平成21年度 サイエンス・パートナーシップ・プロジェクト

暗号の基礎から実用まで

第2部 : Python を使った数値計算

高知大学 理学部

塩 田 研 一

2009年9月3日

2　Python を使った数値計算

現代の暗号では 10 進数で何百桁という大きな数字を扱います。普通のプログラム言語と違って、Python (パイソン) というプログラム言語を用いればそのような大きな数字の計算も簡単に組むことができます。

ここでは Python のサンプルプログラムを使って暗号処理の基礎となっている計算について勉強しましょう。

2.1　はじめに

コンピュータはどんな計算も一瞬でできてしまうと思ってはいませんか。実はひとくちに「計算」と言っても、

- 当たり前に高速にできる計算
- 工夫をすれば高速にできる計算
- どんなに工夫しても高速にはならない計算

の 3 種類があります。

現代の暗号はその違いを巧みに利用して

- 正規ユーザは高速にできる計算で暗号が使える
- 第三者が暗号を解読するためには高速にはできない計算が必要なので事実上不可能である

という仕組みを実現しています。

2.2　ビット

まず、数の大きさを測る「ビット」という言葉を覚えましょう。

難しいことはありません。数字を 2 進数で表したときの桁数を「何ビット」というように数えるだけです。

例えば 10 進数の 53 を 2 進数で表すと 110101 になりますから 53 は 6 ビットの数です。53 のビット長は 6 である、という言い方もします。

10 進数と 2 進数の変換　サンプルプログラムの「１０進数から２進数.py」、「２進数から１０進数.py」を用いると 10 進数と 2 進数の変換ができます。

k ビットの数を 10 進数で表したときの桁数は大体 $0.3\times k$ 位になります。例えば 20 ビットの数は 10 進数では 6 桁位に、1000 ビットの数は 10 進数では 300 桁位になります。では問題です。

問題

(1) 19 のビット長はいくつでしょうか。

(2) 30 ビットの数と 20 ビットの数を足すと、答えは何ビット位の数になるでしょうか。

(3) 30 ビットの数から 20 ビットの数を引くと、答えは何ビット位の数になるでしょうか。

(4) 30 ビットの数と 20 ビットの数を掛けると、答えは何ビット位の数になるでしょうか。

(5) 30 ビットの数を 20 ビットの数で割ると、商は何ビット位の数になるでしょうか。

(6) 30 ビットの数の平方根の整数部分（小数点以下を省いた数）は何ビット位の数になるでしょうか。

2.3　高速にできる計算

2.3.1　四則演算

コンピュータの内部では数字は 2 進数として表されて計算されます。

皆さんは 2 進数の四則演算（足し算、引き算、掛け算、割り算）はできますか。

$$\begin{array}{r} 1\,1\,0\,1\,0\,0\,1 \\ +\quad 1\,1\,0\,0\,1\,1 \\ \hline 1\,0\,0\,1\,1\,1\,0\,0 \end{array} \qquad \begin{array}{r} 1\,1\,0\,1\,0\,0\,1 \\ -\quad 1\,1\,0\,0\,1\,1 \\ \hline 1\,1\,0\,1\,1\,0 \end{array}$$

```
      1 1 1 0 1
×       1 0 1 1
-------------
      1 1 1 0 1
    1 1 1 0 1
  1 1 1 0 1
-------------
1 0 0 1 1 1 1 1 1
```

```
        1 0 1
    -------
1 0 1 ) 1 1 1 0 1
      1 0 1
    -------
        1 0 0 1
        1 0 1
    -------
          1 0 0
```

足し算　足し算の場合、各桁では、下の桁からの繰り上がりがないときは

$$0+0=0,\quad 0+1=1,$$
$$1+0=1,\quad 1+1=10,$$

下の桁から 1 繰り上がっているときは

$$1+0+0=1,\quad 1+0+1=10,$$
$$1+1+0=10,\quad 1+1+1=11$$

という計算をします。これは「全加算器」と呼ばれる回路で実現され、下の桁から順に処理が行われるので、足し算にはおおよそビット数に比例した計算時間が掛かります。一瞬で終わっているように見えますが、ちゃんと時間は掛かっています。サンプルプログラム「四則演算.py」で計算時間を計測してみましょう。

掛け算　同様にサンプルプログラム「四則演算.py」で掛け算の計算時間も計測してみましょう。

同じビット数だと足し算より遥かに長い時間が掛かっていますね。筆算の各部分で全加算器の処理が必要だと考えれば、計算時間は筆算の面積に比例するだろうと考えられ、例えばビット数が 2 倍になれば面積は 4 倍ですから 4 倍くらいの時間が掛かるのは仕方がない、ということになります。(実際には色々と工夫がされているので 4 倍までは掛からないようです。)

2.4　工夫すれば高速にできる計算

2.4.1　最大公約数

天秤クイズ　天秤 (てんびん) と、3g と 5g の分銅が沢山あるとき、測ることのできる一番小さい重さは何 g でしょうか。

答えは 1g。片方の皿に 3g の分銅を 2 個、もう片方に 5g の分銅を乗せれば差が 1g になりますね。

では 5g と 7g の分銅ではどうでしょう。今度も答えは 1g。片方の皿に 5g の分銅を 3 個、もう片方に 7g の分銅を 2 個乗せれば差が 1g になります。

