『アルゴリズム設計マニュアル (上)』要点まとめ

ぱろすけ

2013 年 3 月 17 日

0　この文書について

専ら自身の学習のために趣味で作成したものです。これさえ読み返せば内容が思い返せる（必要に応じて参照できる）ことを目指します。

1　アルゴリズム設計への導入

挿入ソートは、ひとつの要素から始め、まだソート済み列に加えられていない要素をひとつずつ適切な位置に挿入する。アルゴリズムの一例。

1.1　ロボットツアーの最適化

要するに巡回セールスマン問題。近い点に移動し続ける**最近接点**ヒューリスティックも、最も近いチェーンの端点を結んでゆく**最近接ペア**も、うまくいかない。最適解を求めるにはすべてのパターンを試す他ないが現実的な計算量ではない。

1.2　適切な仕事の選択

映画撮影スケジューリング問題。適切なヒューリスティックにより解ける例。

1.3　正しさの論証

1.3.1　アルゴリズムの表現

アルゴリズムは自然言語、擬似コード、プログラミング言語で表し得るが、アイディアが明確に現れるような表現法を選ぶこと。

1.3.2　問題と性質

一般化映画撮影スケジューリング問題には効率の良い解はない。問題を正しく形式化することが重要。定義が曖昧に過ぎたり、複雑すぎる目標を設定したりしやすい。

1.3.3 間違いを実証する

反例は**立証可能性**と**単純性**を持つべきである。そのためには小さな例を網羅的に考え、同点に持ち込んだり極端なものを探して弱点を見つけると良い。

1.3.4 帰納法と再帰

帰納法は証明に便利だよと書いているだけ。

1.3.5 総和

等差級数の場合、n 次式の和は $n+1$ 次式になる。等比級数は比が 1 より小さいとき収束するが、これはアルゴリズムの解析上便利なことがある。

1.4 問題のモデル化

モデル化とは応用問題を厳密な形式に変換すること。

1.4.1 組合せオブジェクト

用語と登場シーンの説明。順列、部分集合、木、グラフ、点、多角形、文字列。きちんと定義された構造とアルゴリズムの言葉でモデル化するよう心がける。

1.4.2 再帰的なオブジェクト

一部を削除しても同じものである。順列、部分集合、木、グラフ、点、多角形、文字列、すべて再帰的なオブジェクトといえる。

1.5 設計奮戦記について

1.6 ボクの設計奮戦記：超能力のモデル化

モデル化を正確に行うことは大事だよという話。いきなり間違ったモデル化をされて混乱する。

2 アルゴリズム解析

2.1 計算の RAM モデル

Random Access Machine は計算量を考えるときの単純なコンピュータのモデル。メモリは無限に持つ。アルゴリズムの話ではだいたい有用。

2.1.1 最悪、最良、そして平均の計算量

基本的には最悪計算量だけ考えれば良い。

2.2 ビッグオー記法

実際の計算量はコードに依存し、グラフのでこぼこも多い。しかし実用的にはオーダーで考えれば良い。$f(n) = O(g(n))$ は $c \cdot g(n)$ が $f(n)$ の上界であることを意味する。$f(n) = \Omega(g(n))$ は $c \cdot g(n)$ が $f(n)$ の下界であることを意味する。$f(n) = \Theta(g(n))$ は $c_1 \cdot g(n)$ が $f(n)$ の上界で $c_2 \cdot g(n)$ が $f(n)$ の下界であることを意味する。

