Vol.2015-CE-132 No.38
Vol.2015-CLE-17 No.38
2015/12/6

アルゴリズムデザインと再帰

浅野考平[†1]

本論文では，アルゴリズム教育における“再帰”の概念の教育方法を提案する．古典的なアルゴリズムにおいて，“再帰”は大規模なデータを処理するためには欠かすことのできない手法である．しかしながら，再帰は，学習者にとって理解することが最も難しい概念の１つであるといわれている．ここでは，アルゴリズムの設計過程のモデルをつくり，そのモデルに基づいて，古典的なアルゴリズムを分析することによって，再帰を含むアルゴリズムの新しい教育方法を考案する．

Algorithm design and recursion

KOUHEI ASANO[†1]

In this study, we propose a method for teaching the concept of “recursion” in algorithm education. In classical algorithms, “recursion” is an essential method that process large-scale data. However, recursion is considered as one of the most difficult concepts for learners to understand. Here, we create a model of the algorithm design process, and by providing an analysis of classical algorithms, develop a new method for teaching algorithms that include recursion based on the proposed model.

1. はじめに

本論文では，アルゴリズム教育における“再帰”の概念の教育方法を提案する．現実に作成されるプログラムは多様であり，プログラミングにおいてインターフェースの作成の占める割合が大きくなっているなど大きく変化しているが，大規模なデータを処理する古典的なアルゴリズムを学ぶことも依然として必要である．“再帰”は，このような古典的なアルゴリズムにおいて欠かすことのできない手法であるにもかかわらず，アルゴリズムを学習する者にとって理解することが最も難しい概念の１つである．

筆者は、アルゴリズムの教育においては，その設計過程を含めて教えることが有効であると考えて教育方法について提案を行った [1][2]．なぜならば，このことによってアルゴリズムを効果的に記憶し，使うことが可能になるとともに，アルゴリズムを自ら設計する能力を向上させることも期待できると考えているからである．

当然のことながら，それぞれのアルゴリズムが設計された“本当の”過程はおそらく誰にもわからないだろう．しかし，教える際に，それを再現する必要はなく，学習者が「起こりえただろう」と考えられる過程を示せばよい．別な言い方をすれば，学習者が自ら設計するときに類推するための基礎となるような過程であればよい.そこで,まず，設計過程のモデルをつくり，それに沿って過程を説明するという方法を用いる．

第２節において，まずアルゴリズムの設計過程のモデルを作成する．そして，第３節において，モデルに基づいて既知のアルゴリズムの設計過程の説明を試みた.その結果，通常は，再帰アルゴリズムとして表現されていないアルゴリズムを含めて，情報教育の基礎的な段階で教えられる多くのアルゴリズムは，対象とする問題をサイズの小さい入力に対する同型問題を解くことによって，解決するという問題解決法によって，問題のアルゴリズムを設計したと考えることができた．そして，第５節において，この結果に基づいて教育方法を提案する．

2. アルゴリズムの設計過程のモデル

最初に，アルゴリズム設計過程のモデルをつくる．設計過程を教えるためには，過程をモデル化し，そのモデルに沿って過程を学習者に説明することが効率的である．例えば，古くから発見学習では，モデルに沿って学習者にヒントをあたえるという方法が用いられており[3][4]，この方法は一般的である．

以下のモデル作成の基本になる考え方は，「アルゴリズムは問題の解決法であり，アルゴリズムの設計は問題解決法の発見である」ということである．問題解決方法を発見する方法は，問題解決法と呼ばれるが，この意味での問題解決法は一般的には存在しない.しかし,「解くべき問題を，解決がより易しい小問題にブレークダウンする」という素朴な方法がある．アルゴリズムの設計においても，この素朴な方法によって，設計する過程を説明することができる場合がある．ここで「小問題」は，さらにブレークダウンする必要がある場合があるので，以下では「下位問題」ということにする．また，下位問題に対して，元の問題を上位問題ということにする．

