アルゴリズムの計算量

Computational complexity of algorithms

講義「アルゴリズムとデータ構造」配付資料: 井上 康介

1 アルゴリズムの評価基準

コンピュータに同じ計算 (作業) をさせることに対して，様々な手順 (アルゴリズム) でその問題を解くことができる．例えば，多数あるデータを特定の基準に基づいて並べ替える作業をソーティング (sorting) というが，同じソーティングという問題に対して，基本交換法 (bubble sort)，改良選択法 (heap sort)，改良交換法 (quick sort) など多数のアルゴリズムが提案されている．コンピュータに解かせたい問題が与えられた際，どのアルゴリズムを利用するべきかを考えるためには**アルゴリズムの評価**をしなければならないが，ここでの評価基準としては以下のものが考えられる．

(1) reliability: 信頼性が高いこと
(2) efficiency: 処理効率がよいこと
(3) generality: 一般性があること
(4) scalability: 拡張性があること
(5) intelligibility: 分かりやすいこと
(6) portability: 移植性が高いこと

このうち，(1) については，これは当然の前提であり，間違った答えや誤差の大きな答えを出すようなアルゴリズムを利用することはできない．また，(3)~(6) はどちらかといえば各論的であり，状況に応じた部分のある基準といえる．そこでアルゴリズム理論では主に (2) の基準，すなわち，「計算回数が少なくてすみ，処理スピードが速いこと」に着目して議論する．

2 オーダによる処理時間の評価

処理効率を測るための基準として，手続きを実行し始めてから終了するまでの計算時間が分かればよい．これを知るための方法として，最も素朴に考えれば，それぞれのアルゴリズムをコンピュータプログラムに翻訳し，実際のコンピュータで走らせて時間を測るという方法が考えられるが，それは不適切である．なぜならアルゴリズムを比較する目的は，どれをプログラムにすべきかを決めるためであって，プログラムに翻訳する前にどちらがよいかを判断したいということがまずある．更に，使うコンピュータの種類やオペレーティングシステム，使用する言語などに依存しない尺度であればその方が望ましい．

ここで登場するのが**オーダ** (order) と呼ばれる「ものさし」である．これは，アルゴリズムがどのような速さで目的の処理を実行するかを，入力の大きさ n の関数の形で表現するものであり，実際にプログラムを作って走らせてみなくても評価ができるという利点がある．

A を 1 つのアルゴリズムとする．A に与える**入力の大きさ** (input size) を n とする．例えば多数の数の中で最大値を求めるアルゴリズムにおいては，数の個数 n が入力の大きさとなる．入力の大きさ n のことを**問題の大きさ** (problem size) と呼ぶこともある．さて，大きさ n の入力に対してアルゴリズム A が結果を出すまでに要する処理時間を $f(n)$ とする．とは言っても，$f(n)$ という関数の具体的な形までが分かるという訳ではない．なぜなら，アルゴリズム A をどのようなプログラム言語に翻訳してどのようなコンピュータで走らせるかによっても，計算時間は変わってしまう．だからここでは，計算環境が決まれば $f(n)$ という関数も決まるだろうと想像するだけである．

ここで n をどんどん大きくしていったとしよう．すると $f(n)$ も当然大きくなっていく．このとき $f(n)$ がどの程度のスピードで大きくなっていくか，そのスピードに注目する．ここで，スピードを評価する目的で，何らかの n の簡単な関数 $p(n)$ を使って，$f(n)$ とスピードを比較することを考える．即ち，次の記号を導入する．

定義 (オーダ) $f(n)$ と $p(n)$ を，自然数の上で定義された 2 つの関数とする．任意の自然数 n に対して

$$\frac{f(n)}{p(n)} < C \tag{1}$$

を満たすという性質を持った，n に依存しない定数 C が存在するとき，

$$f(n) = \mathrm{O}(p(n)) \tag{2}$$

と書いて，「$f(n)$ は $p(n)$ の**オーダ** (order) である」という．

式 (1) が満たされるということは，どんな n に対して

も，$f(n)$ は $p(n)$ の C 倍を超えない——即ち，n を大きくしていったとき，$f(n)$ はたかだか $p(n)$ に比例するスピードでしか大きくならない——ことを意味する．そこで，$f(n)$ が与えられたとき，$f(n) = \mathrm{O}(p(n))$ を満たす**できるだけ簡単な** $p(n)$ を見つければ，$f(n)$ の振る舞いを $p(n)$ で近似的に眺めることができるようになる．