2.3 増加率と支配関係

2.3.1 支配関係

よく現れるクラスと支配関係の話。特に難しい話はない。

2.4 ビッグオーを使いこなす

特に難しい話はない。

2.4.1 関数の加算

2.4.2 関数の乗算

2.5 効率に関する議論

2.5.1 選択ソート

選択ソートは残りの要素のうち最も小さいものをソート済み列に加える。計算量は $\Theta(n^2)$ になる。簡単。

2.5.2 挿入ソート

挿入ソートの計算量は $O(n^2)$ になる。簡単。

2.5.3 文字列パターンマッチング

単純な文字列パターンマッチングについての計算量の細かな議論。落ち着けば何も難しいことはない。

2.5.4 行列の乗算

3 乗のアルゴリズムであることは明らか。

2.6 対数とその応用

対数が現れる例をいくつか示す。

2.6.1 対数と 2 分探索

2.6.2 対数と木

2.6.3 対数とビット

2.6.4 対数と乗算

任意の a と b について $a^b = \exp(b \ln a)$ と計算できる。

2.6.5 高速な指数計算

大きな n について a^n を計算するときは $a^n = a(a^{[n/2]})^2$ とすると高速。これは分割統治法の一種。

2.6.6 対数と総和

調和数について $H(n) = \sum_{i=1}^{n} 1/i \sim \ln n$ が成り立つ。これは計算量の解析に使えることがある。クイックソートなど。

2.6.7 対数と刑事裁判

合衆国では詐欺罪の損害額と量刑は対数的な関係にある。だから一気にたくさん騙そう。

2.7 対数の性質

アルゴリズムの解析では、定数倍の違いにしかならないため、対数の底が何であるかは無視して良い。

2.8 ボクの設計奮戦記：ピラミッドの謎

ナップザック問題をスパコンで力ずくで解こうとしたけど無理だったけれど、アルゴリズムを工夫したら手元のコンピュータでも容易に解けた、という話。アルゴリズム的には『プログラミングコンテストチャレンジブック』の最初に出てきたものと酷似している。

2.9 高度な解析

高度な話題であり、上巻には他に現れないが、下巻で有用であるもの。

2.9.1 難解な関数

逆アッカーマン関数は、きわめてゆっくり増加する関数であり、かなり大きな n についても $\alpha(n) < 5$ である。$\log n / \log\log n$ は次数が $\log n$ である木の高さとして現れる。

2.9.2 極限と支配関係

$n! \gg c^n \gg n^3 \gg n^2 \gg n^{1+\varepsilon} \gg n\log n \gg n \gg \sqrt{n} \gg$
$\log^2 n \gg \log n \gg \log n/\log\log n \gg \log\log n \gg \alpha(n) \gg 1$

3 データ構造

コンテナ、辞書、優先順位付きキューについて、配列とリストでの実装を詳説する。

3.1 連続データ構造と連結データ構造

連続データ構造とはメモリ上で連続しているデータ構造であり、配列、行列、ヒープ、ハッシュ表などが含まれる。連結データ構造とはいくつかのメモリ領域がポインタで結ばれる構造であり、リスト、木、隣接リストなどが含まれる。

3.1.1 配列

配列はメモリの局所性により高速キャッシュメモリを活かすことができる。配列に対し合計 n 回の挿入を行うとして、動的配列で実現するとその総移動回数は $2n$ 回であるから、この計算量はなんと $O(n)$ となり、静的配列の場合と等しい。

3.1.2 ポインタと連結構造

C 言語によるリスト構造の実現の話。

3.1.3 比較

連結リストはメモリを食うが挿入と削除は簡単。リストも配列も再帰的なオブジェクトであることに留意すべき。

3.2 スタックとキュー

コンテナとはデータを内容に無関係に出し入れするデータ構造のこと。スタックでもキューでも平均待ち時間は同じ。

3.3 辞書

辞書の概要と、実装の違いによる基本操作の計算量の違いについて述べている。長いけれども特に難しい点はない。

3.4 2 分探索木

探索が高速でかつ更新も柔軟なものの例としての 2 分探索木。ある頂点について、左部分木のすべての頂点はその頂点より小さく、右部分木のすべての頂点はその頂点より大きい。このような木は複数の形で存在し得る。

3.4.1 2 分探索木の実装

木の高さを h とすると、ある要素の探索は $O(h)$、最小要素の探索も $O(h)$ である。すべての頂点と辺を横断する際は、各頂点について左部分木、自身、右部分木の順で処理すれば良く、$O(n)$ で済む。挿入については、可能な場所はただひとつであり、それは探索により見つかり、定数時間で追加を行え、$O(h)$ である。削除の場合は、その頂点をその次の値で置き換えると考え、右部分木の最小値をそこに割り当てる。