モデルそのものは以下のように単純である．

1 関西学院大学理工学部
School of Science and Technology, Kwansei Gakuin University

解くべき問題を，解決がより易しい下位問題にブレークダウンする．コンピュータの基本的な操作である「加減乗除」「比較」「条件による分岐」によって直ちに解決可能な問題までブレークダウンする．ここで,「下位問題にブレークダウンする」という意味は，下位問題の解から，上位問題の解が構成できるような問題に帰着させる，ということである．また，アルゴリズムは,“最下位の”問題までのブレークダウンする手順と,“最下位の”問題の解から“最上位の”問題の解を合成する手順を併せた手順である．

このモデルによる設計過程を例で具体的に説明する．

例１：重複要素の出力問題
整数の配列 a が与えられたとき重複する要素を出力する．

単純に，すべての対(i,j)に対して，a[i]と a[j]を比較して重複するものがあれば出力すればよい．しかし，この場合，

（１）配列をソートする．
（２）ソート済みの配列が与えられたとき，重複する整数を出力する．

という２つの問題の組み合わせが下位問題となる．そして，この下位問題から，アルゴリズムを構成することができる．ソートが既知であるとすれば，この問題を解くアルゴリズムは以下のように単純に記述することができる．また，効率の良いソートを用いれば，先に述べた素朴なアルゴリズムより効率がよい．

（Ｉ）a をソートする．
（Ⅱ）a[0]から順に探索して，同じ整数が連続しているとき出力する．

「易しい」という評価は，主観的あるいは感覚的である．この場合，下位問題（１）のソート問題は，解決方法を知っている者にとって，知っているから易しいということであって，ソートのアルゴリズムを知らない者にとっては易しいとはいえない．
情報科学の基礎的な教育において教えられるアルゴリズムの多くの設計過程は，この素朴なモデルに沿って説明できる．

例２：最大公約数問題に対するユークリッド互除法
上位問題：a, b（ a > b ）の最大公約数を求める．
下位問題：（１）a, b（ a > b ）に対して，
a mod b = 0 のとき，a, b の最大公約数を求める．
（２）a, b（ a > b ）に対して，b≠0 のとき，
b と a mod b の最大公約数を求める．

問題（１）の解は，b であるから，解決方法は自明である．下位問題（２）は，上位問題よりも入力サイズが小さい問題であり，このブレークダウンを繰り返すことによって，解決方法が自明な問題（１）に帰着することができる．アルゴリズムは,「 a mod b = 0 になるまで，a mod b を繰り返し，a mod b = 0 になれば，b を返す」，ということになる．

重要なのは，下位問題をどのようにして見いだすか，ということである．重複問題の例でも説明したように,「易しい」という評価は主観的あるいは感覚的であり，明確に定義することはできない．しかし，次のような問題が候補となる．

（ア）問題を解決する方法を知っている．
（イ）同じ問題に対して，サイズの小さい入力を与える．
（ウ）出力の条件を緩和する．
（エ）入力の条件を追加する．

重複要素の出力問題で言えば，ソート問題（１）が（ア）のタイプの問題であり，ソート済み配列に対する重複要素の出力問題が（エ）のタイプの問題になる．ここで候補としてあげた（ア）～（エ）の問題を形式的につくっても，下位問題を構成できるとは限らない．それらと必要に応じて他の問題を組み合わせて上位問題と同等の問題となる問題群を構成できたとき，はじめて下位問題となる．また，単純にはタイプ分けできない．例えば上記の重複問題の下位問題（２）は（エ）であって，かつ（ア）であるといえる．最大公約数問題であれば，（１）は，（ア）タイプの問題で，（２）が（イ）タイプの問題になる．
タイプ（イ）の問題はサイズの小さい入力に対する同型問題であり，常にもとの問題より易しいことは明らかである.(ウ)は,多くの場合元の問題より易しい問題であろう．しかし，例えば，ある条件を満足するすべてを出力するようなタイプの問題に対しては易しいとは限らない.(エ)についても適切な条件でなければ追加しても易しいとは限らないが，例えば重複問題の例で上げたように，入力が順序集合であれば，ソート済みであるという条件を付け加えると多くの場合易しくなる．
いずれのタイプでも形式的に問題をつくるだけでは，下位問題とはならないのであって，具体的に問題ごとに下位問題となるか，個別に検討する必要がある．それぞれの問題に“分割”するという問題や，問題の解を“合成”するという問題が必要になる．また，モデルでも述べたようにコンピュータの基本的な操作で直ちに解決できる問題にまで，ブレークダウンを繰り返すことによってアルゴリズム