例えば，$f(n) = 3n^2+8n+6$ であるとしよう．$p(n) = n^2$ とすると，

$$\frac{f(n)}{p(n)} = \frac{3n^2+8n+6}{n^2} = 3 + \frac{8}{n} + \frac{6}{n^2}$$
$$\leq 3+8+6 = 17$$

であるから，17 より大きい C に対して式 (1) が成り立つ．従って $f(n) = \mathrm{O}(n^2)$ である．同様に $p(n) = 2n^2$ や $p(n) = n^3$ とおいても式 (1) を満たす C が作れるから，$f(n) = \mathrm{O}(2n^2)$ でもあり，$f(n) = \mathrm{O}(n^3)$ でもある．このように，$f(n)$ のオーダを表す関数 $p(n)$ は無数にある．実際，最高次数の係数が正の 2 次以上の任意の多項式 $p(n)$ に対して，$f(n) = \mathrm{O}(p(n))$ である．

一方で，n の 2 次未満の次数の式 $p(n)$ は式 (1) を満たさない．例えば $p(n) = n\sqrt{n}$ の場合，

$$\frac{f(n)}{p(n)} = \frac{3n^2+8n+6}{n\sqrt{n}} = 3\sqrt{n} + \frac{8}{\sqrt{n}} + \frac{6}{n\sqrt{n}}$$

であり，n が無限大に近づくときはこの比の値も無限大に近づく．以上の観察から，$f(n) = 3n^2+8n+6$ の大きくなるスピードを最も忠実に表すのは n の 2 次式であり，その中で最も簡単な式は $p(n) = n^2$ である．従って，

$$f(n) = \mathrm{O}(n^2)$$

と書くのがよい．

> 一般に，$f(n)$ が k 次多項式なら，$f(n)$ の大きくなるスピードを忠実に表す $p(n)$ はやはり k 次多項式であり，そのうち最も簡単なものは $p(n) = n^k$ であるから，$f(n) = \mathrm{O}(n^k)$ と書くのがよい．

計算時間をオーダで表記することによって，用いるコンピュータや言語に依存しない形で，入力の大きさに従って計算時間がどのようなスピードで増大していくのかを記述することができた．オーダで評価するということの意味は，「計算時間は $3n^2$ で増えるのか，$4n^2$ で増えるのか」ではなく，「計算量は何次式のオーダで増えるのか」を問題にしているということである．従って，**最大次数以外の項は無視し，更に定数係数も無視する**わけである．

一般的にオーダをどのように考えればよいかについては，以下のように言える．

$\mathrm{O}(1)$　最速．問題の大きさ n に依存せず一定時間で終わる．

$\mathrm{O}(\log n)$　これは非常に速い．

$\mathrm{O}(n)$　これも速い．

$\mathrm{O}(n \log n)$　これも速いと言って良い．n が大きいとき $\log n$ は非常に緩やかに増加する関数なので，$\mathrm{O}(n)$ と大きく変わらない．

$\mathrm{O}(n^2)$　これは，許されないほどではないが，かなり遅い．アルゴリズムを設計するときには，できる限りこれ以下のオーダのものを作る努力をすべきである．

$\mathrm{O}(n^3)$　これは多くの場面で許される限界である．これよりオーダの大きなアルゴリズムは，特殊な場合を除いて実用に適さない．

$\mathrm{O}(c^n)$ (**ただし c は 1 より大きい定数**)　これは**指数オーダ** (exponential order) と呼ばれる．遅すぎて話にならない．

3　計算量

アルゴリズム A の計算時間 $f(n)$ について，$f(n) = \mathrm{O}(p(n))$ と書くことができるとき，それは「$f(n)$ はたかだか $p(n)$ に比例するスピードでしか大きくならない」という意味であった．このとき，アルゴリズム A の**時間複雑度** (time complexity) は $\mathrm{O}(p(n))$ であるという．これは，アルゴリズムの実行にかかる時間に着目した評価指標である．

一方，計算時間と同様，アルゴリズムの中で使われる記憶領域の大きさもオーダで測ることができる．アルゴリズム A が，大きさ n の入力に対して大きさ $g(n)$ ビットの記憶領域を必要とし，$g(n)$ が $g(n) = \mathrm{O}(q(n))$ を満たすとする．このとき，$\mathrm{O}(q(n))$ をアルゴリズム A の**空間複雑度** (space complexity) という．

時間複雑度と空間複雑度を合わせたものを，**複雑度** (complexity) または**計算量**[*1]という．

[*1] 日本語の「計算量」は英語の “complexity” に対応する．また，アルゴリズムの計算量であるという意味を強調して “computational complexity” (直訳すれば「計算の複雑度」) とも呼ぶ