本書の構成と使い方

構成

本書の各章は、アルゴリズムのはじめの一歩を無理なく確実に完全攻略できるように、ウォーミングアップ➡基本的なアルゴリズムとデータ構造➡やや高度なアルゴリズムとデータ構造➡高度なアルゴリズムと特殊なアルゴリズム、という順に構成されています。

対象とする読者

本書の内容は、初級エンジニア、大学・専門学校の学生、基本情報技術者試験の受験者を主な対象としています。読者のイメージは、何らかの入門書でプログラミングの学習経験があり、プログラミングの基礎をおおまかにわかっている人です。本書のゴールは、お手本を見ずに、挿入法や二分探索法のプログラムを作れるようになることです。それによって、国家試験である基本情報技術者試験レベルの問題が解けるようになるはずです。

使い方

本書は、第1章から第10章まで、徐々にレベルを上げる内容になっているので、はじめてアルゴリズムを学習する人は、第1章から順にお読みください。本書の各章は独立した内容になっているので、すでにある程度の経験がある人は、興味のある章からお読みください。基本的に、どの章も、•アルゴリズムの説明、•擬似言語とJavaのコード、•Javaの実行結果、•手作業によるアルゴリズムのトレース、•Javaによるアルゴリズムのトレース、という構成になっています。

JavaとC言語のサンプルコードのダウンロード

本書に掲載されているJavaのコード、および同じ機能のプログラムをC言語で記述したコードは、技術評論社のWebサイト（https://gihyo.jp/book/2019/978-4-297-10394-1）からダウンロードできます。

各章の概要

本書の流れを次のページに示します。各章の終わりには、知識の確認問題とコラムがあります。また、本文の中には、知識の幅を広げるためのクイズを用意しました。

第1章 ウォーミングアップ
Keyword 考えるコツ、4つの処理、3つの流れ、ユークリッドの互除法

↓

第2章 ループと配列の基本と線形探索
Keyword ループ、配列、合計値、線形探索、トレース

↓

第3章 二分探索と計算量
Keyword 昇順、降順、二分探索、計算量、ビッグ・オー表記、O ($\log_2 N$)

↓

第4章 多重ループと挿入法
Keyword 多重ループ、掛け算の九九表、挿入法、ループの条件

↓

第5章 連結リストの仕組みと操作
Keyword データ構造、連結リスト、構造体、ポインタ、要素の挿入と削除

↓

第6章 二分探索木への追加と探索
Keyword 二分探索木、深さ優先探索、再帰呼び出し、要素の追加と探索

↓

第7章 ハッシュ表探索法
Keyword ハッシュ表、ハッシュ関数、ハッシュ値、シノニム、O(1)

↓

第8章 再帰呼び出しとクイックソート
Keyword 再帰呼び出し、階乗、クイックソート、基準値

↓

第9章 動的計画法とナップサック問題
Keyword 動的計画法、再帰呼び出し、フィボナッチ数、ナップサック問題

↓

第10章 遺伝的アルゴリズムとナップサック問題
Keyword 遺伝的アルゴリズム、適応度、交叉、突然変異、ナップサック問題

↓

付 録 基本情報技術者試験の問題で腕試ししてみよう
Keyword ヒープの性質を利用したデータの整列、文字列の誤りの検出

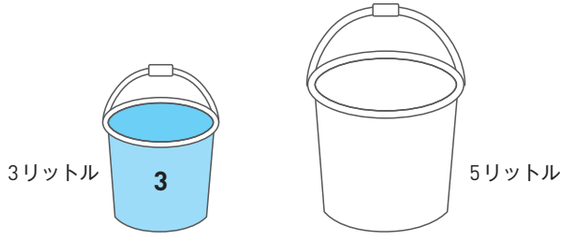
それでは、始めましょう！

ここが Point
「何かを行ったら、そこで一息つく」という感覚を持てば、処理を区切れるようになる

ケツの水を捨てる」に区切れます。「何か処理を行ったら、そこで一息つく」という感覚を持てば、処理を区切れるようになります。

焦らないことも重要です。すぐに答えが得られなくても、焦らずに、何か処理を行ってみましょう。両方のバケツに同時に水を汲んだら先に進めないので、とりあえず、3リットルのバケツに水を汲むことからスタートしてみましょう。