が構成できる．

特に，サイズの小さい入力に対する同型問題（タイプ（イ）の問題）を含む下位問題が構成させた場合には，ブレークダウンを繰り返すことによって自明な下位問題に帰着させることができる．従って，もし，ブレークダウンの過程の手順を構成することができれば，アルゴリズムをつくることができるはずである．

3. 設計モデルに基づくアルゴリズムの設計過程の説明

この節では，実際にアルゴリズムの教育の中で学習の対象となっているアルゴリズムの多くは，前節でつくったモデルに基づいて分析すると，サイズの小さい入力に対する同型問題を含む下位問題によって設計したと説明できることを示す．

このモデルで設計過程を説明できるアルゴリズムを，設計過程に忠実に表現すると,“再帰”を含む形で表現される．しかし，現実には，必ずしも再帰アルゴリズムとして表現されている訳ではない．ここでは，通常は再帰を含まないアルゴリズムで表現されるような問題を例として取り上げている．以下，選択ソート，バブルソート，二分探索，ユークリッド互除法，挿入ソートを例にあげて，われわれのモデルに沿った設計過程を説明する．

例３：選択ソート（バブルソート）とソート問題の下位問題

上位問題：
長さ n の配列 a[0], …, a[n-1]をソートする．
下位問題：
（１）長さ n の配列 a[0], …, a[n-1]に対して，a[0]を最小値とするように入れ換える．
（2）長さ n-1 の配列 a[1], …, a[n-1]をソートする．

問題（２）は，サイズが１だけ小さい問題であり，（１）はソート問題を，各 i に対して，a[i]を順位が i+1 の要素となるよう並べ換えるという問題であると考えると，出力条件を緩和するという（ウ）のタイプの問題である．

このような問題のブレークダウンを用いて，アルゴリズムを設計すると，ブレークダウンの手順としては，最小値を左端に移動するアルゴリズムになる．また，（２）の問題の入力サイズが減少し，１となるとソートの必要がなくなるので，解の合成の過程は不要になる．すなわち，（１）の問題に対するアルゴリズムの選択によって，バブルソートまたは選択ソートができる．

例４：二分探索と探索問題の下位問題

上位問題：昇順にソートされた長さ n の重複要素のない配列 a とデータ x が与えられたときに，x が存在するか，しないかを判定し，x が存在するときは，x の位置を返す．
下位問題：
（１）昇順にソートされた長さ n/2 の配列と x が与えられたとき，x が存在するか，しないかを判定し，x が存在するときは，x の位置を返す．
（２）昇順にソートされた長さ n の配列と x が与えられたとき，x が a[0]～a[n/2]に存在するか，a[n/2+1]～a[n-1]存在するか，判定する．

（１）は，サイズの小さい入力に対する同型問題である．
（２）は，例示した（ウ）タイプ，すなわち，出力条件の緩和によって得られた問題と考えられる．あるいは，（１）の問題に帰着させるために，対となる問題として案出したとも考えられる．

例５：挿入ソートとソート問題の下位問題

上位問題：
長さ n の配列 a[0], …, a[n-1]をソートする．
下位問題：
（１）昇順に並んだ長さ n の配列 a[0], …, a[n-2]に対して，a[0], …, a[n-1]が昇順になるように a[n-1]を挿入する．
（２）長さ n-1 の配列 a[0], …, a[n-2]をソートする．
ブレークダウンの過程は配列の要素は移動せず，長さを１つずつ減らすだけであるから，何らの操作を行う必要がないが，解の合成の過程で（１）を解かなければならない．

