第 6 章 問題の解き方

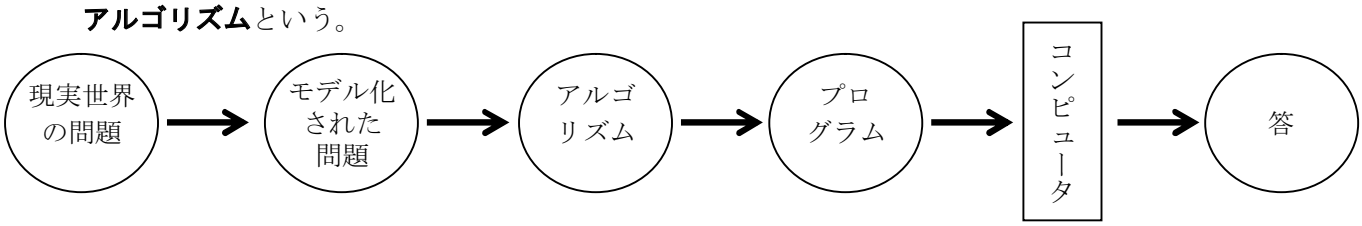
Ⅰ. アルゴリズム

1. アルゴリズムの役割

コンピュータによって問題を解くというのは、人間が書いたプログラムをコンピュータが実行することである。現実世界の問題からプログラムを作る過程を詳しく見ると、

1. 問題をモデル化する
2. モデル化された問題に対して、それを解く計算手順を考える
3. 手順通りに計算するプログラムを作る

という 3 との段階に分かれる。ここで、プログラムになる前の段階(1.～2.)の計算手順のことを、**アルゴリズム**という。



アルゴリズムの選択次第で、プログラムの実行時間は長くも短くもなる。なるべく速く適切に問題の解決できるアルゴリズムを選ぶことが重要である。

2. アルゴリズムの実例

例題
$x\,(>0)$ が与えられたとき、精度 δ で $\sqrt{x}$ を求めよ。
すなわち、$\sqrt{x}-\delta<y<\sqrt{x}+\delta$ の範囲の y を 1 つ求めよ。

一. 反復法

まず考えられるのが、「y をちょっとずつ大きくしていって丁度良いところで止める」という方法である。この場合、y を 0 から δ ずつ大きくしていって、$\sqrt{x}$ を超える寸前に止めればよい。
これを前章でならった記法で表すと、次のようになる。

$y \leftarrow 0$	y を 0 にする。
while $(y+\delta)^2 < x$ **do**	$(y+\delta)^2$ が x を超えない限り ($y+\delta$ が $\sqrt{x}$ を超えない限り)、
$y \leftarrow y+\delta$	y に δ を加える
done	という処理を繰り返す。
return y	それが終わったら ($y+\delta$ が $\sqrt{x}$ を超えてしまったら)、答えを y として計算終了。(**return** はそういう意味)

このようにすると、例えば x が 2 で δ が 0.0001 の時、

0 回目：$y+\delta=0.0001$ … まだ超えていない　$((y+\delta)^2=0.00000001)$
1 回目：$y+\delta=0.0002$ … まだ超えていない　$((y+\delta)^2=0.00000004)$
⋮
14141 回目：$y+\delta=1.4142$ … まだ超えていない　$((y+\delta)^2=1.99996164)$
14142 回目：$y+\delta=1.4143$ … 超えた！　$((y+\delta)^2=2.00024449)$

といった具合に、計算を 14142 回繰り返さないと答えに辿り着けない。この方法では、精度を 1 桁(10 倍)上げると、10 倍の時間がかかってしまう。

そこで、より良いアルゴリズムとして、二分法というのが考えられる。

二．二分法

今度は、答えの存在しうる区間 (a, b) を徐々に半分ずつにしていく方法を考える。具体的には、「答えが区間 (a, b) にあるとする。答えがその下半分$\left(a, \frac{a+b}{2}\right)$にあるか、または上半分$\left(\frac{a+b}{2}, b\right)$にあるかを判定し、次はその半分だけを調べる。区間が十分に(δ よりも)狭くなったら終了。」という方法である。この場合、はじめは区間 $(0, x)$ を考え、毎回その中点の 2 乗を調べればよい。これを前章でならった記法で表すと、次のようになる。

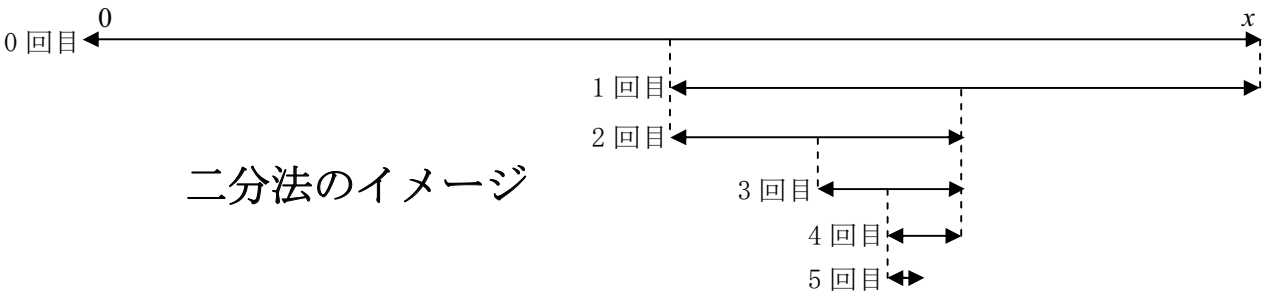
$a \leftarrow 0$	a を 0 にする。
$b \leftarrow x$	b を x にする。
while $b - a > \delta$ **do**	区間 (a, b) が δ よりも広いとき、
$c \leftarrow \frac{a+b}{2}$	区間 (a, b) の中点 $\frac{a+b}{2}$ を記号 c で表す。
if $c^2 > x$	もし c が $\sqrt{x}$ よりも大きかったら、
then $b \leftarrow c$	b を c にする(次回は区間 (a, c) が考察の対象)。
else $a \leftarrow c$	そうでなければ(c が $\sqrt{x}$ 以下だったら)
endif	a を c にする(次回は区間 (c, b) が考察の対象)。
done	という処理を繰り返す。
return y	それが終わったら(区間 (a, b) が δ 以下の狭さになったら)、答えを a として計算終了。