3.4.2 2 分探索木はどれほどよいか？

基本操作はすべて $O(h)$ で行え、挿入順がランダムの場合は平均の木の高さは $O(\log n)$ になる。

3.4.3 平衡探索木

高さが常に $O(\log n)$ になる平衡探索木が提案されている。詳細についてはここでは触れられない。平衡探索木を使うことで $O(n \log n)$ の様々なソートが提案される。

3.5 優先順位付きキュー

優先順位付きキューもいろいろな実装があるが、最小要素の位置を別に持つことでどのような実装でも定数時間で見つけることができるようになる。ただしこのときは削除に余計な時間がかかる。

3.6 ボクの設計奮戦記：三角形を連ねる

きちんとデータ構造を選びましょうねという話。

3.7 ハッシングと文字列

3.7.1 衝突の回避

ハッシュ値が衝突することがある。連鎖法はハッシュの各要素を隣接リストとすることで回避する。開アドレス法では、既に埋まっていた場合、その次の位置に要素を配置する。探索するときは、本来あるべき位置から連続してある要素をすべてチェックする。削除する場合は、そのあとに連続する要素をすべて再挿入する。ぐぐると、削除済みというフラグをつければ良いという記述もある。

3.7.2 ハッシングを用いた効率的な文字列マッチング

Rabin-Karp のアルゴリズム。部分文字列パターンマッチング問題において、テキスト文字列のすべての位置からの文字列をハッシュ化しておくことにより、ハッシュ表をチェックする問題になる。n 文字と m 文字のマッチングでは $O(n)$ の表を構成することになり、ハッシュの計算に $O(m)$ かかる。しかし、ハッシュは実は計算結果を活かして定数時間で行うことが可能である。それゆえ、だいたい $O(n+m)$ 時間で走らせることが可能になる。

3.7.3 ハッシングによる重複検出

ハッシングは、重複文書の検出や、盗作の検出、ファイルのハッシュ化による暗号化などの応用がある。ハッシングはランダム化アルゴリズムにおける基本的なアイディアで、$\Theta(n \log n)$ や $\Theta(n^2)$ となるような問題にも線形時間のアルゴリズムをもたらす。

3.8 特定目的のデータ構造

文字列にはサフィックス木とサフィックス配列があり、パターンマッチングに用いられる。幾何的データ構造には kd 木があり、高速な探索を可能にする。集合データにはビットベクトルがある。これらは下巻で詳しく述べられる。

3.9 ボクの設計奮戦記：数珠つなぎ

2 分探索木で探す、ハッシュを用いる、サフィックス木、圧縮サフィックス木、と改良することで問題を解けたよという話。サフィックス木の詳細は下巻。

4　ソートと探索

4.1　ソートの応用

配列中に任意の要素 k が現れる回数を知る良い方法は、配列をソートし、$k-\varepsilon$ と $k+\varepsilon$ の位置を探すことである。また、**凸包**を構成する良い方法は、ある軸ですべての点をソートし、その順で凸包に点を加えてゆくことである。新たな点は必ず凸包をなし、かつ削除すべき点は新たな点によって作られる領域にあるため効率が良い。ふたつの配列の共通要素を持つか判定するには小さい方をソートしたのち大きい方の各要素を探すことが考えられるが、実用的にはハッシュを用いるのが最良である。

4.2　ソートの実際

もとの並びの相対的な順序を崩さないものを**安定ソート**と呼ぶ。

4.3　ヒープソート：データ構造による高速ソート

選択ソートは素直に実装すると $O(n^2)$ だが、最小要素の検索にヒープや平衡 2 分木を用いると $O(n\log n)$ となる。ヒープソートは、実際のところはヒープを用いた選択ソートに過ぎない。

4.3.1　ヒープ

ヒープは 2 分木であり、min ヒープでは親は 2 つの子より小さな値を持つ。これは配列を用いることにより効率よく実現できる。ヒープの高さは常に $[\lg n]$ である。

4.3.2　ヒープを構成する

ヒープに要素を追加するときは、配列の空きの最初の要素に配置する。もしその値が親より小さければ親と要素を入れ替える。これは再帰的に行う。

4.3.3　最小要素を取り出す

最小要素は根であるから単にそれを参照すれば良い。空きはもっとも右端にある要素で埋める。もしこれが 2 つの子より大きければそれと入れ替える。これも再帰的に行う。ヒープソートは**インプレースソート**であり、ソートする要素を格納する配列以外に余分なメモリを使わない。

