アルゴリズムとデータ構造

アルゴリズムとは早く止まる漸化式

森 立平
mori@c.titech.ac.jp

2019 年 6 月 14 日

今日のメッセージ

- **アルゴリズム ≈ 漸化式**
- 漸化式は「1」でなくて「0」で止める
- 漸化式の中の条件の数は減らす
- $y > x$ の最大公約数は $O(\log x)$ 回の剰余の計算で計算できる

今日の目標

- 漸化式からアルゴリズムが設計できることを理解する
- ユークリッドの互除法の計算量を理解する

1 アルゴリズムとは

アルゴリズムとは簡単な計算を組み合わせて複雑な計算をするための手続きのことである。アルゴリズムは数学や物理と比べると体系的に理解することが難しい。この講義ではできるだけ普遍的なアルゴリズムの設計法、解析手法、性質について考えていく。

2 ユークリッドの互除法

自然数 x と y の最大公約数 (greatest common divisor; GCD) をもとめる問題を考えよう。おそらくみんな知っているユークリッドの互除法というアルゴリズムがある。ユークリッドの互除法は次の等式に基づく。

$$\gcd(x,y) = \begin{cases} x, & \text{if } x = y \\ \gcd(x, y-x), & \text{if } x < y \\ \gcd(x-y, y), & \text{otherwise.} \end{cases} \tag{1}$$

この等式が正しいことは $x \le y$ のとき、$(r \mid x) \wedge (r \mid y) \iff (r \mid x) \wedge (r \mid y-x)$ から分かる。上記の等式は任意の自然数 x と y について $\gcd(x,y)$ の値を一意に定める。例えば $x = 6, y = 15$ の場合、$\gcd(6,15) = \gcd(6,9) = \gcd(6,3) = \gcd(3,3) = 3$ と最大公約数が定まる。このように、上記の等式を繰り返し適用 (左辺を右辺で置き換える) していくと**必ず停止する**。このような等式のことを「**漸化式**」と呼ぶことにしよう。効率がいいかどうかは別として**漸化式はそのままアルゴリズムになる**。

次にこのアルゴリズムの実行時間を考えてみよう。実行時間の目安となる量として「計算量」という概念がある。「計算量」の定義は様々であるが、ここでは「引き算をする回数」を計算量の定義としよう。式 (1)に基づくアルゴリズムで $\gcd(n,1)$ を計算しようとすると $n-1$ 回

引き算を実行する必要がある。$x \leq y$ のときは、$y < x$ になるまで y から x を引くので、$y - x$ を $y \,\%\, x$ に置き換えることができる。

$$\gcd(x, y) = \begin{cases} y, & \text{if } x \,\%\, y = 0 \\ x, & \text{if } y \,\%\, x = 0 \\ \gcd(x, y \,\%\, x), & \text{if } x < y \\ \gcd(x \,\%\, y, y), & \text{otherwise.} \end{cases} \tag{2}$$

この漸化式を使えば、$\gcd(n, 1)$ は直接 1 と計算できる。このアルゴリズムの計算量を「剰余を計算する回数」と定義すると、(1)に基づいたアルゴリズムより (2)に基づくアルゴリズムの方が計算量が小さそうである。一般的に $y > x$ について $\gcd(x, y)$ を計算するには $\log x$ に比例する計算量となることを 4章で説明する。

3 漸化式を立てるコツ

アルゴリズムの計算量は次章で詳しく評価することにして、もう少しこの漸化式を改善してみよう。今まで x と y を自然数としていたが、0 で漸化式を止めることにすると次の漸化式が得られる。

$$\gcd(x, y) = \begin{cases} y, & \text{if } x = 0 \\ x, & \text{if } y = 0 \\ \gcd(x, y \,\%\, x), & \text{if } x \leq y \\ \gcd(x \,\%\, y, y), & \text{otherwise.} \end{cases} \tag{3}$$