手順 1 3リットルのバケツに水を汲む



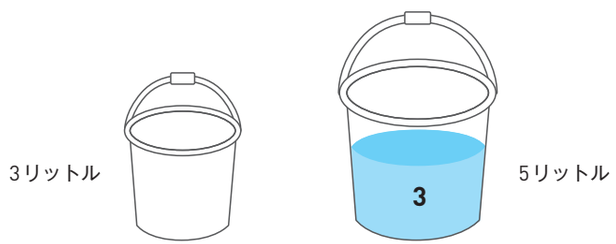
ここが Point
あせらずに、落ち着いて、ゆっくり考える

ここで焦ってはいけません。アルゴリズムを考えるのが苦手な人は、焦りすぎなのです。**落ち着いて、ゆっくり考えましょう。**次の手順として、5リットルのバケツに水を汲むと、両方のバケツが一杯になり、先に進めません。3リットルのバケツの水を捨てたら、両方のバケツが空になり、最初の状態に戻ってしまいます。

そうなると、3リットルのバケツの水を、5リットルのバケツに移すしかありません。とりあえずやってみましょう。この**「とりあえずやってみる」**ということが重要です。それによって**「わかった！」とひらめく**ことがあるからです。

ここが Point
「とりあえずやってみる」ことで、「わかった！」とひらめくことがある

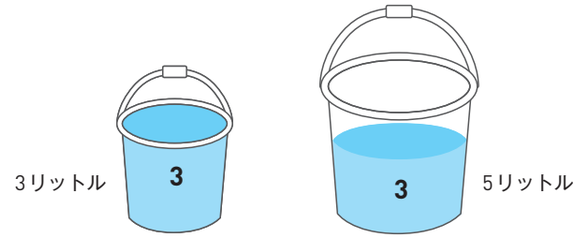
手順 2 3リットルのバケツの水を5リットルのバケツに移す



次の手順として、5リットルのバケツの水を捨てたら、両方のバケツが空になり、最初の状態に戻ってしまいます。5リットルのバケツに水を汲んだら、水を

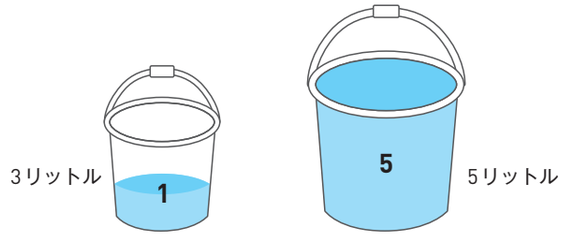
移した意味がありません。そうなると、3リットルのバケツに水を汲むしかありません。とりあえずやってみましょう。

手順 3 3リットルのバケツに水を汲む



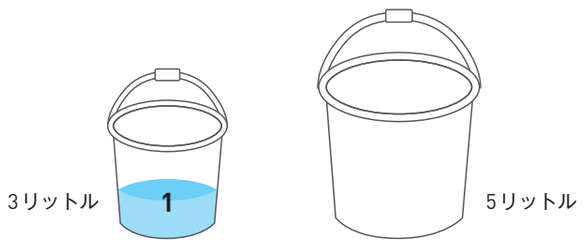
この時点で「わかった！」とひらめいたのではないでしょうか。次の手順として、3リットルのバケツの水を5リットルのバケツに移すと、残りは1リットルになります。1リットルと3リットルを足すと4リットルになります。はやる気持ちを抑えて、3リットルのバケツの水を5リットルのバケツに移しましょう。

手順 4 3リットルのバケツの水を5リットルのバケツに移す



5リットルのバケツの水は必要ないので、捨てましょう。

手順 5 5リットルのバケツの水を捨てる



Quiz　昇順を降順に変えるには？

ここで示したJavaのプログラムでは、昇順にソートを行いました。これを、降順にソートするプログラムに改造してください。

ヒント たった1文字を変えるだけで改造できます。

解答は 282ページ にあります。

Quiz　breakを使わずにループを途中終了するには？

ここで示したJavaのプログラムでは、breakという表記でループを途中終了しています。これを、breakを使わずにループを途中終了するプログラムに改造してください。

