〔論　文〕

プログラミング言語の並行処理機能を活用するアルゴリズムの視覚化支援

佐塚　秀人*1

Visualization support for algorithms that utilize the concurrent processing function of programming languages

Hideto SAZUKA*1

Abstract

We propose a method to incorporate the animation of data structure and algorithm into its learning of which is the basic theme of computer science. Using the co-routine function for visualization graphics and algorithm processing, these two functions are parallelly operated and a programming style that is easy for beginners to understand is obtained. This paper proposes an algorithm visualization support using the Generator function, a co-routine function that has been introduced in ECMAScript 2015 (JavaScript).

Key Words：educational programming, concurrent programming, graphical programming, algorithm, data structure

1．緒　　言

プログラミングの初級教育は基本的な数値出力や文字出力の環境からスタートし，基礎的な内容を習得したのちにグラフィックス機能をはじめとするさまざまなメディア機能やインターネット上のリソースなどを駆使する環境に移行してくスタイルであった．しかし，小学生にプログラミング教育が導入される時代となり，導入時からグラフィックス機能を駆使した学習が一般的となりつつある．大学においても少なからずその影響を受け，数値や文字だけを扱ってきた『データ構造とアルゴリズム』の教育にも，ビジュアルな演出が求められている．学生に対して処理過程を視覚的に理解できるような支援が必要とされている[(1)(2)(3)]．

一方，プログラミング言語やプログラミング環境も変化しスクリプト言語や Web 上の環境も利用が一般的になっている．p5.js[(4)]は Java をベースした Processing[(5)]を Web ブラウザ上の JavaScript に対応させたもので，グラフィックス機能とオンラインで利用できる開発環境から，初級者向けのプログラミング学習環境として評価されている．

時代の変化の中，基礎プログラミング教育にも新しいスタイルが求められているが，ビジュアライズ対応はプログラムを複雑にし，本来学生に伝えたい簡潔でエレガントな記述を学習者に見えなくしてしまう問題をもたらしている．

グラフィックスを駆使したデータ構造のビジュアライゼーションの部分と，本来のアルゴリズム記述を明解に分離し，並行動作を実現できれば両者の学習を分けて進めることができる．並行処理記述は高度なスキルという扱いがされてきたが，近年のプログラミング言語には並行処理機能が容易に利用できる形で導入されおり，自然な形で学習できるならば基本機能として扱うことができると判断した．

初級者向けのアルゴリズム学習を進める手段として，特別な学習ツールや環境を用いることなく，容易に利用できる Web ブラウザとオンライン開発環境のみで，①アルゴリズム学習，②グラフィックス，③並行処理（同時処理），この３つのスキルを独立させバランスよく学ぶ手法について具体的なアプローチの例を提案する．

*1 情報ネットワーク工学科
令和元年11月７日受理

2．教育におけるプログラミング環境の変化

2・1　求められるグラフィックス環境

情報工学分野の初級教育において『データ構造とアルゴリズム』は従来から基盤をなすカリキュラムに位置づけられてきた．本来はプログラミング言語やプログラミング環境とは独立して扱われる内容であるが，実際の演習講義ではプログラミング言語やプログラミング環境に大きな影響を受けている．この科目が生まれた時代は Pascal や C 言語といったコンピュータのプリミティブな機能に基づく機能を前提にし，数値や文字出力を前提に教育内容が構成されていた．しかし，現在はこのようなプログラミングの基礎カリキュラムであってもグラフィックス機能や応用指向のプログラミング環境を使用する方向にあり，従来の考え方や教材・資料はそのままでは利活用できなくなっている．

久留米工業大学情報ネットワーク工学科では2017年度から 1 年次のプログラミングの導入教育環境として CUI 上の C 言語環境からアート・プログラミングを指向する Processing に移行し，WEB 上のプログラミングを対象とした 2 年次以降の授業でも p5.js を使用している（図 1 ）．