このようにしても今までと同じ計算結果になることは簡単に確認できる。また、$\gcd(x, 0) = x$ というように**定義域が拡張される**がこれは $x \geq 1$ のときは妥当な定義であろう (x と 0 の最大公約数は $(r \mid x) \wedge (r \mid 0)$ を満たす最大の整数 r と考えれば、これは x である)。しかし一方で $\gcd(0, 0) = 0$ ということになる。これは一見すると不自然な定義にも見える ($\gcd(0, 0) = \infty$ としたいところであろう)。これを正当化するには 2 通り方法がある。1 つ目は $x \preceq y \overset{\text{def}}{\iff} x \mid y$ と半順序を定義した場合に 0 は「最大」の数である。また、半順序の公理として $x \preceq x$ は任意の x について成り立たないといけないので、$0 \mid 0$ を仮定することになる。これは $x \mid y \overset{\text{def}}{\iff} \exists z \in \mathbb{Z}, y = zx$ と考えれば自然であろう。そうすれば、$\gcd(0, 0) = 0$ は正当化できる。もう一つの正当化の方法として、$\gcd(0, 0) = 0$ とおけば、$(\mathbb{Z}_{\geq 0}, \gcd)$ は 0 を単位元とするモノイド (逆元を持つとは限らない群) となることが挙げられる。上のように、漸化式の停止条件を 1 ステップ遅らせることで、**関数の定義域が自然に拡張**された。この拡張された定義域を持つ関数を使ってアルゴリズムを設計すると、漸化式の適用回数が 1 回増えるが、**似たような計算を 2 箇所に書くことを防げる**ことが多い。そのため、「漸化式はできるだけ簡単なところで止める」、「**漸化式は「1」でなくて「0」で止める**」というのは漸化式を立てるコツである。また、もう一つのコツとして漸化式の中では**計算量を大きく増やさない範囲で条件分岐は減らした方がよい**。条件分岐を減らすことで最終的に次の漸化式が得られる。

$$\gcd(x, y) = \begin{cases} y, & \text{if } x = 0 \\ \gcd(y \,\%\, x, x), & \text{otherwise.} \end{cases} \tag{4}$$

この漸化式では $x \leq y$ を期待しているが、$x > y$ の場合は単に引数を入れ替える操作をしている。例えば $x = 15, y = 6$ の場合、$\gcd(15, 6) = \gcd(6, 15) = \gcd(3, 6) = \gcd(0, 3) = 3$ と計算できる。条件分岐を減らしたせいで最初の 1 回分漸化式の適用と計算量が増えている。しかし、条件分岐を減らすことでアルゴリズム (漸化式) は見通しが良くなり、通常はアルゴリズムの実行時間も短くなる。ここまで漸化式を変形してきたが、(2)から(4)まではアルゴリズムの実行時間と計算量は大きくは変わらない。しかし、「アルゴリズムの簡潔さ」と「アルゴリズムの実行時間の速さ」を両立する(4)が一番優れていると考えられる。このような漸化式を立てられるようになるため、この 2 つのコツを押さえておく必要がある。

- **漸化式は「1」でなくて「0」で止める**

- **計算量を大きく増やさない範囲で条件分岐を減らす**

このユークリッドの互除法は紀元前 300 年頃の「ユークリッド原論」に書かれており、世界最古のアルゴリズムと考えられている。

4 ユークリッドの互除法の時間計算量

この章では $\gcd(x,y)$ を計算するアルゴリズム (4)の計算量を解析する。知りたいのは**与えられた x と y 対する計算量**であるが、以下ではまず **与えられた計算量に対する最小の x と y** を考える。非負の整数 x,y が $x<y$ を満たすとき、$r_0:=y$, $r_1:=x$ と置き、$r_{i+2}:=r_i \,\%\, r_{i+1}$ と定義する。$r_m=0$ となったところで数列は停止するとする。この m はユークリッドの互除法の計算量 +1 に他ならない。フィボナッチ数列 $F_0,F_1,\dots$ を

$$\begin{aligned} &F_0:=0, && F_1:=1\\ &F_n:=F_{n-1}+F_{n-2}, \qquad n\geq 2 \end{aligned}$$

と定義する。フィボナッチ数列を使って、逆順にした数列 $r_m,r_{m-1},\dots,r_0$ を下から抑えることができる。

定理 1. 任意の $1\leq k\leq m$ について $r_{m-k}\geq F_{k+1}$.