ヒント 内側のループを改造します。繰り返しの条件がポイントです。

解答は 282ページ にあります。

4-2-2　アルゴリズムのトレース

挿入法で、要素数5個の配列aを昇順にソートするアルゴリズムを、手作業でトレースしてみましょう。以下に示した手順を、1つずつ確認してください。

手順 1 insに初期値として1を代入する。ins ＜ 5が真なので外側のループ処理を行う

a[0]	a[1]	a[2]	a[3]	a[4]
90	34	78	12	56

ins	cmp	temp
1	?	?

手順 2 挿入するa[ins]の値をtempに逃がす（代入する）

a[0]	a[1]	a[2]	a[3]	a[4]
90	34	78	12	56

ins	cmp	temp
1	?	34

手順 3 cmpに初期値としてins － 1を代入する。cmp ≧ 0が真なので内側のループ処理を行う

a[0]	a[1]	a[2]	a[3]	a[4]
90	34	78	12	56

ins	cmp	temp
1	0	34

手順 4 a[cmp] ＞ tempが真なので、a[cmp＋1]にa[cmp]を代入して、a[cmp]を1つ後ろに移動する

a[0]	a[1]	a[2]	a[3]	a[4]
90	90	78	12	56

ins	cmp	temp
1	0	34

手順 5 cmpの値を1減らす。cmp ≧ 0が偽なので内側のループ処理を終了する

a[0]	a[1]	a[2]	a[3]	a[4]
90	90	78	12	56

ins	cmp	temp
1	−1	34

擬似言語

```
/* StationList構造体の定義 */
○構造型：StationList { 文字列型：name, 整数型：next }

/* 連結リストの実体となる配列（要素数を最大10個とする）*/
○StationList：list[10]

/* 先頭ポインタ */
整数型：top

/* 初期状態の連結リストを作成する関数（ここから）*/
○initStationList()
/* 駅名とポインタを設定する */
・list[0].name ← "新大阪"
・list[0].next ← -1
・list[1].name ← "名古屋"
・list[1].next ← 3
・list[2].name ← "東京"
・list[2].next ← 4
・list[3].name ← "京都"
・list[3].next ← 0
・list[4].name ← "新横浜"
・list[4].next ← 1

/* 先頭ポインタを設定する */
・top ← 2
/* 初期状態の連結リストを作成する関数（ここまで）*/

/* 連結リストの要素を表示する関数（ここから）*/
○printStationList()
○整数型：idx
・idx ← top
■ idx ≠ -1
|・list[idx].nameと"→"を表示する
|・idx ← list[idx].next
■
・改行する
/* 連結リストの要素を表示する関数（ここまで）*/

/* プログラムの実行開始位置となるmain関数（ここから）*/
○main
/* 初期状態の連結リストを作成して表示する */
・initStationList()
・printStationList()
/* プログラムの実行開始位置となるmain関数（ここまで）*/
```


initStationList関数では、連結リストのそれぞれの要素に駅名とポインタを設定し、先頭ポインタtopに先頭の要素の添え字である2を設定します。list[0].nameやlist[0].nextは、list[0]という要素のnameやnextという意味です。この「.（ドット）」を「～の」と読むとわかりやすいでしょう。list[0].nameなら「list[0]のname」です。構造体にまとめられた個々のデータ（構造体のメンバと呼びます）は、このようにドットを使った表現で読み書きします。

printStationList関数には、リストの操作において大いに注目すべきポイントがあります。通常の配列の場合は、ループカウンタを配列の要素番号に合わせて、0～末尾まで順番に1ずつ増やしながら要素を読み出します。それに対して、**連結リストの場合は、次の要素を読み出す変数idx（index＝「添え字」という意味です）を用意し、idxに先頭topの値を格納することからスタートして、idxの値を現在の要素のポインタで上書き更新することで、次の要素を読み出します。**これは、「・idx ← list[idx].next」の部分です。末尾の要素を読み出した後は、「・idx ← list[idx].next」によってidxに－1が格納されるので、**リストをたどる繰り返し処理の条件は、「idxが－1でない限り」**です。これは、「■ idx ≠ －1」の部分です。