4.3.4　ヒープの高速な構成法

ひとつずつ要素を加えることでヒープを構成するのではなく、すべての要素を配置してからヒープ内を整理するほうが効率が良い。子を持つものだけ整合性をチェックすればよく、かつそのとき木は概ね低いからである。これはほぼ線形時間で済む。

ヒープの k 番目に小さい要素が x 以上かを判定したいとき、これを x より小さいものが k 個以上かと読み替える。そして、x より小さい節を次々と訪問し、その数を数える。これが k を超えたら終了する。訪問する節の数は親と子 2 つで高々 $3k$ なのでこれは $O(k)$ のアルゴリズムである。頭良い。

4.3.5 逐次挿入によるソート

挿入ソートのような方法を**逐次挿入**と呼ぶ。これもデータ構造に平衡 2 分木を用いると $O(n \log n)$ で済む。

4.4 ボクの設計奮戦記：飛行機のチケットをくれないか

とりあえず飛行機のチケットの値段はものすごい複雑な感じになってることはわかった。

4.5 マージソート：分割統治法によるソート

マージソートはランダムアクセスに頼ることがないのが利点である。一方で、配列をソートするための補助バッファが必要となることが欠点である。部分配列はキューにコピーすると良い。

4.6 クイックソート：ランダム化によるソート

実装について書いてあるのでさすがに書けるようにしとくとよさそう。

4.6.1 直観：クイックソートの平均の場合

ランダムな 2 分探索木では n 回の挿入の後の平均の高さは $2 \ln n$ である（証明なし）。これは $1.386 \lg n$ なのでそれほど高くはない。ゆえに計算量は $O(n \log n)$ であるといえる。

4.6.2 ランダム化アルゴリズム

決定的なクイックソートのアルゴリズムでは必ず最悪の入力例が存在する。そこで、入力を予め**ランダム化**しておくことで計算量の期待値を $\Theta(n \log n)$ とできる。ランダム化はサンプリングやハッシング、探索などでも用いられる。

4.6.3 クイックソートは本当にクイック？

適切に実装されたクイックソートはマージソートやヒープソートよりたいてい 2,3 倍早い、らしい。最も内側のループでの操作がより単純だから。

4.7 分配ソート：バケットを用いたソート

名前をソートするとき、まず頭文字別に分け、それぞれをソートすると効率的である。これを**バケットソート**あるいは**分配ソート**といい、データが均一のときに強い。実装などの細かい話はなし。

4.7.1 ソートの下界

ソートは $n!$ の順列のそれぞれについて異なる動きをしなければならない。そのような木を考えると高さは $\Theta(n \log n)$ になる。これがソートの下界である。このように自明でない下界が得られるアルゴリズムは少ない。

4.8 ボクの設計奮戦記：スキーナの抗弁

非常に大きなデータをソートするときはいろいろ話が違うよねという話。

4.9 2 分探索と関連アルゴリズム

4.9.1 出現の数え上げ

ソートされた配列中で要素 k の出現回数を求める場合、2 分探索を改造して他の要素との境界を求めるようにすると高速に求まる。

4.9.2 片側 2 分探索

配列の要素数の上限がわからないとして、配列中の要素の転換点を求めたい。その場合は $A[1], A[2], A[4], A[8], ...$ と探索して 2 分探索の窓を決定すると早い。

4.9.3 平方根とその他の根

2 分探索で平方根や方程式の解を求められるよねという話。

4.10 分割統治法

分割統治法は、問題を（たとえば）半分に分割しそれぞれを解く。

4.10.1 再帰式

4.10.2 分割統治法の再帰式

Strassen は $n \times n$ 行列の積を $n/2 \times n/2$ 行列の積 7 つに分割することにより、積を $O(n^{2.81})$ で計算するアルゴリズムを得た。

4.10.3 分割統治法の再帰式を解く

分割統治法の再帰式 $T(n) = aT(n/b) + f(n)$ を簡単に解く**マスター定理**の紹介。ぐぐって使えれば良い。ここでは直感的な解釈も示されている。