アルゴリズムの設計と解析

北陸先端科学技術大学院大学
情報科学研究科
浅野哲夫
t-asano@jaist.ac.jp

1　計算のモデル

1.1　アルゴリズムとは

アルゴリズムとは，計算機で問題を解くときの手順を記述したものである．記述に曖昧さがなければ計算機言語で記述する必要はないので，自然言語で記述されることが多い．アルゴリズムとは多分に概念的なものであり，具体的なプログラムとの対応を必ずしも明示する必要がないからである．

では，問題を解くための手順を記述したものなら何でもアルゴリズムと呼ぶかと言うと，暗黙のうちに効率が良いと思われるものだけがアルゴリズムの名前で呼ばれているようである．上にも述べたように，アルゴリズムという概念そのものが抽象的なものであり，計算機システムとも計算機言語とも独立な存在であるので，アルゴリズムの効率という概念も抽象的なものにならざるを得ないという背景がある．

1.2　計算のモデル

アルゴリズムの理論では量的な尺度を導入するために，計算機のモデルを設定してから議論を始めるのが通例である．計算機のモデルとしては，最も原始的なチューリング機械から現在の計算機に近いランダムアクセス機械など，様々なものが考えられている．いずれにしても，これらの機械の１回の操作に要する時間を単位時間と考えて，計算を終了するまでに必要な時間によって計算の効率を評価しようというのが基本的な考え方である．もちろん，このような抽象的な機械の１回の操作と現在我々が用いている計算機の１操作には大きな隔たりがある．最も原始的な機械語レベルで考えても，１機械語命令を実行するのに前述の抽象的な機械では多数の操作が必要になり，しかも何回の操作が必要になるかは命令によって異なるという厄介な事情がある．計算時間を正確に評価するためには，モデル計算機と現実の計算機における演算に正確な対応を確立することが必要であるが，その努力は得られる結果との関係において考えるべきものである．特に，ＣＰＵレベルにおいても各種の並列動作が取り入れられている現状を勘案すると，アルゴリズムの計算時間を正確に評価する計算式を求めることは不可能に近いと考えてよい．

2　アルゴリズム比較のための解析

上述のように，アルゴリズムの計算時間を正確に予測する式を求めることは不可能に近い．そこで，考え方を改めて，計算時間を予測するというよりは，２つのアルゴリズムの計算効率を比較する尺度として計算効率を解析しようと考えるのは自然である．では，アルゴリズムの比較において何が本質的な役割を果たすであろう．一つの方法は，多数の例題に対する実際の計算時間を比較するという統計的な手法である．実際，実用性に重きを置く学会ではベンチマークテストと呼ばれる共通の標準データを多数用意して，それらに対する計算時間によってアルゴリズムの比較を行うことも多い．しかし，いずれにせよ限られた数のテストデータに対する計算時間だけに基づく比較が完全でないことは明らかである．

アルゴリズム理論においては，より一般的な尺度として，データ数の増加に対する計算時間の増加の割合を用いることが多い．たとえば，２つのアルゴリズムＡとＢの計算時間 $T(A)$ と $T(B)$ が，データ数 n に対して，$T(A) = 100n + 1000$ と $T(B) = 2n^2 + 5n + 100$ によって与えられるとき，２倍のデータ数に対する計算時間は，アルゴリズムＡでは高々２倍であるのに対して，アルゴリズムＢでは４倍近くになってしまう．この例からも分かるように，アルゴリズムの計算時間をデータ数を変数とする式で表現したとき，係数は無視して，最高の次数の大きさによって比較しようとしているのである．もちろん，計算時間を表す式が一意に決まることは稀であり，実際には計算時間の下限と上限を表す式を用いて効率の比較を行わなければならない．そこで，微妙な差は無視して，明らかに差が認められるかどうかによって比較を行う，というのがアルゴリズム理論の基本的な立場である．

2.1　計算時間解析の例

具体的な例題として次のような問題を考えよう．

最大区間差問題: n 個のデータが配列 $a[0..n-1]$ に蓄えられているとき，区間 $[s,t]$, $0 \le s \le t < n$ に対して定まる差（区間差）$a[t] - a[s]$ を最大にする区間 $[s,t]$ を求めよ．

n 個のインデックスの中から２個を取り出して昇順に並べると区間が定義されるから，全部で高々$n(n+1)/2$ 通りの区間が存在する．それらをすべて列挙し，最大の区間差を求めればよいから，n^2 に比例する時間で十分であることが分かる．しかし，それが最適であろうか？t を区間の終点とするすべての区間の中で区間差が最大になるのは，区間 $[0,t]$ における最小値が区間の始点になるときである．その値を $MIN(t)$ と書くことにしよう．問題は如何に効率良く $MIN(t)$ の値を求めるかである．もちろん，対応する区間の値をすべて調べれば求まるが，それでは効率が悪い．ここで重要なのは再帰的なものの考え方である．すなわち，$MIN(t)$ を求めようとするとき，その直前での値 $MIN(t-1)$ は分かっているから，それをうまく利用するのである．$MIN(t-1)$ は，$[0,t-1]$ の範囲における最小値であるから，これを $[0,t]$ の範囲に拡張するには，$a[t]$ の値と比較するだけでよい．つまり，$MIN(t)$ の値が $MIN(t-1)$ と異なるとすれば，それは $a[t] < MIN(t-1)$ のときに限ることを利用するのである．下に示すアルゴリズムはこの考え方に沿ったものである．

```
msf = 0; min = a[0];
for t = 1 to n − 1{
    if(a[t] − min > msf) {
```

```
        msf = a[t] − min;
    }else if(a[t] < min) min = a[t];
}
output "maximum difference is " msf;
```

