

① 予備知識

分子が1（で分母が自然数）の分数を単位分数と呼び、与えられた（1より小さい正の）分数を（いくつかの）相異なる単位分数の和で表すことを「単位分数分解」と呼びます。

例えば、分数 $\frac{17}{70}$ の場合、 $\frac{1}{7} + \frac{1}{10}$ や $\frac{1}{5} + \frac{1}{24} + \frac{1}{840}$ などが単位分数分解の例になります。

一般に、等式

$$\frac{1}{n} = \frac{1}{n+1} + \frac{1}{n(n+1)}$$

が成り立ちますから、1つでも単位分数分解が見つければ、無数の単位分数分解を生成することが出来ます。

② 欲張りアルゴリズム (greedy algorithm)

1より小さい正の分数 $\frac{b}{a}$ が与えられたとき、「欲張りアルゴリズム」を使うと、必ず単位分数に分解できます。これは、分数 $\frac{b}{a}$ を超えない最大の単位分数 $\frac{1}{n}$ を選び、差 $\frac{b}{a} - \frac{1}{n}$ を計算する、という操作を、差が0になるまで繰り返すだけのいたって素朴なアルゴリズムです¹。しかもこの操作は高々 b 回の繰り返しで終了します。なぜなら、単位分数 $\frac{1}{n}$ の選び方から、不等式

$$\frac{1}{n} \leq \frac{b}{a} < \frac{1}{n-1} \tag{★}$$

が成り立ち、各不等式 $\frac{1}{n} \leq \frac{b}{a} < \frac{1}{n-1}$ ごとに分母を払って、適宜移項することにより、不等式

$$0 \leq bn - a < b$$

が得られます。すなわち、差 $\frac{b}{a} - \frac{1}{n} = \frac{bn - a}{an}$ の分子 $bn - a$ は、元の分数 $\frac{b}{a}$ の分子 b より小さい非負整数となるからです。

また、[★]式から、不等式

$$0 \leq \frac{b}{a} - \frac{1}{n} < \frac{1}{n-1} - \frac{1}{n} < \frac{1}{n(n-1)} \leq \frac{1}{n}$$

が得られます。この不等式 $\frac{b}{a} - \frac{1}{n} < \frac{1}{n}$ から、選び出される単位分数が全て異なることも分かります。

なお、ここで示した事実が Fibonacci により証明されていたことから、「欲張りアルゴリズム」は「Fibonacci のアルゴリズム」とも呼ばれています。

¹ 常に最良の選択枝を選択する操作を繰り返すことにより目的を達成しようとするアルゴリズムを、一般に「欲張りアルゴリズム」と呼びます。

早速欲張りアルゴリズムを Maxima で実装してみましょう。

```
1  unit_frac_decomp_g(r) := block(                                     - unit_frac_decomp_g.mac -
2      [R: r, n: 1, L: []],
3      if (ratnumb(R) and R > 0 and R < 1 ) then (
4          for i while (R # 0) do (
5              for j while (R < 1/n) do n: n + 1,
6              R: R - 1/n,
7              L: endcons(1/n, L)
8          ),
9          return(L)
10     ) else (
11         return(false)
12     )
13 );
```

3行目のif文により、「引数Rが0以上1以下の有理数」の場合にのみ、処理「4行目から9行目」が実行されます。

外側（4行目）のforループで、差が0になるまで処理を繰り返し、内側（5行目）のforループで、最大の単位分数を探しています。なお、4行目と5行目の変数i、jはダミー変数です。実行例は次の通りです。

```
(%i1) load("unit_frac_decomp_g.mac");
(%o1)                                     unit_frac_decomp_g.mac

(%i2) unit_frac_decomp_g(3/8);
(%o2)                                     1  1
                                           [-, --]
                                           3  24

(%i3) unit_frac_decomp_g(100/1001);
(%o3)                                     1    1    1
                                           [--, ---, -----]
                                           11  112  16016
```

③ 実は失敗作でした！

前節のプログラムでは、分母（変数n）を1つずつ増やしながら最大の単位分数を見つけています。このような処理は、理論上は誤りではありませんが、実際のプログラムでは破綻してしまいます。勇気がある人はunit_frac_decomp_g(114/1133)を実行してみてください（お

そらく返ってこなくなります)。分数 $\frac{114}{1133}$ には、 $\frac{1}{11} + \frac{1}{103}$ という短い分解がありますが、欲張りアルゴリズムを使うと、

$$\frac{114}{1133} = \frac{1}{10} + \frac{1}{1619} + \frac{1}{6114424} + \frac{1}{56079265163240}$$

となり、前節の関数 `unit_frac_decomp_g` では、56 兆回もの処理を繰り返して初めて結果が得られます²。

分数 $R = \frac{b}{a}$ を超えない最大の単位分数 $\frac{1}{n}$ を求めることは、逆数 $\frac{1}{R}$ 以上の最小の自然数 n を求めることと同値です。この方針で先のプログラムを書き換えると、次のようになります。

```

1  unit_frac_decomp_g(r) := block(                                     - unit_frac_decomp_g.mac -
2      [R: r, L: []],
3      if (ratnumP(R) and R > 0 and R < 1) then (
4          for i: 1 while (R # 0) do (
5              n: ceiling(1/R),
6              R: R - 1/n,
7              L: endcons(1/n, L)
8          ),
9          return(L)
10     ) else (
11         return(false)
12     )
13 );
```

変更点は、5 行目です。ループ処理の代わりに、関数 `ceiling` を用いて、逆数 $\frac{1}{R}$ 以上の最小の自然数 n を求めています。

実行例は次の通りです。

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

(%i5) unit_frac_decomp_g(114/1133);
      1      1      1      1
(%o5)  [--, ----, -----, -----]
      10 1619 6114424 56079265163240

(%i6) apply("+", %);
```

² 差が 0 になるまで処理を繰り返す代わりに、差の分子が 1 になるまで繰り返す方針で若干改良を加えれば、繰り返し回数を 611 万回に減らすことが出来ますが、焼け石に水でしょう。

(%06)

114

1133