合は繰返しの回数は n 回未満になるので、厳密には素数が存在する割合等も考える必要がある。ここでは素数は一定の割合で存在すると考え、平均すれば n の定数倍の計算時間となるとしている。)
- 1〜n の中に素数がいくつあるかを数える単純なアルゴリズム、つまり、個々の数が素数かどうかを判定するのに上のような方法をとる場合の計算量は $O(n^2)$ である。

問題の大きさ 平方根を求めるプログラムの問題の大きさには、平方根を求める数 x と許容誤差 δ の 2 つがあった。ここでは簡単のために許容誤差は一定だとして x に対する計算量を考えることにする。

(許容誤差は小さくなるほど問題が大きくなるものと言える。これも含めて計算量を考える場合には、x に加えて $d = 1/\delta$ のような「精度」を使って問題の大きさを表わし、x と d に対する計算時間の変化を見積ればよい。)

平方根の計算量 線形法では、x の平方根を計算するのに $\sqrt{x}/\delta$ 回の繰返しをする。δ は問題の大きさと無関係としているので計算量は $O(\sqrt{x})$ になる。つまり、このアルゴリズムでは 10 倍大きな数の平方根を求めるのに、およそ $\sqrt{10}$ 倍の時間がかかることが分かる。

一方、二分法の計算回数はどの程度なのだろうか? 解を探す区間は、最初 $(0, x)$ なので幅は x である。繰返しを一回行うたびに区間の幅は半分ずつになってゆくので、n 回目の繰返しにおける区間の幅は $x/2^n$ となる。従って、繰返しが終るのは $x/2^n < \delta$ となったときである。両辺を 2^n 倍して対数をとると $\log_2(x/\delta) < n$ なので、およそ $\log_2(x/\delta) = \log_2 x - \log_2 \delta$ 回の繰返しが行われることになる。問題の大きさは x としているので、対数の底や δ の項を無視することができ、計算量は $O(\log x)$ となる。

両者の計算を具体的な x で比較すると次のようになる:

x	10^0	10^1	10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9	10^{10}
$\sqrt{x}$	1.0	3.2	10.0	31.6	100.0	316.2	1000.0	3162.3	10000.0	31622.8	100000.0
$\log x$	0.0	2.3	4.6	6.9	9.2	11.5	13.8	16.1	18.4	20.7	23.0

繰返し一回あたりの計算は二分法の方が多いが、大きな x になると繰返しの回数に大きな差が生じるので、繰返し一回あたりの計算は問題にならないことが分かる。

練習 6-6: (計算量と計算時間) 線形探索と二分法で平方根を計算するプログラを、いくつかの x で計算したときの時間をそれぞれ測れ。上で議論した計算量の関係になっているかを確認せよ。

練習 6-7: (許容誤差と計算量) 線形法の繰返し回数は $\sqrt{x}/\delta$ である。従って精度 $d = 1/\delta$ を含めた計算量は $O(d\sqrt{x})$ となる。d を含めた二分法の計算量はいくらか。

練習 6-8: (冪乗) x^n の計算を考える。単純には、1 を n 回 x 倍すればよいので、次のようなアルゴリズム (プログラム) によって求められる。

```
1  public static int power(int x, int n) {
2    int e = 1;
3    while (0 < n) {
4      e = e * x;
5      n = n - 1;
6    }
7    return e;
8  }
```

別のアルゴリズムもある。$x^{2n} = (x^2)^n, \quad x^{2n+1} = x \cdot x^{2n}$ であるので

$$x^n = \begin{cases} (x^2)^{n/2} & n \text{ は偶数} \\ x(x^2)^{n/2} & n \text{ は奇数} \end{cases}$$

である。($n/2$ は切りすてだとする。) これをもとにすると、以下の手順で x^n が計算できる (プログラムは上図右):

- 初めは冪数 e=1 とする
- n が奇数ならば e を x 倍する
- n を半分にし、x を 2 乗して、n が 0 になるまで繰り返す。(つまり、e に「x の 2 乗」の「n/2」乗を掛ける。)

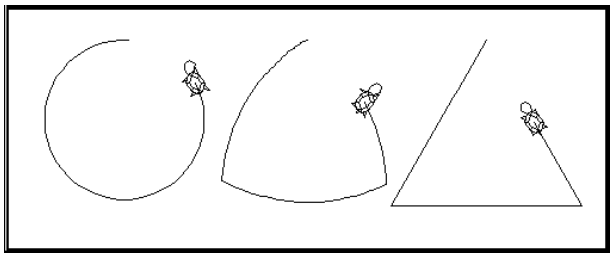
```
1  public static int power(int x, int n) {
2    int e = 1;
3    while (0 < n) {
4      if (n%2 == 1) { // n が奇数の場合
5        e = e * x;   // e を x 倍する
6      }
7      x = x*x;       // 「x の 2 乗」の
8      n = n/2;       // 「n/2」乗を求める
9    }
10   return e;
11 }
```

