Cook, et al.: Combinatorial Optimization
Chap. 1: Problems and Algorithms

安藤 和敏

April 28, 2003

0. アルゴリズム超入門 — ソーティングを例題として —

ソーティングの問題

与えられた実数の (あるいは, 文字の) 列 $a_1, a_2, \cdots, a_n$ を小さい順に並べ替えよ.

この問題を解くアルゴリズムには, 例えば Insertion Sort, Selection Sort, Quick Sort, Merge Sort などがある.

Selection Sort の C コード

```
void selection(int n, int a[]) {

  int i,j,min,t;
  for (i=1;i<n;i++) {
    min=i;
    for (j=i+1;j<=n;j++) {
      if (a[j]<a[min]) min=j;
    }
    t=a[min]; a[min]=a[i]; a[i]=t;
    print(n,a);
  }
}
```

i	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
	8	9	5	7	4	10	6	2
1	**2**	9	5	7	4	10	6	8
2	2	**4**	5	7	9	10	6	8
3	2	4	**5**	7	9	10	6	8
4	2	4	5	**6**	9	10	7	8
5	2	4	5	6	**7**	10	9	8
6	2	4	5	6	7	**8**	9	10
7	2	4	5	6	7	8	**9**	10

Figure 0.1: Selection Sort の動き

Selection Sort における比較演算の回数は, 常に $\frac{n(n-1)}{2}$ 回.

Insertion Sort の C コード

```
void insertion(int n, int a[]) {

  int i,j,v;
  for (i=2;i<=n;i++) {
    v = a[i]; j=i;
    while (a[j-1] > v) {
      a[j]=a[j-1]; j--;
    }
    a[j] = v;
    print(n,a);
  }
}
```

i	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
	8	9	5	7	4	10	6	2
2	8	9	5	7	4	10	6	2
3	5	8	9	7	4	10	6	2
4	5	7	8	9	4	10	6	2
5	4	5	7	8	9	10	6	2
6	4	5	7	8	9	10	6	2
7	4	5	6	7	8	9	10	2
8	2	4	5	6	7	8	9	10

Figure 0.2: Insertion Sort の動き

Insertion Sort における比較演算の回数は, 最悪の場合に $\frac{(n+2)(n-1)}{2}$ 回, 最も少ない場合には, $n-1$ 回.

1.2 Mesuring runnig times

1.2.1. 問題 (problem) と 問題例 (problem instance)

問題とは,

- 与えられた実数の (あるいは, 文字の) 列 $a_1, a_2, \cdots, a_n$ を小さい順に並べ替えよ.
- 2 つの行列 A と B に対して, それらの積 AB を求めよ.
- 二つの正整数 a_0 と a_1 が与えられたとき, a_0 と a_1 の最大公約数を求めよ.
- n 次正方行列 A が与えられたとき, A の行列式を求めよ.

のようなものである. (これは, 定義ではない. 厳密な定義もあるが, 省略する.)

一つの問題に対して, 具体的な数値を与えたものを**問題例**と呼ぶ. 例えば, ソーティング問題に対する問題例は,

$$(3, 9, 2, 11, 4, 6, 20) \text{ とか } (9, 4, 21, 10, 11, 1)$$

などである. 一般に, 一つの問題に対する問題例は無限にある.

1.2.2. 問題とアルゴリズム (algorithm)

ある問題に対してそれを解くための手順の記述をアルゴリズムという. アルゴリズムは, コンピュータ・プログラムの「もと」になるもので, どのような言語を使ってアルゴリズムをもとに特定のプログラムを書くことを**実装する** (implement) とか**コード化する** (code) と言ったりする.

アルゴリズム	比較演算の回数
Selection Sort	$\frac{n(n-1)}{2}$
Insertion Sort	$n-1 \sim \frac{(n+2)(n-1)}{2}$

Table 1: サイズ n の問題例に対する比較演算の回数

1.2.3. アルゴリズムの良し悪しを測るものさし (計算ステップの上界)

一つの問題に対して, 一般に複数のアルゴリズムが存在する. そのような場合には, 個々のアルゴリズムの性能をどのように評価すべきであろうか?

実際にプログラムを書いてみて, それをいくつかの問題例に対して実行し, アルゴリズムごとの計算時間を比較するというのも一つの方法である. しかし, いくつかの問題点がある.

- 同じアルゴリズムでも実装の仕方によって, 計算時間が異ってくるかもしれない.
- 計算機が異なると計算時間が異なってくるかも知れない.
- 問題例は一般に無限個あるのに, 特定の問題例たちだけの計算時間によってアルゴリズムの優劣を判断するのは妥当であろうか?