この節で例として取り上げたアルゴリズムは，いずれも，サイズの小さい入力に対する同型問題を含む下位問題にブレークダウンすることによって設計過程を説明することができ，かつ，通常は再帰を含まない形で記述されているものを選んでいる．それは下位問題として,「サイズの小さい入力に対する同型問題」を選んでも，必ずしもアルゴリズムが再帰を含んだ形で記述されていないことを強調するためである．

もちろん，これらの例が，基礎教育の場で教えられているアルゴリズムのすべてではない．しかし，大規模なデータを処理するアルゴリズムであれば，データの個数が小さい場合に帰着させることが有効であることは容易に推測できることである．

4. 再帰呼び出しの必要性

第３節では，再帰を含まないアルゴリズムのみを例として取り上げた．しかし，言うまでもなく，再帰はアルゴリズムを記述する上で必要である．この節では，再帰の必要性をどのようにして説明するか，について述べる．

前節の例であげた選択ソート，バブルソート，ユークリッドの互除法，二分探索は，問題のブレークダウンに忠実にアルゴリズムを記述したときに末尾に再帰を含むアルゴリズムになり，再帰は，末尾のみであるから，末尾再帰を除去することによって，再帰を含まないアルゴリズムにできる．

しかし，選択ソート，バブルソートは最小値を左端に移動することによって，（2）のサイズが１小さい場合のソート問題を帰着させるのであり，単純にブレークダウンを繰り返せば，サイズが小さくなって，最終的には解が自明である同型問題に帰着する，という考えに基づくアルゴリズムであると考えればよい．このように考えれば，逐次的なアルゴリズムになる．再帰を含んだアルゴリズムを作成した後，再帰を除去したアルゴリズムであると考える必要はない．ユークリッドの互除法，二分探索についても同様である．

挿入ソートも再帰アルゴリズムで記述できるが，サイズの小さい入力に対する同型問題への分割が先頭で行われることから，分割（ブレークダウン）のための操作は必要ない．また，分割と言っても，入力の状態が変更されるのではない．従って，アルゴリズムとしては長さ２の配列を入力とする問題の解から,逐次的に解を構成するだけでよい．通常，挿入ソートとして記述されているのは再帰を含んでいない．単に，解が自明であるようなサイズの小さい同型問題から元の問題の解を逐次的に構成する，すなわち，ソート済みの配列に１つずつ順次要素を挿入することによってソートする過程をアルゴリズムとして記述したものであると考えられる．

再帰アルゴリズムにおいて，再帰呼び出しが先頭に１個だけある場合には，一般的に再帰を除去できる．しかし，再帰を含まない形で記述された挿入ソートは，再帰アルゴリズムから再帰を除去したものではないと考えることができる．

第２節のアルゴリズムの設計過程のモデルの中で，アルゴリズムは，

「最下位の問題へブレークダウンするまでの手順と最下位の問題の解から最上位の問題の解を合成する手順を併せた手順である.」

と述べた．ただし，形式的にブレークダウンと合成という順序で併せてアルゴリズムができるとは限らない．手順を考察し，ブレークダウンと解の合成の過程を適切な順序で組み合わせる必要がある．第３節のアルゴリズムの例においても，問題のブレークダウンの過程，解の合成過程を考察することによって順序を定めている．

サイズの小さい入力に対する同型問題を２個以上含むような下位問題の解を合成する必要のある場合，その多くではアルゴリズム設計における上記の過程は複雑になる．そのとき，関数呼び出しの機能を使って，アルゴリズムの手順を定めることが必要になる．しかし，例えば，フィボナッチ数列を求めるという問題では，再帰を用いないアルゴリズムが簡単に構成できる．

