データ構造とアルゴリズム

05 分割統治法

宮本 裕一郎
miyamoto あっと sophia.ac.jp

上智大学 理工学部 情報理工学科

目次

分割統治法とは

- 再帰アルゴリズムのうち，特に，与えられた問題を小さな問題（部分問題，あるいは小問題）に分割して解き，それらを合わせて元の問題の解を得るアルゴリズムを分割統治法（divide and conquer method）という．
- 分割統治法の考え方を適用することによって，時間複雑度の小さいアルゴリズムを設計できることがある．

並べ替え

- お互いに比較可能な n 個の要素からなる列が与えられたとき，それらを昇順（あるいは降順）に並べ替えることを並べ替え（sorting）という．
- 以降では，簡単のため実数の昇順並べ替えを考える．

問題 (昇順並べ替え【再掲】)

入力　数値の列 $X = (x_1, x_2, \ldots, x_n)$.

出力　数値列 $Y = (y_1, y_2, \ldots, y_n)$，ただし $y_1 \leq y_2 \leq \cdots \leq y_n$ かつ X から Y への全単射が存在.

例

- $(2,3)$ が入力として与えられたら $(2,3)$ を出力する.
- $(5,4)$ が入力として与えられたら $(4,5)$ を出力する.
- $(3,4,2)$ が入力として与えられたら $(2,3,4)$ を出力する.

さまざまな並べ替えアルゴリズム

並べ替えは非常に基本的な問題であり，多くのアルゴリズムが知られている．以下に代表的と思われるものを記す．

- バブルソート
- 選択ソート
- 挿入ソート
- マージソート
- ヒープソート
- 基数ソート
- クイックソート

ここでは，非常に単純なアルゴリズムと思われる，バブルソート，選択ソートを紹介した後に分割統治法の代表例であるマージソートを紹介する．それ以外は後々紹介する．

バブルソートの直感的記述

- 隣り合う値が昇順になっていなければ入れ替える.
- 入れ替えの必要がなくなるまでそれを繰り返す.
- それがバブルソート（bubble sorting）である.
- ただ漫然と「入れ替えの必要がなくなるまで」繰り返すのは要領が悪い.
- より具体的には,
  - 1 番目と 2 番目を見比べる，必要があれば入れ替える.
  - 2 番目と 3 番目を見比べる，必要があれば入れ替える.
  - ……
  - $n-1$ 番目と n 番目を見比べる，必要があれば入れ替える.

  とやった後で
  - 1 番目と 2 番目を見比べる，必要があれば入れ替える.
  - 2 番目と 3 番目を見比べる，必要があれば入れ替える.
  - ……
  - $n-2$ 番目と $n-1$ 番目を見比べる，必要があれば入れ替える.

  と繰り返していけば良い．1 回目の繰り返しで最大の値は n 番目に来ているはずなので，2 回目の繰り返しではその一つ手前まで見比べれば十分であることに注意する.
- このようにして $n-1$ 回繰り返せば十分である.

バブルソート

- バブルソートを BubbleSorting として関数っぽく以下に記す．
- BubbleSorting では，変数の参照，変数への代入，実数の大小比較を基本演算と仮定している．
- BubbleSorting の時間複雑度は $O(n^2)$ である．
- BubbleSorting は入力列以外に大きな記憶領域を必要としないアルゴリズムになっている．このようなアルゴリズムを in-place アルゴリズムという [1]．

BubbleSorting$(X(=(x_1, x_2, \ldots, x_n)))$:

Step 1　それぞれの $i \in (1, 2, \ldots, n-1)$ に関して以下を行う．

　　Step 1-1　それぞれの $j \in (1, 2, \ldots, n-i)$ に関して以下を行う．

　　　　Step 1-1-1　もし $x_j > x_{j+1}$ ならば，
　　　　x_j の値と x_{j+1} の値を入れ替える．

Step 2　X を出力して終了する．

[1] 逆に，入力データ以外に，計算過程で，大きな記憶領域を必要とするアルゴリズムを out-of-place アルゴリズムという．

選択ソートの直感的記述

- 与えられた数値列のうち，1 番小さい値の要素を「選択」して取り出し，出力用リストの先頭に入れる.
- 以降，「残った数値列のうち 1 番小さい値の要素を選択して取り出し，出力用リストの最後尾に加える」という操作を繰り返す.
- これが選択ソート（selection sort）である.
- ここでは話の簡単のため，出力用リストに要素を加えるという out-of-place アルゴリズムとして記述した.
- 「選択した要素を数値列の前の方と入れ替える」という具合に適切にカスタマイズすれば，本質的にはそのままで，選択ソートは in-place アルゴリズムとして容易に記述できる.
- せっかく前回学習したので，次ページでは選択ソートを再帰アルゴリズムとして記述する.

選択ソート

- （再帰表現を用いた）選択ソートを SelectionSorting として関数っぽく以下に記す．
- SelectionSorting では，変数の参照，変数の代入，実数の大小比較，集合差，列の連結を基本演算としている．
- SelectionSorting の時間複雑度は $O(n^2)$ である．
- 再帰を用いたら速くなるというものでもない．
- 選択ソートは繰り返しで表現しても大差ない．

SelectionSorting$(X(= (x_1, x_2, \ldots, x_n)))$:

Step 1 もし $n = 1$ ならば，(x_1) を出力して終了する．

Step 2 もし $n > 1$ ならば以下を行う．

Step 2-1 添字 i を，$x_i = \min(x_1, x_2, \ldots, x_n)$ をみたすものとする．

Step 2-2 $(x_i) \circ \text{SelectionSorting}((x_1, x_2, \ldots, x_n) \setminus (x_i))$ を出力して終了する．ここで $\setminus$ は集合の差を，$\circ$ は列の連結を表す二項演算子である．

昇順の数値列の統合

- 次に，マージソートを紹介する．
- マージソートそのものの説明の前に，そこで部分問題として現れる，昇順の数値列の統合問題を考える．

問題 (昇順の数値列の統合)

入力 2つの数値列 $X = (x_1, x_2, \ldots, x_k)$, $Y = (y_1, y_2, \ldots, y_l)$

- ただし $x_1 \leq x_2 \leq \cdots \leq x_k$ かつ $y_1 \leq y_2 \leq \cdots \leq y_l$

出力 $Z = (z_1, z_2, \ldots, z_{k+l})$

- ただし，$z_1 \leq z_2 \leq \cdots \leq z_{k+l}$ かつ $X \cup Y$ から Z への全単射が存在

例

入力 $X = (2, 3, 5, 10)$, $Y = (1, 6, 7, 13)$

出力 $Z = (1, 2, 3, 5, 6, 7, 10, 13)$

マージアルゴリズムの直感的記述

- 昇順の列を 2 つ合わせて，1 つの昇順の列にする場合に最も採用されそうな方法を以下に直感的に記述する．
- 出力用リストとして空のリストを用意する．
- それぞれの列の先頭のどちらかが最小の値のはずなので，その小さい方（より正確には大きくない方）を取り出し，出力用リストの最後尾に加える．
- それぞれの列は昇順であることから，先頭を 1 つ取り出した後でも「それぞれの列の先頭のどちらかが，残った中で最小の値のはず」という事実は変わらない．
- よって「それぞれの列の先頭のどちらか小さい方（より正確には大きくない方）を取り出し，出力用リストの最後尾に加える」という操作を繰り返せばよい．
- これをマージアルゴリズムとよぶことにする．次ページにマージアルゴリズムを再帰アルゴリズムとして記述する．