ここがPoint

連結リストでは、現在の要素のポインタをたどることで、次の要素を読み出す

プログラムの実行開始位置となるmain関数では、initStationList関数を呼び出してから、printStationList関数を呼び出しています。これによって、初期状態の連結リストの要素が画面に表示されます。

Javaでプログラムを作って、実際の動作を確認してみましょう。以下は、先ほど擬似言語で示したプログラムをJavaで記述したものです。Javaでは、クラスの中にメソッドを記述しなければならない約束になっているので、StationListクラスとは別にLinkedListクラスがあり、LinkedListクラスの中に、連結リストの実体となる配列list、先頭ポインタtop、initStationListメソッド、printStationListメソッド、およびmainメソッドを記述しています。このプログラムをLinkedList.javaというファイル名で作成してください。

Java
LinkedList.java

```
// StationListクラスの定義
class StationList {
  public String name; // 駅名
  public int next;    // ポインタ
}
```

ここが Point

最終的な解のナップサックに入っている品物が何であるかを調べるときに、それぞれのナップサックに最後に入れた品物の情報を使う

最終的な解として、6kgのナップサックの価値の最大値は800円になりました。**このナップサックに入っている品物が何であるかを調べるときに、それぞれのナップサックに最後に入れた品物の情報を使います。**この仕組みも、プログラムの実行結果で確認できるようにしてあります。6kgのナップサックに最後に入れたのは、品物Dです。品物Dは4kgなので、残りは6kg－4kg＝2kgです。2kgのナップサックに最後に入れたのは、品物Bです。品物Bは2kgで、残りは2kg－2kg＝0kgなので、これ以上品物は入っていません。

6kgのナップサックに入っている品物を調べて、解を表示する

```
＜ナップサックに入っている品物を調べる＞
6kgのナップサックに最後に入れた品物はDです。
  D, 4kg, 500円
  6kg - 4kg = 2kgです。
2kgのナップサックに最後に入れた品物はBです。
  B, 2kg, 300円
  2kg - 2kg = 0kgです。

＜解を表示する＞
重量の合計値 = 6kg
価値の最大値 = 800円
```

9-2-3 動的計画法でナップサック問題を解くプログラム

以下に、ナップサック問題を解くプログラムを示します。かなり長いプログラムなので、擬似言語を省略してJavaだけで示します。KnapsackDP.javaというファイル名で作成してください。プログラムは、いくつかのフィールド、showKnapメソッド、mainメソッドから構成されています。**このプログラムは、ナップサック問題を解く仕組みを説明するためのものなので、プログラムの概要を理解できれば十分**です。第10章でも、かなり長いJavaのプログラムが登場します。そのときにプログラムの内容を詳しく理解することにチャレンジしてください。

小さく分割した部分問題の解は、2次元配列maxValue[ ][ ]に記憶します。それぞれのナップサックに最後に入れた品物は、lastItem[ ]に記憶します。showKnapメソッドは、品物を吟味した直後のmaxValue[ ][ ]の内容を表示します。

mainメソッドの中にある多重ループで、品物を1つずつ増やしながら、0kg～6kgの耐重量のナップサックの価値の最大値を求め、その結果をmaxValue[ ][ ]に記憶していきます。すべての品物の吟味が終わったとき、6kgの耐重量のナップサックの価値の最大値は、maxValue[ITEM_NUM － 1][KNAP_MAX]に格納されています。mainメソッドの最後で、解を表示します。