例６：フィボナッチ数列

問題:$f(n) = f(n\text{-}2) + f(n\text{-}3), f(0) = a, f(1) = b$ で定まる数列の第 n 項を求める．

この場合，下位問題は明らかであり，アルゴリズムはサイズの小さい同型問題２個の解の和を求める，というものである．よく知られているように，$i = 2$ から n まで順に $f(i) = f(i\text{-}2) + f(i\text{-}1)$ を計算することによりフィボナッチ数列の項が求められる．また，この方が再帰呼び出しを用いるよりも速い．同様に，下位問題によっては再帰を用いない動的計画法によるアルゴリズムとして記述できる場合もある．

一方，再帰呼び出しが１個である場合でも，計算量のオーダーが対数である効率的なベキ乗計算法のように，再帰を除去しようとするとスタックを用いるので，アルゴリズムの記述が複雑になる場合もある．この場合，問題のブレークダウンの過程が自明ではなく，かつ解の合成過程が必要であるからである．

まとめれば，問題のブレーダウンの過程と解の合成過程の組み合わせが逐次的な手順によって実現できないときには，プログラム言語が持つ関数の呼び出しの機能である再帰呼び出しを用いて記述する必要があるということである．

5. 教育方法の提案

これまでの考察により，サイズの小さい入力に対する同型問題を下位問題に含むブレークダウンという設計方法は,実際に再帰アルゴリズムの形で記述されているアルゴリズムに特有のものではなく，多くのアルゴリズムの設計過程に共通するものであることを示した．このことは，教育の過程として，問題をサイズの小さい入力に対する同型問題に帰着させるという考え方の理解と再帰を含むアルゴリズムの理解は分離できることを意味している．

学習者の前提であるが，少なくとも関数呼び出しについての理解が必要である．具体的には，アルゴリズムを実行中に関数を呼び出したときには，呼び出した側の関数の実行を一時停止し，呼び出された側の関数の実行を開始し，実行が終了したときに，必要ならば値を返して終了し呼び出した側の関数の実行を再開するということの理解である．

再帰の教育方法の提案の概略を述べる．アルゴリズムの設計モデルの説明を最初に行う．場合によっては，サイズの小さい入力に対する同型問題のよるブレークダウンだけでも良いと考えられる．

ユークリッド互除法，二分探索，選択ソート，バブルソ

ートなどの初等的なアルゴリズムを教える際に，この考え方に沿って設計過程を説明できるということも教える．また，フィボナッチ数列の計算などを通して，問題のブレークダウンとアルゴリズムの設計との関係を説明する．

その後，ベキ乗計算問題に対して，ベキを，0 になるまで 2 で割るという，下位問題へのブレークダウンに基づいてアルゴリズムを設定することを試み，逐次的なアルゴリズムとして実現することが困難であることを理解させる．

同様にしてクイックソートに至る下位問題を説明して，逐次的なアルゴリズムとして記述することが難しいことを理解させる．すなわち，これらの例を通して再帰の必要性を経験的に理解させ，その後で，再帰によるアルゴリズムの記述を行う．

6. 今後の課題

本稿において，再帰の教育方法について理論的に分析した．現実にアルゴリズム教育の一部として実践するためには，さらに詳細化することが必須である．例えば，教育に要する時間的な分析もなされていない．

参考文献

1) 浅野考平, 森戸隆文:“アルゴリズム教育再考”, 情報処理学会 コンピュータと教育研究会報告， vol.2014-CE-125 No.1，2014
2) 浅野考平，森戸隆文 :“アルゴリズム教育における発見学習の試み”，情報処理学会 コンピュータと教育研究会報告，vol.2014-CE-129 No.1，2015
3) ガニェ，R.M.，ウェイジャー，W.W.，ゴラス，K.S.，ケラー，J.M.，: インストラクショナルデザインの原理，鈴木克明，岩崎信訳，北大路書房，2007， p. 82
4) 森田英嗣 : 日本教育工学会（編）教育工学事典，実教出版，2000，pp. 432−433