マージアルゴリズム

- 昇順の列の統合問題の解法であるマージアルゴリズムを Merge として関数っぽく以下に記す．
- マージアルゴリズムは繰り返しでも再帰でもどちらで記しても大差ないが，ここでは再帰を用いて記す．
- Merge では，変数の参照，変数への代入（列の要素の取り出し，列の連結），実数の大小比較，アルゴリズムの呼び出しを基本演算とする．
- Merge の時間複雑度は $O(k+l)$ である．

$\mathtt{Merge}((x_1, x_2, \ldots, x_k), (y_1, y_2, \ldots, y_l))$:

Step 1 もし $k = 0$ ならば，$(y_1, y_2, \ldots, y_l)$ を出力して終了する．

Step 2 もし $l = 0$ ならば，$(x_1, x_2, \ldots, x_k)$ を出力して終了する．

Step 3 もし $x_1 < y_1$ ならば，
$(x_1) \circ \mathtt{Merge}((x_2, \ldots, x_k), (y_1, \ldots, y_l))$ を出力して終了する．

Step 4 もし $x_1 \geq y_1$ ならば，
$(y_1) \circ \mathtt{Merge}((x_1, \ldots, x_k), (y_2, \ldots, y_l))$ を出力して終了する．

マージソート

- マージアルゴリズムを利用したマージソート（merge sorting）を MergeSort として関数っぽく以下に記す．ここでは再帰アルゴリズムとして記述する．

MergeSort$((x_1, x_2, \ldots, x_n))$:

Step 1 もし $n = 1$ ならば，(x_1) を出力して終了する．

Step 2 もし $n > 1$ ならば，
MergeSort$(\text{MergeSort}((x_1, \ldots, x_{\lfloor n/2 \rfloor})), \text{MergeSort}((x_{\lfloor n/2 \rfloor + 1}, \ldots, x_n)))$
を出力して終了する．

- MergeSort の時間複雑度は……
  - MergeSort$((x_1, \ldots, x_n))$ の時間複雑度を $f(n)$ とする．
  - $f(n) = \begin{cases} O(1) & (n = 1) \\ f(\lfloor n/2 \rfloor) + f(\lceil n/2 \rceil) + O(n) & (n > 1) \end{cases}$ である．
  - 漸化式を解くと，$f(n) = O(n \log n)$ であるとわかる．（→ 演習問題）

並べ替えに関するコメント

- 参照，比較，代入しか使えない状況での並べ替えの
  時間複雑度の下限は $\mathrm{O}(n \log n)$ であることが知られている．
  （並べ替えの時間複雑度の下限の詳細は，ここでは扱わない．）
- よってマージソートは，時間複雑度の意味で，最良のアルゴリズム
  （の１つ）である．

行列の積

問題 (正方行列の積)

入力 $n \times n$ 行列 $X(= (x_{ij}))$ と $Y(= (y_{ij}))$

出力 $n \times n$ 行列 $Z = XY$

- 行列の積の定義より，Z の要素を z_{ij} とすると，
  $z_{ij} = \sum_{k=1}^{n} x_{ik} \cdot y_{kj}$ である．
- この定義を用いてそのまま各 z_{ij} を計算すると，
  正方行列の積の時間複雑度は $O(n^3)$ である．

Strassen のアルゴリズム

以下では，話を簡単にするために，n が 2 のべき乗の場合を考える．与えられた $n \times n$ 行列 X, Y を，$n/2 \times n/2$ 行列 A, B, C, D, E, F, G, H を用いて

$$X = \begin{pmatrix} A & B \\ C & D \end{pmatrix}, \quad Y = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

と表現する．Strassen のアルゴリズム `Strassen` を以下に記す．

$\texttt{Strassen}(X(= \begin{pmatrix} A & B \\ C & D \end{pmatrix}), Y(= \begin{pmatrix} E & F \\ G & H \end{pmatrix}))$

