

① 予備知識

a^8 を素朴な方法で計算すると、

$$a \times a \times a \times a \times a \times a \times a \times a$$

7 回の掛け算が必要ですが、2 乗計算を繰り返すと、

$$\left((a^2)^2 \right)^2$$

3 回の掛け算で済みます*1。

一般の自然数 n に対して、 a^n を計算する場合は、 n の 2 進展開の考え方を利用します。自然数 n を

$$n = c_{k-1} 2^{k-1} + \dots + c_3 2^3 + c_2 2^2 + c_1 2 + c_0$$

と 2 進展開 (従って、係数 c_i は全て 0 または 1) するとき、

$$a^n = a^{c_{k-1} 2^{k-1} + \dots + c_3 2^3 + c_2 2^2 + c_1 2 + c_0}$$

$$= a^{c_{k-1} 2^{k-1} + \dots + c_3 2^3 + c_2 2^2 + c_1 2} \times \underbrace{a^{c_0}}_{x_1} = \left(\underbrace{a^2}_{a_1} \right)^{\overbrace{c_{k-1} 2^{k-2} + \dots + c_3 2^2 + c_2 2 + c_1}^{n_1}} \times a^{c_0}$$

↑ 末尾 a^{c_0} を別扱いとし、底 a を 2 乗し (a_1)、指数 $n - c_0$ を 2 で割る (n_1)。

$$= a_1^{c_{k-1} 2^{k-2} + \dots + c_3 2^2 + c_2 2} \times \underbrace{a_1^{c_1} a^{c_0}}_{x_2} = \left(\underbrace{a_1^2}_{a_2} \right)^{\overbrace{c_{k-1} 2^{k-3} + \dots + c_3 2 + c_2}^{n_2}} \times a_1^{c_1} a^{c_0}$$

↑ 末尾 $a_1^{c_1}$ を別扱いとし、底 a_1 を 2 乗し (a_2)、指数 $n_1 - c_1$ を 2 で割る (n_2)。

$$= a_2^{c_{k-1} 2^{k-3} + \dots + c_3 2} \times \underbrace{a_2^{c_2} a_1^{c_1} a^{c_0}}_{x_3} = \left(\underbrace{a_2^2}_{a_3} \right)^{\overbrace{c_{k-1} 2^{k-4} + \dots + c_3}^{n_3}} \times a_2^{c_2} a_1^{c_1} a^{c_0}$$

↑ 末尾 $a_2^{c_2}$ を別扱いとし、底 a_2 を 2 乗し (a_3)、指数 $n_2 - c_2$ を 2 で割る (n_3)。

= (以下同様)

と変形できます。従って、

- 指数が偶数の場合は、底を 2 乗し、指数を 2 で割る。

*1 具体的な計算手順は、まず、 $a_1 = a^2$ を計算し、次に、 $a_2 = a_1^2$ を計算し、最後に、 $a_3 = a_2^2$ を計算します。

- 指数が奇数の場合は、末尾 (=底) を別の変数 x に蓄え、指数から 1 を引き、その上で、偶数の場合と同じ処理をする。

この手順によって a^n が計算できます。この方法を Binary Method と呼びます。

② Maxima による実装例

Maxima を使って、Binary Method を実装してみましょう。

```

1  binary(a, n) := block([x: 1],                                     - binary.mac -
2      while ( n # 0 ) do (
3          if oddp(n) then (
4              x: x * a,
5              n: n - 1
6          ),
7          a: a * a,
8          n: n / 2
9      ),
10     return(x)
11 );
```

4 行目と 5 行目が、指数が奇数の場合の処理であり、7 行目と 8 行目が、指数の偶奇によらず共通に必要な処理です。念のため、正しく実装されているか実験してみましょう。

```
(%i1) load("binary.mac");
(%o1)                                     binary.mac
(%i2) binary(2, 10);
(%o2)                                     1024
(%i3) binary(a, 1000);
(%o3)                                     1000
(%o3)                                     a
```

③ Fibonacci 数

漸化式

$$F_0 = 0, \quad F_1 = 1, \quad F_{n+2} = F_{n+1} + F_n \quad (n \geq 0)$$

で定義される数列 $\{F_n\}$ を Fibonacci 数列、Fibonacci 数列に現れる数を Fibonacci 数と呼びます。定義式により、任意の自然数 n に対して、

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}$$

が成り立つことから、

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^2 \begin{pmatrix} F_{n-1} & F_{n-2} \\ F_{n-2} & F_{n-3} \end{pmatrix} = \cdots = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix} \\ = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

と表せることが分かります*2。そこで、Binary Method を用いて $\begin{pmatrix} F_n & F_{n-1} \\ F_{n-1} & F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1}$ を計算することで、 n 番目の Fibonacci 数 F_n を求める関数を作りたいと思います。

```

1  binary_fib(n) := block([X: ident(2),                                - binary_fib.mac -
2      A: matrix([1, 1], [1, 0])],
3      if (n = 0 or n = 1) then
4          return(n)
5      else (
6          n: n - 1,
7          while ( n # 0 ) do (
8              if oddp(n) then (
9                  X: X . A,
10                 n: n - 1
11             ),
12             A: A . A,
13             n: n / 2
14         ),
15         return(X[1, 1])
16     )
17 );
```

引数が0または1の場合は、そのまま n を出力し (4 行目)、2 以上の場合に、Binary Method を用いて、 $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1}$ を計算します (6 行目から 14 行目)。なお、`ident` は単位行列を生成する関数で、`ident(2)` により、2 次の単位行列 $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ が生成されます。

この関数 `binary_fib` を実行してみると、次のようになります。

```
(%i4) load("binary_fib.mac");
(%o4)                                binary_fib.mac
```

*2 厳密には、仮定 $n \geq 2$ のもとで数学的帰納法で証明し、更に、 $n = 1$ のときにも成り立つことを確認すべきです。

```
(%i5) binary_fib(100);  
(%o5) 354224848179261915075  
(%i6) binary_fib(10000) - fib(10000);  
(%o6) 0
```

引数に 100 と 10000 を与えて実験してみました。10000 番目の Fibonacci 数については、正しく求められているか、組み込み関数 `fib` の結果と比較してみました。