2 つのアルゴリズムが正しく冪乗を計算することを確め、この 2 つのアルゴリズムの計算量を求めよ。

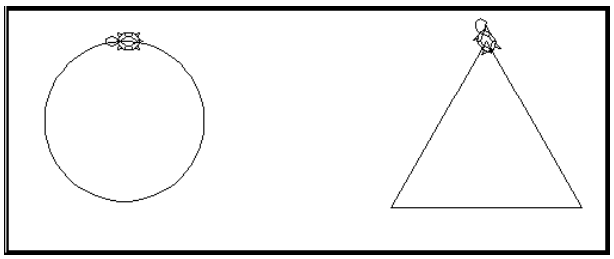
第 1 回課題: Morphing

- 期限: 12 月 4 日 (水)
- 提出方法: 以下のどちらか
  - HTML 形式で作ったファイルをスクリプト実行によって提出 (詳細は追って通知する)、あるいは
  - 紙に書かれたものをレポートボックスへ提出
- 提出物: 各項目について、課題の考察・解いた方法の説明、コンパイル・実行可能なプログラム。
- 注意事項:
  - 常識の範囲を越えた類似部分のあるレポートがあった場合は、追加の面接試験を行う場合や、当該レポートの評価を 0 点にすることがある。
  - 全ての項目を提出しなくてもよい。
  - 複数の項目をまとめて解いた場合には、分かるように明記すること。
  - 提出物の中心は考察および解いた方法の説明である。(プログラムは説明が正しいことの証明に過ぎないので、それだけを提出してはならない。)
  - レポートの読みやすさ・独自性も採点の対象である。

Morphing とは図形を 1 つの形から別の形へ滑らかに変化させることである。ここでは簡単化して、次の図のように円と正三角の「中間」にあたる図形を描くことを考える。



a. 円と正三角形　タートルグラフィクスによって次の図のような円と正三角を描け。ただし、円と正三角形の上端の位置はそれぞれ $(100, 25)$, $(400, 25)$ で、円および正三角形の高さ $\ell = 150$ とする。



ヒント: タートルグラフィクスで正確な円を描くことはできないので、円は正 N 角形で近似すればよい ($N = 60$ 程度)。また、T41.java は正五角形を描くプログラムである。一回あたりにタートルが進む距離は、円周 $\pi\ell$ を N に分けているので $\pi\ell/N$ とすればよい。

また、高さ ℓ の正三角の一辺の長さは $2\sqrt{3}\ell/3$ である。

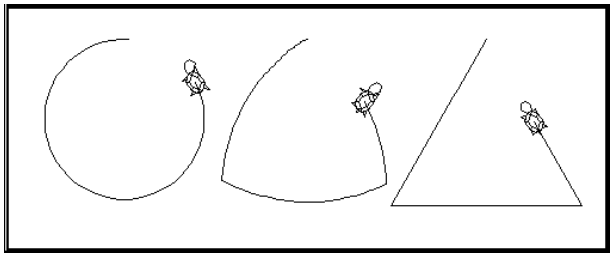
b. 並行に作図　円と正三角形を並行して描け。つまり、円・正三角形の周をそれぞれ N 本の直線に分割し、円のための直線と正三角のための直線が 1 本ずつ交互に描かれるようにせよ。
ヒント: 正三角形を N 本の直線に分けて描くには、タートル m_T を適切な位置に 330 度 (あるいは 30 度) を向けて作り、「m_T を $2\sqrt{3}\ell/N$ 進める処理」を N 回繰返す。そして、$0, N/3, 2N/3$ 回目には進む前に 120 度回転させればよい。

円と正三角形を並行して描くには、次のようにすればよい:

- 円と正三角形を描くタートル m_C, m_T を適切な場所に作る
- 「m_C を進め回転させる、m_T を (必要であれば回転させ) 進める」処理を N 回繰返す

c. 中間の図形　次の図のように円と正三角形の中間の図形を描け。(図は描いている途中の状態である。) 中間の図形とは次のようなものである:

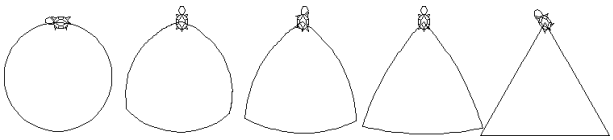
図形を時刻 $0 \leq t \leq 1$ の間に移動する点と考え、点の座標を $x(t), y(t)$ のように表わすとする。いま、円と正三角の図形はそれぞれ $x_C(t), y_C(t)$ および $x_T(t), y_T(t)$ だとする。このとき、中間の図形は $x_M(t) = (x_C(t) + x_T(t))/2,\ y_M(t) = (y_C(t) + y_T(t))/2$ である。



ヒント: b. のように円と正三角形を描く際に、m_C と m_T の中間点をたどってゆくと、「中間」にあたる図形を描くことができる。そこでタートル m_C, m_T に加えてタートル m_M を作り、m_C, m_T が動く度に、m_M をその中間の位置 (X, Y 座標がそれぞれ 2 つのタートルの座標の平均値になっている場所) に移動させればよい。

なお、タートル `m` の現在位置の X, Y 座標は `m.getX()`, `m.getY()` というメソッド呼出しで得られる。また `m` を位置 (x, y) へ移動させるには `m.moveTo(`x, y, a`)` というメソッド呼出しを行えばよい。(ただし a は角度。)

d. 沢山の中間の図形　複数の段階の「中間」にあたる図形を描け。次の図は 3 段階の例である。



ヒント: m_C, m_T に加えて、$k-1$ 個のタートル $m_1, m_2, \ldots, m_{k-1}$ を作る。m_C, m_T を移動させる度に、その X 座標 x_C, x_T を調べ、m_j を X 座標が $((k-j)x_C + jx_T)/k$ (Y 座標も同様) となるように移動させればよい。

e. 様々な中間の図形　両端の図形として、円や正三角形のかわりに星形や文字などを描くようにして、それらの「中間」にあたる図形を作成するプログラムを作れ。