Step 1　もし X, Y が 1×1 行列ならば，XY を定義通りに計算して出力して終了する．

Step 2　もし X, Y が 1×1 行列でないならば以下を行う．

- Step 2-1　P_1 に $\texttt{Strassen}(A, (F - H))$，$P_2$ に $\texttt{Strassen}((A + B), H)$，$P_3$ に $\texttt{Strassen}((C + D), E)$，$P_4$ に $\texttt{Strassen}(D, (G - E))$，$P_5$ に $\texttt{Strassen}((A + D), (E + H))$，$P_6$ に $\texttt{Strassen}((B - D), (G + H))$，$P_7$ に $\texttt{Strassen}((A - C), (E + F))$ を代入する．
- Step 2-2　$\begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{pmatrix}$ を出力して終了する．

Strassen のアルゴリズムの時間複雑度

- `Strassen` の時間複雑度は……
  - `Strassen`(X, Y) の時間複雑度を $f(n)$ とする．
  - $f(n) = \begin{cases} O(1) & (n = 1) \\ 7f(n/2) + O(n^2) & (n > 1) \end{cases}$ である．
  - 上記の漸化式より，$f(n) = O(n^{\log_2 7}) = O(n^{2.807})$ である．
    （→ 演習問題）
- 一般に，n が 2 のべき乗と限らなくても，Strassen のアルゴリズム（の考え方）は適用できて，その時間複雑度は $O(n^{\log_2 7})$ である．

その他の代表的な分割統治法の例

- n ビットの 2 進数同士の積は，普通の筆算の方法で計算すると時間複雑度が $\mathrm{O}(n^2)$ であるが，分割統治法を用いると時間複雑度を $\mathrm{O}(n \log n)$ にできる．
- Euclid 空間内の n 点が指定されたとき，最近点対（のうちの 1 つ）を見つける問題を最近点対問題という．単純に全点対を吟味すると時間複雑度は $\mathrm{O}(n^2)$ であるが，分割統治法を用いると時間複雑度を $\mathrm{O}(n \log n)$ にできる．
- n 個の要素からなる集合の中央値（median）の計算は，並び替えをした後に中央値を見つけると計算複雑度が $\mathrm{O}(n \log n)$ となるが，分割統治法をうまく用いると時間複雑度を $\mathrm{O}(n)$ にできる．
- 離散フーリエ変換（Discrete Fourier Transformation，DFT）を計算するためのアルゴリズムである高速フーリエ変換（Fast Fourier Transformation，FFT）は分割統治法の代表例である．フーリエ変換には様々な工学的応用がある．また，量子計算において有名な，素因数分解を高速に計算できる Shor のアルゴリズムを理解するときにも高速フーリエ変換の理解が必要となる．

分割統治法

- マージソート
- 行列の積と Strassen のアルゴリズム
- その他の代表的な分割統治法の例
- 分割統治法の時間複雑度
- 演習問題

分割統治法の時間複雑度

大きさ n の問題を，大きさが高々 $\left\lceil \frac{n}{b} \right\rceil$ の部分問題 a 個に分割し，
それぞれを再帰的に解いた結果を $O(n^d)$ で統合するアルゴリズムを
考える．ここで $a > 0,\ b > 1,\ d \geq 0$ は定数とする．このアルゴリズムの
時間複雑度を $f(n)$ とすると

$$f(n) = a \cdot f\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$$

であり，以下の定理が成り立つ．

定理 (分割統治法の時間複雑度)

$$f(n) = \begin{cases} O(n^d) & (d > \log_b a) \\ O(n^d \log n) & (d = \log_b a) \\ O(n^{\log_b a}) & (d < \log_b a) \end{cases}$$

証明は演習問題

分割統治法

- マージソート
- 行列の積と Strassen のアルゴリズム
- その他の代表的な分割統治法の例
- 分割統治法の時間複雑度
- 演習問題

マージソートの時間複雑度

- $f(n) = \begin{cases} O(1) & (n = 1) \\ f(\lfloor n/2 \rfloor) + f(\lceil n/2 \rceil) + O(n) & (n > 1) \end{cases}$ とする．
  - n が 2 のべき乗のとき，$f(n) = O(n \log n)$ となることを示せ．
  - n が 2 のべき乗とは限らないときも，$f(n) = O(n \log n)$ となることを示せ．

