

# 基礎から アルゴ

- Part1** → 基本アルゴリズムを徹底図解  
イメージをつかんでコードに生かす p.16
- Part2** → Java APIのソースから読み解く  
アルゴリズム実装テクニック p.26
- Part3** → 劇的に処理時間を短縮  
実例から学ぶアルゴリズム活用法 p.38

Photo: Gary Cornhouse/Getty Images

図解と  
コードで  
徹底理解!

# 学ぶ アルゴリズム

アルゴリズムと聞いて、「ちょっと苦手だなあ」と思う方は少なくないと思います。最近のプログラミングでは、とりあえずメソッドや関数を呼び出せばアルゴリズムを利用できるようになっているので、アルゴリズムそのものを学ぶ必要はないと考えている方もいるかもしれません。

しかし、プログラムを作っているときに、頭に浮かんだ処理方法をコードとして表現するのに苦労することはありませんか？ アルゴリズムとはなんらかの問題を解決する手順であり、それをコンピュータが理解できるように記述したものがプログラムです。つまり、プログラムを書いている以上、なんらかのアルゴリズムを記述しているといえます。アルゴリズムを学ぶことは、プログラミングの腕を上げるために欠かせないのです。

この特集では、アルゴリズムの基本をじっくりと解説します。Part1では、図をふんだんに使って、探索とソートにかかわる七つのアルゴリズムを説明します。ここでは、基本的な考え方を紹介してから、コードとして表現するための考え方を詳しく解き明かすという2段階の構成になっています。図でイメージをつかんでから、コードとして表現するための考え方を身に付けましょう。

Part2では、オープンソースのJava開発キット「OpenJDK」のソースコードをのぞいてみます。Java APIのソート・メソッドのソースコードを読みながら、アルゴリズムの実装例を具体的に解説します。Part1で基礎をつかんでから読むと、より深く理解できるでしょう。

Part3では、アルゴリズムの工夫が処理速度の劇的な短縮につながった、開発現場での実例をお見せします。問題を解決する方法は一つとは限らないということと、様々な方法を試行錯誤することの大切さを実感してください。「頭で考えた処理を、なかなかコードとして表現できない」という方には特に参考になるでしょう。

# ヒープソート

先に紹介した二分探索木のように、木構造を利用するソートアルゴリズムが「ヒープソート」です。ここでいう「ヒープ」は、木構造の一つを意味します。ヒープの木構造は、二分探索木とは異なり、親ノードと子ノードの関係は親が最も大きくなればよく、子同士の大小関係に決まりはありません。

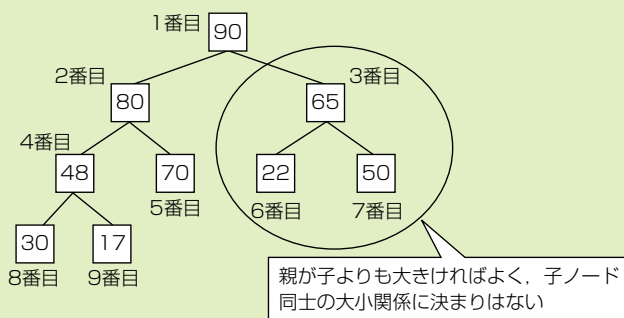
ヒープソートはあくまで仮想的なもので、木構造を表現するために専用のデータ構造を用意する必要はなく、配列で表現できます。手順としては、まず未ソートの配列（ももとの配列）

からヒープを構築します。ヒープが完成したらその頂点から値を取り出します。すると、ノード間の大小関係が崩れるので、ヒープを再構築し、再び頂点から値を取り出します。この作業を繰り返すことで、配列の要素をソートしていきます。

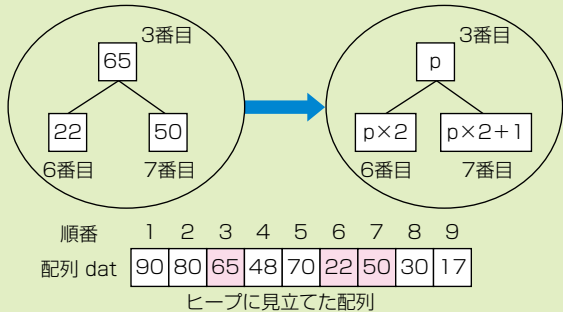
一般にヒープソートの平均的な処理速度は、このあとで紹介するクイックソートに比べると多少劣りますが、最遅のケースと最速のケースを比べても大きな差が発生せず、比較的安定しているの特徴です。

## 基本的な考え方

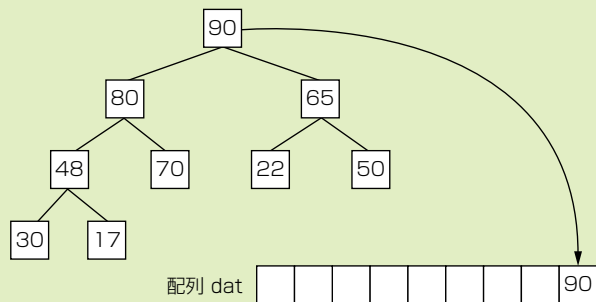
①未ソートの配列をヒープ（木構造の一種）に見立てて、ヒープの上方に大きな値が行くように並べ替える（ヒープの構築）



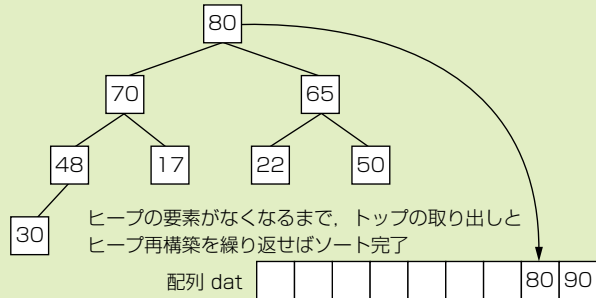
②ヒープ上の親子の位置は簡単な計算で配列上の位置に換算できる



③ヒープが完成したら、頂点から最大値を取り出して配列の末端に置く



④頂点を取り出してヒープが壊れたら残りの要素で再構築し、再び頂点から最大値を取り出して、配列の末端より一つ手前に置く



## プログラムとして実装するための考え方

①未ソートの配列をヒープに見立てる