このアルゴリズムには一つのループだけが含まれ，ループにおける毎回の操作は高々２回の条件式評価と１回の代入文の実行だけであるから，全体でデータ数 n に比例する程度でしかない．これは最初に述べた n^2 に比例する素朴なアルゴリズムに比べて本質的に優れている．

2.2 計算時間の上界と下界

最大区間差を求める上記のアルゴリズムは，ある意味において最適であり，改善の余地は残っていない．つまり，問題を解くためにはすべてのデータを読み込まなければならないからデータ数に比例する時間は必要であるが，上記のアルゴリズムは更にデータ数に比例する時間で答を求めることができるから，比例係数を無視すれば，これ以上の改善は不可能である．このように，アルゴリズム理論では計算時間を表す式の比例係数や低次の項は無視して，最高次数の項だけで計算時間を表現することが多い．たとえば，上で述べた n^2 に比例するアルゴリズムの計算時間は，高々n^2 に比例する時間で計算を終わるという意味において，計算時間は $O(n^2)$ であるという．また，データを読み込むのに必ず n に比例する時間が必要になるから，後の処理を如何に工夫しようと n に比例する時間は必要になるという意味において，計算時間は $\Omega(n)$ であるという．このように比例係数を隠してしまうことから，アルゴリズムの解析は信用が置けないと言われることが多いが，前にも述べたように，計算時間を予測する式を求めようとしているのではなく，アルゴリズムの性能を本質的な意味で比較するための道具として解析を使おうとしていることを忘れてはならない．

上に述べたように，アルゴリズムの計算時間をデータ数の関数として表したとき，その正確な表現を得ることは難しいから，関数値の上界と下界によって表現するのが一般的である．n 個のデータを大小順に並び替えるソーティングの一方法である挿入法について考えてみよう．この方法では，データを一つずつ取り出しては既にソート順に並べられた列の中に挿入していく．このとき，正しい挿入位置を逐次探索の要領で求めるのが一般的である．既にソートされた列が入力として与えられた場合には，毎回１回の比較で正しい挿入位置がわかるから，データ数 n に比例する時間で処理を終わることができるが，逆順に並んでいる場合には，毎回最後まで比較を行わなければならないので，n^2 に比例する時間がかかってしまうことになる．n に比例する時間で処理を終わる場合もあるが，n^2 時間かかる最悪の入力例もあるから，挿入法の計算時間は $O(n^2)$ ということができる．

2.3 計算問題の計算複雑度

では，ソーティング問題そのものの計算困難度はどのように評価すべきであろうか．一般にアルゴリズムの計算複雑度の解析よりも，計算問題の計算複雑度の解析の方が格段に難しい．まず，ソーティング問題の計算複雑度の上界について考えよう．挿入法はどんな入力に対しても $O(n^2)$ の時間で処理を終わるから，ソーティング問題の上界は $O(n^2)$ であるということができる．しかし，ヒープソートと呼ばれる方法では，どんな入力に対しても $O(n \log n)$ の時間で

処理を完了できることが知られている．したがって，ソーティング問題の（より良い）上界は $O(n \log n)$ である．一方，どんなアルゴリズムを使おうと，そのアルゴリズムでソートを行うのに $n \log n$ に比例する回数の比較が必要になる入力例が存在することが知られている．この意味において，ソーティング問題の下界は $\Omega(n \log n)$ であると言う事ができる．ソーティング問題の場合には上界と下界が一致するので，そのような場合には Θ という記号を用いて，ソーティング問題の計算複雑度は $\Theta(n \log n)$ であると言う．

2.4 多項式と指数関数の相違

アルゴリズム理論では最悪の場合の計算時間を解析することが多いが，何と言っても統計数学の知識を駆使しなければならない平均の場合の解析に比べて，最悪の場合の解析は簡単である．また，最悪の場合の計算時間を評価できれば，妥当な時間内に解ける見込みがあるかどうかをある程度まで判定することができる．アルゴリズム理論では，解ける見込みがあるかどうかの境界線を，最悪の場合の計算時間がデータ数の多項式で抑えることができるかどうかに置いている．データ数を n とするとき，n^2 も n^{100} も共に多項式であるから，$O(n^{100})$ の時間がかかろうと，解けるものと考えるのである．逆に，計算時間が a^n $(a > 1)$ のような指数関数で表される場合には，a がどんなに 1 に近くても解けないものとするのである．実際問題としてはデータ数 n の値によっては指数関数時間かかるアルゴリズムは 1 秒で終わるが，多項式時間のアルゴリズムでは 1000 年以上もかかってしまうということも考えられるが，先にも述べたように，基本的にはデータ数の増加に対する計算時間の増加の割合に注目してアルゴリズムを分類しようとしていることに注意されたい．