この計算過程では、常に $a \le \sqrt{x} < b$ が保たれており、最終的に $b - a \le \delta$ となった時の a がちゃんと答えになっている。

このようにすると、例えば x が 2 で δ が 0.0001 の時、

0 回目：$(a, b) = (0, 2)$ … まだ広い　($b - a = 2$)
1 回目：$(a, b) = (1, 2)$ … まだ広い　($b - a = 1$)
2 回目：$(a, b) = (1, 1.5)$ … まだ広い　($b - a = 0.5$)
3 回目：$(a, b) = (1.25, 1.5)$ … まだ広い　($b - a = 0.25$)
⋮
14 回目：$(a, b) = (1.4141845703125, 1.414306640625)$ … まだ広い　($b - a = 0.0001220703125$)
15 回目：$(a, b) = (1.4141845703125, 1.414245560546875)$ … 十分狭い！($b - a = 0.00006103515625$)

といった具合に、反復法では 14142 回繰り返していた計算がたったの 15 回で済む。この方法では、精度を 2 倍に上げる毎に計算回数が 1 回増える(つまり回数が精度の対数に比例する)。



二分法のイメージ

3．計算量

問題解決の上で、その時間が重要な問題となる。例えば、翌日の天気を予報するのに 1 週間かかってしまっては意味がない。実際、同じ問題を解く場合でも、良いアルゴリズムでは半日で済むものが悪いアルゴリズムだと 100 年以上かかるという場合もある。

前にも述べたとおり、アルゴリズムによって**計算量**は大きく左右される。この計算量が時間の目安となる。そこで、大雑把な**計算量のオーダー**を考える。例えば、処理すべきデータ数 N に対して、計算量が「N に比例する」「log$N$ に比例する」などという具合である。

先程の反復法による $\sqrt{x}$ の計算では、$\frac{\sqrt{x}}{\delta}$ 回の繰り返しが行われるので、計算量のオーダーは $\frac{\sqrt{x}}{\delta}$ である。一方、二分法による計算では、$\frac{x}{2^n} < \delta$ となる最小の n の数だけ計算が行われるので、オーダーは $\log\frac{x}{\delta}$ である。こうして見ると、一般に二分法の方が効率が良いことがわかる。

1 回の計算に 1 ナノ秒かかるとしたときの、オーダーの影響を表したものが次の表である。ここから、オーダーが計算量・計算時間に大きな影響を与えることが見てとれる。
（ここでは log は常用対数の値を示している。）

n	1	10	100	1,000	10,000	100,000	10^6	10^7
log$n$	—	1 ns	2 ns	3 ns	4 ns	5 ns	6 ns	7 ns
n	1 ns	10 ns	100 ns	1 μs	10 μs	100 μs	1 ms	10 ms
n^2	1 ns	100 ns	10 μs	1 ms	100 ms	10 s	16.7 min	27.8 h
n^3	1 ns	1 μs	1 ms	1 s	16.7 min	11.6 日	31.7 年	3 万年
$2^{n/2}$	1.4 ns	32 ns	13.0 日	10^{134} 年				

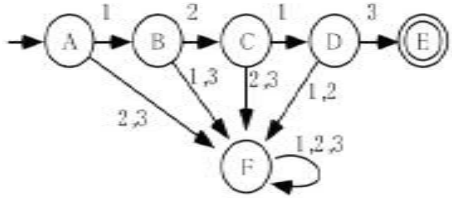
（教科書 p. 136 の表は最下行に誤りがあります）

II．計算のモデル化

有限状態機械のモデル

内部に状態を持ち、その状態と外界から受け取った信号を元に次の状態と出力を決める電子回路のモデルを考える。状態を持ち、入出力が行える機械を単純化したものが**有限状態機械**である。

例えば、「1」「2」「3」という 3 つのボタンが付いていて、正しい暗証番号「1213」を押した場合にのみ開く電子錠などがそうである。これは、「現在までにどのようにボタンが押されたか」という状態を内部で記憶しており、それと押されたボタンとから次の状態と出力（鍵を開けるか否か）を決めるものである。この有限状態機械は右図のように表せる。
（図の細かい説明についてはここでは省略するが、オートマトンの演習でやった通りである）



有限状態機械の「有限」とは、機械の中に覚えておける状態が有限個であることを意味する。このことは現実のコンピュータが広大な記憶装置を扱えるのとは大きく異なり、故にコンピュータで解ける問題も有限状態機械では解けないこともある。

第 6 章はここまで。