マージソートの時間複雑度の解答例（要素数が 2 のべき乗の場合）

$$f(n) = \begin{cases} O(1) & (n = 1) \\ f(\lfloor n/2 \rfloor) + f(\lceil n/2 \rceil) + O(n) & (n > 1) \end{cases}$$

とする．n が 2 のべき乗のとき，O 記法の定義より，定数 $\alpha, \beta \in \mathbb{R}_{>0}$ が存在し

$$\begin{aligned} f(n) &\leq 2f\left(\frac{n}{2}\right) + \alpha n + \beta \leq 2\left(2f\left(\frac{n}{4}\right) + \alpha\frac{n}{2} + \beta\right) + \alpha n + \beta \\ &\leq 2\left(2\left(2f\left(\frac{n}{8}\right) + \alpha\frac{n}{4} + \beta\right) + \alpha\frac{n}{2} + \beta\right) + \alpha n + \beta \end{aligned}$$

である．ここで $n = 2^k$ となる k，すなわち $k = \log_2 n$ を用いると，

$$f(n) \leq 2^k f(1) + 2^{k-1}\left(\alpha\frac{n}{2^{k-1}} + \beta\right) + 2^{k-2}\left(\alpha\frac{n}{2^{k-2}} + \beta\right) + \cdots + 2^0\left(\alpha\frac{n}{2^0} + \beta\right)$$

である．ここでさらに O 記法の定義より，定数 $\gamma \in \mathbb{R}_{>0}$ が存在し

$$\begin{aligned} f(n) &\leq 2^k\gamma + k\alpha n + \sum_{i=0}^{k-1} 2^i\beta = \gamma n + \alpha n \log_2 n + \beta\frac{1-2^k}{1-2} \\ &= \gamma n + \alpha n \log_2 n + \beta(n-1) = O(n \log n) \end{aligned}$$

である．

マージソートの時間複雑度の解答例（一般の場合）

$$f(n) = \begin{cases} O(1) & (n = 1) \\ f(\lfloor n/2 \rfloor) + f(\lceil n/2 \rceil) + O(n) & (n > 1) \end{cases}$$

とする．n が 2 のべき乗のとき，$f(n) = O(n \log n)$ となることは示せた．n が 2 のべき乗とは限らないときは，n 以上で 2 のべき乗となっている最小の数を m とする．このとき明らかに

$$f(n) = O(f(m)) = O(m \log m)$$

である．m の設定より，$m \leq 2n$ なので，

$$f(n) = O((2n) \log(2n)) = O(n \log n)$$

である．

Strassen のアルゴリズムの時間複雑度

- $f(n) = \begin{cases} O(1) & (n = 1) \\ 7f(n/2) + O(n^2) & (n > 1) \end{cases}$ とする.

  n が 2 のべき乗のとき，$f(n) = O(n^{\log_2 7})$ となることを示せ.

Strassen のアルゴリズムの時間複雑度の解答例

$$f(n) = \begin{cases} O(1) & (n = 1) \\ 7f(n/2) + O(n^2) & (n > 1) \end{cases}$$

とする．n が 2 のべき乗のとき，O 記法の定義より，定数 $\alpha, \beta \in \mathbb{R}_{>0}$ が存在し

$$\begin{aligned} f(n) &\leq 7f\left(\frac{n}{2}\right) + \alpha n^2 + \beta \leq 7\left(7f\left(\frac{n}{4}\right) + \alpha\left(\frac{n}{2}\right)^2 + \beta\right) + \alpha n^2 + \beta \\ &\leq 7\left(7\left(7f\left(\frac{n}{8}\right) + \alpha\left(\frac{n}{4}\right)^2 + \beta\right) + \alpha\left(\frac{n}{2}\right)^2 + \beta\right) + \alpha n^2 + \beta \end{aligned}$$