Proof. $k=1,2$ の場合は、$r_{m-1}\geq 1=F_2$, $r_{m-2}\geq 2=F_3$ より定理は成り立つ。数列は $r_{i+1}<r_i$ を満たす。$r_{i+2}=r_i\,\%\,r_{i+1}\leq r_i-r_{i+1}$ より $r_i\geq r_{i+1}+r_{i+2}$ が成り立つことから、一般の k についても成り立つ。 □

よって、ユークリッドの互除法で計算量が $m-1$ 回以上であれば、$x\geq F_m$ となることが分かった。対偶をとれば、「$x<F_m$ **であればユークリッドの互除法の計算量は $m-2$ 以下**」であることが分かる。黄金比 $\phi:=(1+\sqrt{5})/2$ に対して、$F_m>\frac{1}{\sqrt{5}}\phi^m-1$ であるので、$1+x\leq\frac{1}{\sqrt{5}}\phi^m\Rightarrow x<F_m$ である。よって計算量は $\lceil\log_\phi[\sqrt{5}(1+x)]\rceil-2$ 回以下。$\log_\phi\sqrt{5}\approx 1.6723$ なので、$\lceil\log_\phi(1+x)\rceil$ で計算量を上から抑えられる。よって次の定理を得る。

定理 2. 式 (4)に基づくユークリッドの互除法の計算量 (剰余の計算回数) は $\lceil\log_\phi(1+x)\rceil$ 以下である。

引き算で計算する(1)は計算量は最悪 $y-1$ であった。よって漸化式(1)より、漸化式(2)–(4)の方が効率的なアルゴリズムを与える。

5 オーダー記法

アルゴリズムの計算量はオーダー記法 $O(\cdot)$ を使って表すことが多い。オーダー記法は

$$f(n)=O(g(n))\iff\limsup_{n\to\infty}\left|\frac{f(n)}{g(n)}\right|<\infty\iff\exists C,n_0,\forall n\geq n_0\quad |f(n)|\leq C|g(n)|$$

と定義される。例えば

$$\begin{aligned} n&=O(n), & 100n&=O(n), & n&=O(n^2)\\ \log_{10}n&=O(\log_2 n), & \log_2 n&=O(\log_{10}n), & \log_2 n&=O(n^{0.0001}) \end{aligned}$$

が成り立つ。オーダー記法は定数倍を気にしないで漸近的な計算量を表したい場合に用いられる。例えばユークリッドの互除法の計算量は $O(\log x)$ である。オーダー記法においては log の底が定数であれば、それがなにか気にする必要はない。また、コンピュータサイエンスの分野では log の底が省略されている場合は底は 2 であるとするのが通例である。

6 Binary GCD

ユークリッドの互除法の変種として Binary GCD というアルゴリズムがある。これは Stein によって 1961 年に発表された。Binary GCD は次の漸化式に基づく。

$$\gcd(x,y) = \begin{cases} y, & \text{if } x = 0 \\ x, & \text{if } y = 0 \\ 2\gcd(x/2, y/2), & \text{if } x \text{ and } y \text{ are even} \\ \gcd(x/2, y), & \text{if } x \text{ is even and } y \text{ is odd} \\ \gcd(x, y/2), & \text{if } x \text{ is odd and } y \text{ is even} \\ \gcd(x, (y-x)/2), & \text{if } x \text{ and } y \text{ are odd and } y \geq x \\ \gcd((x-y)/2, y), & \text{otherwise.} \end{cases} \tag{5}$$

この漸化式が正しいことは簡単に確かめられるだろう。条件分岐が多いので、元のユークリッドの互除法よりも悪いように見えるかもしれないが、この漸化式には**剰余 (%) の計算が含まれていない**。その代わり引き算と 2 で割る計算が含まれているが、これらは剰余の計算と比較すると高速に計算ができる (10 進数で表されている 10 の倍数を 10 で割るのが簡単であるのと同じ)。

7 Binary GCD の計算量

引き算を計算する回数を binary GCD の計算量と定義しよう。2 つの整数 x と y のどちらかが偶数のときは最初のステップで引き算が発生しないので、両方とも奇数の場合を考える。

定理 3. 2 つの奇数 x と y が $y > x$ を満たすとき、binary GCD の計算量は $1 + \log_2(y-1)$ 以下である。

Proof. Binary GCD の計算量が m であるとするときの y の下界について考える。奇数のペアの列 $\{(a_n, b_n)\}_{n=0,1,\ldots,m}$ を次のように定義する。

$$\begin{aligned} (a_0, b_0) &= (y, x) \\ a_{i+1} &= \max\{\phi(a_i - b_i), b_i\} \\ b_{i+1} &= \min\{\phi(a_i - b_i), b_i\}. \end{aligned}$$

ただし、非負の整数 t について ϕ は

$$\phi(t) = \begin{cases} 0, & \text{if } t = 0 \\ t, & \text{if } t \text{ is odd} \\ \phi(t/2), & \text{otherwise} \end{cases}$$