Java
KnapsackDP.java

```
public class KnapsackDP {
  // ナップサックの耐重量
  public static final int KNAP_MAX = 6;

  // 品物の種類
  public static final int ITEM_NUM = 5;

  // 品物の名称
  public static char[] name = { 'A', 'B', 'C', 'D', 'E' };

  // 品物の重量
  public static int[] weight = { 1, 2, 3, 4, 5 };

  // 品物の価値
  public static int[] value = { 100, 300, 350, 500, 650 };

// 品物を吟味した直後の価値
public static int[][] maxValue = new int[ITEM_NUM][KNAP_MAX + 1];

// 最後に入れた品物
public static int[] lastItem = new int[KNAP_MAX + 1];

// item番目の品物を吟味した直後のナップサックの内容を表示するメソッド
public static void showKnap(int item) {
  int knap; // 0kg～6kgのナップサックを指す

  // 吟味した品物の情報を表示する
  System.out.printf("＜%c, %dkg, %d円を吟味した結果＞¥n",
  name[item], weight[item], value[item]);

  // ナップサックの耐重量を表示する
  for (knap = 0; knap <= KNAP_MAX; knap++) {
    System.out.printf("%dkg¥t", knap);
  }
  System.out.printf("¥n");

  // ナップサックに詰められた品物の価値の合計を表示する
  for (knap = 0; knap <= KNAP_MAX; knap++) {
```

プログラムの実行結果の例（第3世代）

```
下位50%を淘汰しました。
個体4と個体5を1の位置で交叉しました。
個体6と個体7を2の位置で交叉しました。
個体4の3の位置で突然変異しました。
個体7の0の位置で突然変異しました。
個体7の3の位置で突然変異しました。

＜第3世代＞
遺伝子                重量      価値      適応度
[0][1][0][1][0]     6kg     800円     800
[1][1][1][0][0]     6kg     750円     750
[1][1][1][0][0]     6kg     750円     750
[1][1][1][0][0]     6kg     750円     750
[0][1][1][0][0]     5kg     650円     650
[0][0][0][1][0]     4kg     500円     500
[1][1][1][1][0]    10kg    1250円       0
[1][0][1][1][0]     8kg     950円       0

＜ナップサックに入っている品物を表示する＞
B, 2kg, 300円
D, 4kg, 500円

＜解を表示する＞
重量の合計値 = 6kg
価値の最大値 = 800円
```

最も適応度の高い個体を最終的な解とする

10-1-3 遺伝的アルゴリズムの仕組みを説明するプログラムの概要

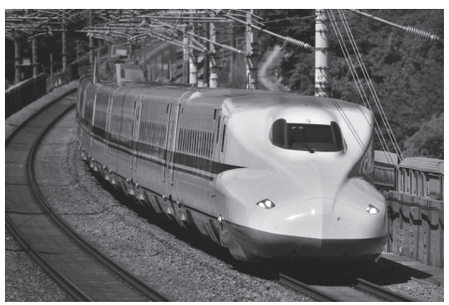
遺伝的アルゴリズムでナップサック問題を解くJavaのプログラムを作る前に、プログラムの概要を擬似言語で示しておきます。以下は、Javaのプログラムのmainメソッドの内容を擬似言語で示したものです。キー入力で指定された世代の数だけ、淘汰、交叉、突然変異を繰り返し、それぞれの世代における個体の内容を表示します。この繰り返しが終わったら、その時点で最も適応度の高い個体をナップサック問題の解として表示します。ここでは、変数や関数を示さずに、文章で処理内容を示しています。

擬似言語

```
/* プログラムの実行開始位置となるmainメソッド（ここから）*/
○main
・最大の世代をキー入力する
・ランダムに第1世代の個体を8つ生成する
・適応度を計算する
・適応度の大きい順にソートする
・個体の内容を表示する
・世代を1つ進める
■ 最大の世代以下である限り繰り返す
  ・適応度の大きい順にソートする
  ・上位50%の個体を下位50%にコピーして、下位50%を淘汰する
  ・下位50%にコピーした個体で交叉を行う
  ・下位50%にコピーした個体で突然変異を行う
  ・適応度を計算する
  ・適応度の大きい順にソートする
  ・個体の内容を表示する
  ・世代を1つ進める
■
・最も適応度の高い個体をナップサック問題の解として表示する
/* プログラムの実行開始位置となるmainメソッド（ここまで）*/
```

Quiz 遺伝的アルゴリズムがどこに使われている？

2007年7月1日にデビューしたN700系新幹線は、従来の700系から大幅なスピードアップを実現しています。N700系新幹線の設計では、トンネルにおける騒音を抑えるために遺伝的アルゴリズムが活用されています。どの部分の設計でしょうか。



解答は 283ページ にあります。