10g と 14g の分銅ではどうでしょうか。さっきの重さの 2 倍ですから、さっきと同じ分銅の個数で 2g が測れ、これが答えです。

> **定理**　ag と bg の分銅が沢山あるとき、a と b の最大公約数を d とすると、測ることのできる最小の重さは dg である。

最大公約数の数値計算　では最大公約数はどうやって計算するのでしょうか。

数が小さいときは素因数分解をして共通する素因数を拾いましたね。

$$48=2^4\times 3,\quad 108=2^2\times 3^3$$
$$\Rightarrow\ 48\text{ と }108\text{ の最大公約数は }2^2\times 3=12$$

プログラムを簡単に書こうと思うと、a と b の小さい方から数を 1 ずつ減らしていってもう片方を割り切るまで探す、という方法が考えられます。

108 を 48 で割ると余り 12
108 を 47 で割ると余り 14
108 を 46 で割ると余り 16
⋮
108 を 12 で割ると余り 0
$\Rightarrow$ 48 と 108 の最大公約数は 12

しかしこれらの方法は数が大きくなると全く役に立ちません。その代わり「ユークリッドのアルゴリズム」という方法を用いると瞬時にして答えを出すことができます。サンプルプログラム「最大公約数.py」で計算時間を比較してみましょう。

2.4.2　分銅の個数も計算しよう

先の定理を言い換えると次のようになります：

> **定理**　二つの自然数 a, b に対して、a と b の最大公約数を d とすると、$ax+by$ の形に表すことのできる最小の自然数は d である。
> 特に、$d = ax + by$ と表すことのできる整数 x, y が必ず存在する。

例えば 3^{50} と 5^{30} の最大公約数は 1 ですから

$$3^{50} \times 115608417198323432874 + 5^{30} \times (-8911525644182230377582 5) = 1$$

となります。5^{70} と 7^{50} なら

$$5^{70} \times (-5461582105441054148794747352651082688073 46) + 7^{50} \times 25722712856476880783165403679181811491987155924 99 = 1$$

といった具合です。

このような x, y が存在することは定理で証明されていますから、x は 1 から順番に探してゆけばみつかるはずです。しかしそれは「理論的に」正しいだけであって、実際に計算すると宇宙が終わっても答えにたどり着くことができません。

ここでやはり「ユークリッドのアルゴリズム」を用います。サンプルプログラム「てんびん.py」で確かめてみましょう。

2.4.3　素数判定

まずは問題です。

問題

(1) 137 は素数でしょうか。

(2) 187 は素数でしょうか。

皆さんはどうやって答えを出しましたか。小さな数から順番に割ってみて、約数がみつかったら合成数、みつからなかたら素数、という考え方を使った人が多いのではないでしょうか。

自然数 n の約数を探すとき、$\sqrt{n}$ までの約数を探せばよい、ということは知っていますか。$\sqrt{n}$ より大きい約数 a があれば $b = n/d$ は $\sqrt{n}$ より小さい約数になりますから、a は n/b として見つけることができるからです。

しかし、この考え方で素数判定をしようとしても 45 ビットあたりから次第に計算時間が増加してゆき、使い物にならなくなります。そこで暗号の設計には「確率的素数判定法」という方法が多く用いられます。そのうちのひとつ、ソロベイ・ストラッセン法という方法をサンプルプログラム「素数生成.py」に組んでありますので計算時間を測ってみましょう。

2.4.4　べき乗計算

またまた問題です。

問題　x の数値が与えられたとき

(1) 何回の掛け算で x^{64} を計算することができるでしょうか。

(2) 何回の掛け算で x^{83} を計算することができるでしょうか。

植木算で、64 乗は 63 回、83 乗は 82 回と思ってしまいませんでしたか。

答え

(1)
$$\begin{aligned} x^2 &= x \times x, \\ x^4 &= (x^2) \times (x^2), \\ x^8 &= (x^4) \times (x^4), \\ x^{16} &= (x^8) \times (x^8), \\ x^{32} &= (x^{16}) \times (x^{16}), \\ x^{64} &= (x^{32}) \times (x^{32}) \end{aligned}$$
とすれば 6 回。

(2) (1) の計算のあと、
$$x^{83} = x^{64} \times x^{16} \times x^2 \times x$$
とすれば 9 回。

この方法を「反復2乗法」と言います。暗号の計算ではべき指数が1000ビットの数であるようなべき乗計算が必要となります。単純に1回ずつ掛けると

$$2^{1000} \qquad 10^{300} \text{ 回}$$

位の掛け算が必要となりますが、反復2乗法なら

$$2 \times 1000 \qquad 2 \times 10^3 \text{ 回}$$

以内で済みます。その差は実に 10^{296} 倍以上になり、宇宙の推定年齢 4.33×10^{17} と比べると如何に大きな差があるかがわかります。

2.5 工夫しても高速にならない計算

2.5.1 素因数分解

誰もが知っている素因数分解法としては、先ほどの素数判定と同じく約数を探すという方法があります。しかし、これも素因数が25ビット位になると急激に計算時間が増大してしまいます。サンプルプログラム「素朴な素因数分解.py」で計測してみましょう。

現在までに色々な素因数分解法が提案されてきましたが、その中でもかなり優秀な「複数多項式二次ふるい法」という方法をサンプルプログラム「高級な素因数分解.py」に組んであります。どの位のビット数の数まで有効か実験してみてください。

2.6 まとめ

- 当たり前に高速にできる計算
  - 四則演算
- 工夫をすれば高速にできる計算
  - 最大公約数
  - てんびんクイズ
  - 素数判定
  - べき乗
- どんなに工夫しても高速にはならない計算
  - 素因数分解

今日の話はここまでです。次回はこれらの計算を組み合わせて設計されている「RSA暗号」のお話です。