上に述べたように，アルゴリズム理論では，問題が解けるとは，その問題をデータ数の多項式時間で解くアルゴリズムが存在することを言う．したがって，解けない問題とは，そのような多項式時間のアルゴリズムが存在しないものであると言うことができる．しかし，多項式時間のアルゴリズムが存在しないと言い切るのは非常に困難であり，そこに理論の難しさがある．

易しい問題と計算困難な問題を区別するための概念としてＮＰ困難性がある．非常に長い出力を要求する問題を除外するために，YES/NO の区別だけを要求するいわゆる判定問題を考えよう．たとえば，多数の与えられた都市をすべて訪問する巡回路の中で総距離が最小のものを求める問題は巡回行商人問題として良く知られているが，これに対する判定問題は，任意に与えられた長さ L に対して，総距離が L 以下の巡回路があるかどうかを問う問題である．この問題に対して解の候補として一つの巡回路を与えたとき，それが YES の答を与える解であるか，すなわちその巡回路の総距離が L 以下かどうかは巡回路の長さに比例する時間で判定することができる．このように，解の候補が与えられたとき，それが YES を与える解かどうかを多項式時間で判定するアルゴリズムが存在するとき，その問題はクラス $\mathcal{NP}$ に属するという．一方，YES を与える解が存在するかどうかを多項式時間で判定するアルゴリズムが存在するときには，その問題はクラス $\mathcal{P}$ に属するという．定義より明らかにクラス $\mathcal{P}$ に属する問題はクラス $\mathcal{NP}$ にも属するから，後者の方がより広いクラスであると予想されている．

クラス $\mathcal{P}$ と $\mathcal{NP}$ に差があるかどうかは未解決であるが，殆どの計算機科学者は差があるものと信じている．さらに，その差の部分に位置する問題はクラス $\mathcal{NP}$ の中でも難しい問題（ＮＰ完全問題）であるが，そのような問題に対しては多項式時間のアルゴリズムは存在しないであろうと予想されている．

3　アルゴリズム設計技法

問題ごとに効率の良い解き方は異なるから，アルゴリズムの設計も問題ごとにならざるを得ないという側面があることは事実であるが，まったく諦めて思いつきに走るのは早計に過ぎる．アルゴリズムが理論として確立してきたことの背景には，かなり広範囲に適用可能なアルゴリズム設計技法が確立されてきたことがある．再帰を用いるのもアルゴリズム設計技法の一つと言えなくもないが，再帰以外にも実に様々な技法が知られている．したがって，問題が与えられてアルゴリズムを設計しようとするとき，最も科学的な態度とは，まず既存のアルゴリズム設計技法に基づいて効率の良いアルゴリズムが設計できるかどうかを調べ，どの技法を用いても不可能なら，全く新しいアルゴリズムを考えるべきである．

例をあげて説明しよう．ここでは，平面上に点集合 S が与えられたとき，S の点を包み込む最小の凸多角形（S の凸包）を求める問題について考えよう．これは計算幾何学の代表的な問題であり，従来から多数のアルゴリズムが報告されている．アルゴリズムを設計する上でまず基本となる観察は，凸包の頂点は必ず S の点だということである．ある点 $p \in S$ が凸包の頂点となるのは，点 p を通って凸包に接線が引けるときである．そのような接線を引けたとすると，接線で分割された一方の半平面に S の残りの点がすべて含まれている．この性質を利用すると，$O(n \log n)$ の時間で凸包上の点かどうかを判定することができる．この方法で凸包の頂点をすべて列挙できれば，後はそれらの点の重心からの偏角によってソートすれば，求める凸多角形を構成することができる．

上に述べた方法は $O(n^2 \log n)$ の時間を要する．効率を改善するための最も一般的な方法は分割統治の考え方を適用することである．すなわち，元の点集合 S をほぼ同じサイズの部分集合に分割した上で，それぞれに対して再帰的に凸包を構成し，最後に得られた 2 つの凸包の共通接線を求めることにより一つの凸包にまとめるのである．最後の統合の操作を $O(n)$ 時間で実行することができるなら，全体の計算時間は $O(n \log n)$ となり，かなり効率が改善できたことになる．実際に統合操作を $O(n)$ 時間で行うことができるが，ここでは紙数の関係で詳しくは説明しない．

分割統治法以外にも，動的計画法，逐次構成法，走査法，スケーリング法，枝刈り探索法，パラメトリック探索法，線形計画法など，実に多様なアルゴリズム設計技法が知られている．これらの技法をすべて試して，どれも適用できないと分かれば始めてオリジナルなアルゴリズムをトライするというのが正しい態度であろう．