と定義する。この奇数のペアの列が m 番目で停止するとき、binary GCD の計算量は m である。すると、$a_{i+1} + b_{i+1} \leq (a_i - b_i)/2 + b_i = (a_i + b_i)/2$ が成り立つ。よって、$a_i + b_i$ は単調減少で非負であるため必ず停止する。$a_{m-1} + b_{m-1} \geq 2$ と $a_i + b_i \geq 2(a_{i+1} + b_{i+1})$ より、$y + x = a_0 + b_0 \geq 2^m$ が成り立つ。y と x は奇数であるので $y \geq x + 2$ が成り立ち、$y - 1 \geq 2^{m-1}$ が得られる。よって、$m \leq 1 + \log_2(y-1)$ が得られる。 □

また、$y = 2^{m-1} + 1$, $x = 2^{m-1} - 1$ のとき等号は達成される。奇数のペアの列は $(2^{m-1} + 1, 2^{m-1} - 1)$, $(2^{m-1} - 1, 1)$, $(2^{m-2} - 1, 1)$, $\ldots$, $(3, 1)$, $(1, 1)$, $(1, 0)$ となる。通常のユークリッドの互除法の計算量は $O(\log x)$ であるが、binary GCD の計算量は $O(\log y)$ である。y が x と比べてとても大きい場合でも、通常のユークリッドの互除法では次のステップで両方の値が x 以下になる。一方で binary GCD では $(2^n - 1, 1)$ からスタートすると、実際に計算量が $\log_2(y+1)$ になる例がある。また、$\log_\phi x > \log_2 x$ である。よって、y が x に比べてとても大きい場合は通常のユークリッドの互除法の方が、そうでない場合は binary GCD の方が計算量が小さいことが分かる。計算量は通常のユークリッドの互除法では% の回数、binary GCD では引き算の回数と定義しているため、計算量が同じ場合も binary GCD の方が実際のアルゴリズムの実行は高速である。

8 C 言語による実装

ユークリッドの互除法 (4)を C 言語で書くと次のようになる。

```
unsigned int Euclidean_gcd_rec(unsigned int x, unsigned int y){
  if(x == 0) return y;
  return Euclidean_gcd_rec(y % x, x);
}
```

数式 (4)をそのままコピーしたようなものだと分かるだろう。次にこれを反復を使って書くと次のようになる。

```
unsigned int Euclidean_gcd_itr(unsigned int x, unsigned int y){
  while(x != 0){
    int z = x;
    x = y % x;
    y = z;
  }
  return y;
}
```

9 おまけ: 連分数展開

実数 z の (正則) 連分数展開とは

$$z = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \cdots}}}}$$

という表現のことである。ここで $a_0, a_1, a_2 \ldots$ は整数とする。実数 z に対してこの連分数展開を得るのは簡単で、次の手続に従って計算すればよい

$$\begin{aligned} a_0 &= \lfloor z \rfloor, & x_0 &= z - a_0, \\ a_{i+1} &= \left\lfloor \frac{1}{x_i} \right\rfloor, & x_{i+1} &= \frac{1}{x_i} - a_i. \end{aligned}$$

ただし、$x_i = 0$ となったら停止する。非負の整数 x, y が $x < y$ を満たしているときに、$z = x/y$ であるとすると,

$$\begin{aligned} a_0 &= 0, & x_0 &= \frac{x}{y} \\ a_1 &= \left\lfloor \frac{y}{x} \right\rfloor, & x_1 &= \frac{y \,\%\, x}{x} \end{aligned}$$

となる。よって、4章で定義した $r_0, r_1, r_2, \ldots,$ を用いると、$x_i = \frac{r_{i+1}}{r_i}$ である。また、$a_i = \lfloor \frac{r_{i-1}}{r_i} \rfloor$ であり、これはユークリッドの互除法における「商」である。つまり、有理数 x/y の連分数展開は $\gcd(x, y)$ のユークリッドの互除法と対応しており、$a_0 = 0$ で $a_1, a_2, \ldots$ は商の列 $\lfloor \frac{r_0}{r_1} \rfloor, \lfloor \frac{r_1}{r_2} \rfloor, \ldots$ となる。この連分数展開は実数を有理数で近似したり、有理数をより分母の小さい有理数で近似するのに役に立つ。