これらの問題を解決するために, Turing Machine や RAM (Random Access Machine) などの計算機の数学的モデルを使ってアルゴリズムを解析するという方法がある. ここでは, これら計算機のモデルについては深く踏み込まない. その代りに, あるアルゴリズムが与えられたサイズ n を持つ問題例を解くためにかかる**計算ステップ**を数えることにする.

しかし, まだ問題がある. 例えばソーティングの例において, 比較演算の数を数えてみよう. Selection Sort と Insertion Sort を比べると, 一概にどちらが良いかとは言えない (表 1 を見よ).

全ての問題例について, 平均をとるという方法もあるのが, その計算は一般に困難である.

一つの評価基準として, **最悪の問題例に対する計算ステップ**, あるいは, **全ての問題例に対する計算ステップの上界**がある. これは, それ以上計算ステップがかかることがないという保証を与える.

例えば, Selection Sort のサイズ n の問題例たちに対する計算ステップ (比較演算) の上界は $\frac{n(n-1)}{2}$ であり, Insertion Sort のそれは, $\frac{(n+2)(n-1)}{2}$ である.

良い上界をもつアルゴリズムが実際に速いとは限らない. というのは, もしあるアルゴリズムに対する最悪の問題例というものが現実にはほとんど起らないものである場合には, そのアルゴリズムの性能に対する評価は悲観的過ぎるかも知れない. (このことに対する有名な例が, 線型計画問題に対するアルゴリズムであるシンプレックス法である: シンプレックス法は広いクラスの問題に対して非常に速いが, 理論的に良い計算ステップの上界をもっていない.)

1.2.4. 計算ステップの数えかた (1) 算術演算モデル

算術演算モデル (*arithmetic operation model*) とは, 算術演算: 足し算 ($+$), 引き算 ($-$), かけ算 ($\times$), 割り算 /, 比較 ($\leq$, $=$), 代入, 等を 1 計算ステップとみなす.

さきほどのソーティングでは, 比較演算の回数しか数えていなかったが, Insertion Sort においても Selection Sort においても, 実際には代入演算も行われている. 代入演算は何回行われているか?

1.2.5. 計算ステップの数えかた (2) ビット演算モデル

実際には, 計算機の中では数は 2 進数で表現されており, 算術演算も 2 進数に対して実行される. **ビット演算モデル** (*bit operation model*) においては, 1 ビットごとの演算 ($+, -, \times, <, \cdots$) を 1 計算ステップとして数える.

正整数 x を 2 進表記する場合に何ビット (何桁) 必要か? 2 進表記したときに, l ビットが必要な数を n とすると,

$$2^{l-1} = \overbrace{10\cdots0}^{l\text{ 個}} \leq n \leq \overbrace{1\cdots1}^{l\text{ 個}} = 2^l - 1$$

であるから, lg ($= \log_2$) をとって, $l - 1 \leq \log n < l$. よって, $l = \lfloor \log n \rfloor + 1$. (実数 α に対して, $\lfloor \alpha \rfloor$ は, α を越えない最大の整数を表す. $\alpha \geq 0$ のときは, 小数点以下を切り捨てた整数値.)

例えば2つの正整数 x と y の足し算には, 高々$2 \cdot \max\{\lfloor \log x \rfloor + 1, \lfloor \log y \rfloor + 1\}$ 回のビットごとの足し算が行われる.

1.2.6. 漸近的解析

一つのアルゴリズムを解析するとき, 我々は大きなサイズの問題例に対するパフォーマンスに関心がある. (小さい問題だったら, どんなアルゴリズムでも満足いく時間内に解ける.)

関数 $f: \mathbb{Z}_+ \to \mathbb{Z}_{++}$ と $g: \mathbb{Z}_+ \to \mathbb{Z}_{++}$ に対して,

$$\exists c > 0, \exists \text{ 正整数 } n_0, \forall n \geq n_0 : f(n) \leq cg(n) \tag{1.1}$$

であるときに,

$$f(n) \text{ は} \textbf{オーダー } g(n) \text{ と言い, } f(n) = \mathrm{O}(g(n)) \text{ と表す.}$$

例 1.1: $5n^2 + 3n = \mathrm{O}(n^2)$.
(証明)

$$8n^2 - (5n^2 + 3n) = 3(n^2 - n) = 3n(n-1) \geq 0 \text{ for } n \geq 1$$

であるから, $c = 8, n_0 = 1$ とすれば, 式 1.1 が成り立つ. □

例 1.2: $35 \cdot 2^n + n^3 = \mathrm{O}(2^n)$.
(証明)

$$36 \cdot 2^n - (35 \cdot 2^n + n^3) = 2^n - n^3$$

□

例題 1.3:

$$\sum_{i=0}^{l} a_i n^i = \mathrm{O}(n^l).$$

を証明せよ.

