アルゴリズム及び演習 第 1 回演習解答

小野 孝男*

2007 年 4 月 16 日

まず計算量の表記について確認しよう. ここで $f(n)$, $g(n)$ は n の関数とする.

- 関数 $f(n)$, $g(n)$ について $\boxed{f(n) = O(g(n))}$ であるとは
  2 つの正定数 c, n_0 が存在し, 全ての $n \geq n_0$ に対して $f(n) \leq c \cdot g(n)$ である
  ことをいう.
- $f(n)$, $g(n)$ について $\boxed{f(n) = \Omega(g(n))}$ であるとは
  2 つの正定数 c, n_0 が存在し, 全ての $n \geq n_0$ に対して $f(n) \geq c \cdot g(n)$ である
  ことをいう.
- $f(n)$, $g(n)$ について $\boxed{f(n) = o(g(n))}$ であるとは
  任意の c に対して (c によって定まる) n_c が存在し, $n \geq n_c$ ならば $c \cdot f(n) < g(n)$ である
  ことをいう.

$O(\cdot)$, $\Omega(\cdot)$ の定義では n_0 が (c に依存しない) 正定数であるのに対し $o(\cdot)$ の定義では n_c が c に依存して定まる正数であることに注意すること.

次に, オーダーを考えるうえでは, 以下のことを念頭におくと簡単である:

- 任意の定数 $c > 0$ に対し $\log n = O(n^c)$,
- 定数 $0 < c_1 < c_2$ に対し $n^{c_1} = O(n^{c_2})$,
- 任意の定数 $c > 1$ に対し $n = O(c^n)$,
- 定数 $1 < c_1 < c_2$ に対し $c_1^n = O(c_2^n)$.

1. 各項のうち, 正で最もオーダーの大きなものを選び, 定数係数を無視すればよい:

   $f_1(n)$: $3n$ の項は負なので無視する. $2n^2$ と 1 のうち, オーダーの大きなものは $2n^2$ である. ここで定数係数を無視すると $g_1(n) = n^2$ となる.

   $f_2(n)$: $(\log n)^2 \sqrt{n}$ と n のオーダーを比較するが, この 2 つの比は $(\log n)^2 / \sqrt{n}$ であり, これは $n \to \infty$ で 0 に収束する. つまり $(\log n)^2 \sqrt{n} = O(n)$ なので $g_2(n)$ としては n が適切である.

   $f_3(n)$: 第 2 項は負 (しかも $n \to \infty$ で 1 に収束する) なので第 1 項のみ考える. ここで

* ono@is.nagoya-u.ac.jp

$(5/4)^{n-1} = (4/5)(5/4)^n$ なので, $g_3(n)$ として適切かつ最も簡単なのは $(5/4)^n$ である.

2. それぞれの関数をオーダー表記すると

$$O(n), O((\log n)^3), O(3^n), O(n \log n), O(n^2), O(4^n), O(n^{1/2})$$

である. これらをオーダーの小さいものから順に並べると

$$O((\log n)^3), O(n^{1/2}), O(n), O(n \log n), O(n^2), O(3^n), O(4^n)$$

となる. 元の関数で書けば

$$(\log n)^3,\ \sqrt{n}, n + 4, 0.1(n + 7) \log n, 1.5n^2, 3^{n+2}, 2^{2n}$$

である.

3. $f(n) = O(g(n))$ より, 定数 c_1, n_1 が存在し全ての $n \geq n_1$ に対し $f(n) \leq c_1 \cdot g(n)$ が成り立つ. 同様に, $g(n) = O(h(n))$ より定数 c_2, n_2 が存在し全ての $n \geq n_2$ に対して $g(n) \leq c_2 \cdot h(n)$ である. ここで $c = c_1 c_2, n_0 = \max(n_1, n_2)$ とおくと全ての $n \geq n_0$ に対し $f(n) \leq c_1 \cdot g(n) \leq c_1 c_2 \cdot h(n) = c \cdot h(n)$ となるが, これは $f(n) = O(h(n))$ を意味する.

4. 両方向で証明を行う.

(⇒):

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

は「任意の定数 $\epsilon > 0$ に対し (ϵ に依存して定まる) 正数 N が存在し, 全ての $n \geq N$ に対して $f(n)/g(n) < \epsilon$ である」ことを意味する. ここで $c = 1/\epsilon$ とおくと, 全ての $n \geq N$ に対し $c \cdot f(n) < g(n)$ が得られるが, これはまさに $f(n) = o(g(n))$ の定義である.

(⇐): $f(n) = o(g(n))$ より, 任意の c に対し N が存在し, 全ての $n \geq N$ に対して $c \cdot f(n) < g(n)$ である. つまり $n \geq N$ ならば $f(n)/g(n) < 1/c$ である. ここで $\epsilon = 1/c$ とおくと, 任意の $\epsilon > 0$ に対して正数 N が存在し, 全ての $n \geq N$ に対して $f(n)/g(n) < \epsilon$ となることがわかる. つまり

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

である.

5. 整数 n に対し $\log n$ は n のビット数を表す. このことから, 「定数回の演算によって n が 1 ビット小さくなれば ($\approx n$ が半分になれば) よい」ことがわかる. ここで, 指数関数について $x^{2n} = (x^2)^n = (x^n)^2$ であることから 2 つの再帰的アルゴリズム (図 1) が考えられる.
これらのアルゴリズムは, いずれも n の 2 進展開に基づいて計算を行う. 例えば $n = 13$ を 2 進展開すると 1101_2 となるが, これに対する再帰的な POWER の呼出しで返される値は

$$x^1 \to x^{11_2} = x^3 \to x^{110_2} = x^6 \to x^{1101_2} = x^{13}$$

となる. このように, これらのアルゴリズムは n の 2 進展開を上位ビットから調べていることになる.

一方, 再帰的な計算であっても n の 2 進展開を下位ビットから調べるようなアルゴリズムも設計することができる (図 2 左). これは末尾再帰 (自分自身を最後に 1 回だけ呼出し, その結果をそのまま返す) アルゴリズムになるので, 繰り返しで書換えることができる (図 2 右).

その 1

```
procedure POWER1(x, n)
    if n = 0 then
        return 1
    else
        y = POWER1(x^2, ⌊n/2⌋)
        if n mod 2 = 1 then
            y = y * x
        end if
        return y
    end if
end procedure
```

その 2

```
procedure POWER2(x, n)
    if n = 0 then
        return 1
    else
        y = POWER2(x, ⌊n/2⌋)
        y = y * y
        if n mod 2 = 1 then
            y = y * x
        end if
        return y
    end if
end procedure
```

図 1　2 つの再帰的 `power(x,n)`

```
procedure POWER_TR(x, n)
    return POWER_1(x, n, 1)
end procedure
procedure POWER_1(x, n, z)
    if n mod 2 = 1 then
        z = z * x
    end if
    return POWER_1(x * x, ⌊n/2⌋, z)
end procedure
```

```
procedure POWER_LOOP(x, n)
    z = 1
    while n > 0 do
        if n mod 2 = 1 then
            z = z * x
        end if
        x = x * x
        n = ⌊n/2⌋
    end while
    return z
end procedure
```

図 2　末尾再帰の POWER とその繰り返しでの記述