である．ここで $n = 2^k$ となる k，すなわち $k = \log_2 n$ を用いると，

$$\begin{aligned} f(n) &\leq 7^k f(1) + \alpha\left(7^{k-1}\left(\frac{n}{2^{k-1}}\right)^2 + 7^{k-2}\left(\frac{n}{2^{k-2}}\right)^2 + \cdots + 7^0\left(\frac{n}{2^0}\right)^2\right) \\ &\quad + \beta\left(7^{k-1} + 7^{k-2} + \cdots + 7^0\right) \\ &= 7^k f(1) + \alpha n^2 \sum_{i=0}^{k-1}\left(\frac{7}{4}\right)^i + \beta\sum_{i=0}^{k-1} 7^i = 7^k f(1) + \alpha n^2 \frac{1 - (7/4)^k}{1 - 7/4} + \beta\frac{1 - 7^k}{1 - 7} \\ &= 7^k f(1) + \frac{4}{3}\alpha n^2\left(\left(\frac{7}{4}\right)^k - 1\right) + 6\beta(7^k - 1) = 7^k f(1) + \frac{4}{3}\alpha n^2\left(\frac{7^k}{n^2} - 1\right) + 6\beta(7^k - 1) \end{aligned}$$

である．ここでさらに O 記法の定義より，定数 $\gamma \in \mathbb{R}_{>0}$ が存在し

$$f(n) \leq 7^k\gamma = O(7^{\log_2 n})$$

である．$7 = 2^{\log_2 7}$ であることを利用すると，

$$f(n) = O((2^{\log_2 7})^{\log_2 n}) = O((2^{\log_2 n})^{\log_2 7}) = O(n^{\log_2 7})$$

である．

その他の分割統治法の時間複雑度

問題 定数 $a > 0,\ b > 1,\ d \geq 0$ が $d > \log_b a$ を満たし，

$$f(n) = \begin{cases} \mathrm{O}(1) & (n = 1) \\ a \cdot f(n/b) + \mathrm{O}(n^d) & (n > 1) \end{cases}$$ とする．

このとき，$f(n) = \mathrm{O}(n^d)$ となることを示せ．

解答 これまでとだいたい同じやり方で出来ます．

その他の分割統治法の時間複雑度

入力の大きさが n の再帰アルゴリズムの基本演算の回数を $T(n)$ で表すことにする．ただし，$T(2) = T(1) = 1$ とする．以下のそれぞれに関して，空欄を埋めよ．

1. $T(n) = 7T\left(\left\lceil \frac{n}{7} \right\rceil\right) + n$ のとき，$T(n) = \Theta($ $\boxed{}$ $)$ である．
2. $T(n) = 8T\left(\left\lceil \frac{n}{2} \right\rceil\right) + n^3$ のとき，$T(n) = \Theta($ $\boxed{}$ $)$ である．
3. $T(n) = T(n-1) + 2$ のとき，$T(n) = \Theta($ $\boxed{}$ $)$ である．

（2015, 2016 年度期末試験問題より）

さらなる勉強のために

- マージソート，Strassen のアルゴリズム，分割統治法の時間複雑度は [Dasgupta et al., 2006] から引用した．しかし，いずれも有名なものであるので，他のテキストでも同様である．
- 選択ソートは [Sedgewick and Wayne, 2011] からの引用である．非常に当たり前でありしかも効率的ではないアルゴリズムであるので，意外とテキストに書かれていない．

参考文献

[Dasgupta et al., 2006] Dasgupta, S., Papadimitriou, C., and Vazirani, U. (2006).
Algorithms.
McGraw-Hill Science/Engineering/Math.

[Sedgewick and Wayne, 2011] Sedgewick, R. and Wayne, K. (2011).
Algorithms.
Addison-Wesley.