例 1.4: Selection Sort における比較演算の回数の上界は, $\frac{n(n-1)}{2} = \mathrm{O}(n^2)$. Insertion Sort における比較演算の回数の上界も, $\frac{(n+2)(n-1)}{2} = \mathrm{O}(n^2)$.

例題 1.5: 正整数 l, a, b に対して,

$$a \cdot 2^n + bn^l = \mathrm{O}(2^n)$$

が成り立つことを証明せよ.

微積で習った (はず) の, $o(f(x))$ と比べてみよ.

1.2.7. Nearest Neibor Algorithm の計算量の評価 (算術演算モデル)

行 1 は, $3n$ 回の代入
行 2 は, 1 回の代入
行 3 で, 1 回のの代入
行 5∼16 は, $n-1$ 回繰返される. その各繰返しにおいて,
　行 5 は 1 回の代入,
　行 7 は, n 回の比較, 行 8 は, $6(n-1)$ 回の算術演算
　行 9 と行 10 は, $n-1$ 回の代入
　行 14 は 1 回の代入
　行 15 は 1 回の代入
計 $(n-1)(1+n+6(n-1)+(n-1)+1+1) = (n-1)(8n-4)$ の算術演算がある.
　よって, アルゴリズム全体では, $3n + 2 + (n-1)(8n-4) = \mathrm{O}(n^2)$ の算術演算がある.

Algorithm 1 Nearest Neighbor

Require: n 個の点の座標 $(x_1, y_1), \cdots, (x_n, y_n)$.
Ensure: 巡回路.

```
1: 第 i 番目の要素が, (x_i, y_i, mark) であるような配列をつくる. (最初は, どの mark も 0.)
2: j ← 1.
3: (x_1, y_1) の mark を 1 にする.
4: for (k=1; k<=n-1; k++) do
5:    min ← ∞.
6:    for (i=1; i<=n; i++) do
7:       if (x_i, y_i) の mark が 0 then
8:          if (x_i − x_j)^2 + (y_i − y_j)^2 < min then
9:             min ← (x_i − x_j)^2 + (y_i − y_j)^2.
10:            i* ← i.
11:         end if
12:      end if
13:   end for
14:   j ← i*.
15:   (x_j, y_j) の mark を 1 にする.
16: end for
```

1.2.8. 多項式時間アルゴリズム

Nearest Neibor Algorithm や Selection Sort (Insertion Sort も) のように, 計算ステップの上界が, ある k に対して $O(n^k)$ と表されるものは, **多項式時間アルゴリズム**と呼ばれる. 多項式時間アルゴリズムがまだ発見されていない問題は, 解くのが難しい問題と信じられている. しかし, この本で扱う問題のほとんどは, 多項式時間アルゴリズムが知られているような問題たちである.

以降この本では主に算術演算モデルを用いて, アルゴリズムの解析を行なう.

1.3. Further Readings

アルゴリズムの解析については, Sedgewick [1] (邦訳あり) が読みやすい. ソーティングについて詳しく書かれている.

日本語の本では, 浅野・今井 [2] が良い.

例題 1.6: A と B を $n \times n$ 行列とするとき, AB を計算するためには何回の足し算と何回のかけ算が必要か?

例題 1.7: A を $n \times n$ 行列とする. A の行列式 $\det A$ は,

$$\det A = \sum_{\pi} \mathrm{sgn}(\pi) a_{1\pi(1)} a_{2\pi(2)} \cdots a_{n\pi(n)}$$

で定義される. ここで, π は $(1, 2, \cdots, n)$ の置換全体を動く. また, $\mathrm{sgn}(\pi)$ は π の符号を表す. この定義にしたがって, $\det A$ を計算するとどのくらいの足し算とかけ算が必要か?

実は, Gauss の消去法を使うと $O(n^3)$ で計算がすむ. これを示せ.

References

[1] R. Sedgewick: *Algorithms*. Addison Wesley, 1988.

[2] 浅野孝夫, 今井浩: 計算とアルゴリズム. オーム社, 2000 年.