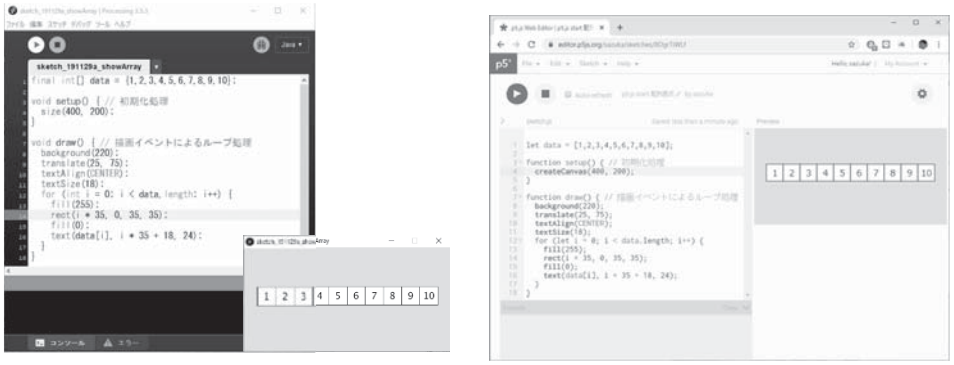

Fig. 1　Processing, p5.js の簡易 IDE によるプログラム環境

近年のアプリケーション作成のフレームワークは，画面描画のイベントサイクルに同期し描画ルーチンがループするスタイルが一般的になっている．GUI が一般化することで，コマンドラインから起動されメインルーチンから順に手続き的にプログラムが動いていくプログラムは実用から外れたスタイルとなった．仮想ターミナル上の CUI による演習環境は，スマートフォンやタブレット PC との異差を強く感じさせてしまう．

統合開発環境（IDE）も利用されるようになったが，GUI を駆使し高度な支援機能を提供はしてくれるが，そこで行うプログラム作成はテンプレート等の利用がブラックボックス化を招くことや，強力な支援機能も初級者の学習には利用できることもなく，起動の遅さや手続きの煩雑さは難しいイメージを演出してしまうこともある．

そのような中，Processing や p5.js は簡単なエディタとグラフィックス表示のみのシンプルな画面，目的をアート指向のアプリケーションに限定することで，フルセットの言語機能はないが教育環境として扱いやすいツールになっている．

2・2　Processing や p5.js 導入による効果

Processing や p5.js に採用されている描画イベントによるプログラム駆動のスタイルは，アニメーション・プログラムを身近なものにしてくれる．描画呼び出しごとパラメータを変更し図形の表示位置を動かすだけで，容易に動きのある画面を作ることができ，アイディアを生かした美しい幾何学アニメーションを実現できる．グラフィックス機能をライブラリではなく基本組み込み機能として提供することで，初級者にとってハードルとなるライブラリ管理を回避できる利点がある．

従来，数値や文字出力しかなかったプログラミング学習の例題は，図形描画を駆使した内容への移行がなされ，言語の文法や抽象的なプログラミング手法が中心あったものが，学習者が主体的に試行錯誤できる内容に変化している．そこには1980年代の PC の黎明期に Microsoft　BASIC[9]が提供してくれたシンプルだが初級者も楽しめた環境が想起される．

図形描画を利用した演習課題は，座標系の把握からはじまり，規則正しい図形配置のための繰り返し記述（for 文等による繰り返しの抽象化），図形の組み合わせる新たな関数を定義することで，視覚的にプログラムの構造化を学ばせることができる．図形描画や画像を操作する演習課題は，従来からいろいろな形で実施されていることもあり，比較的

自然な形でプログラミングの導入学習に利用できる．学生からも最初のステップとしては，印象よく受け入れられているようである．

3．データ構造とアルゴリズムのグラフィックス表現

3・1　アルゴリズム学習とグラフィックス活用の非整合

基礎を学んだ次のステップとして，アルゴリズムやデータ表現を学び，プログラミング技術と結びつけていく．このステップを通してプログラムやデータを構造的に理解し，性能の評価ができるようになっていく．以前はコンピュータ科学の基礎の中核であったが，近年は地味なカリキュラムとして学生からの評価は高くはないのが現状である．

教科書としてデータ構造とその操作が図解され丁寧に説明されたものが数多く出版されている．C言語やJavaを用いた具体的なプログラム例は提示されているが，プログラム例を実際に動かしても表示されるものは結果のみであり動作の理解にはつながらない．積極的にプログラムに手を入れ処理経過を表示するなどの演習が重要となる．

処理過程の理解へグラフィックス機能の活用を考えるが，Processingやp5.jsのような画面描画イベントで駆動されるスタイル（図2）と，一般的なアルゴリズム学習のための例題は，実際のところ整合性が悪いことがわかる．プログラムの基本構造を変更することで描画イベントに対応したプログラムへの記述変更は可能となるが，for文やwhile文のような標準的な繰り返し構造が使えなくなるため，初級者にとっては元のプログラム（アルゴリズム）の理解ができなくなってしまう．

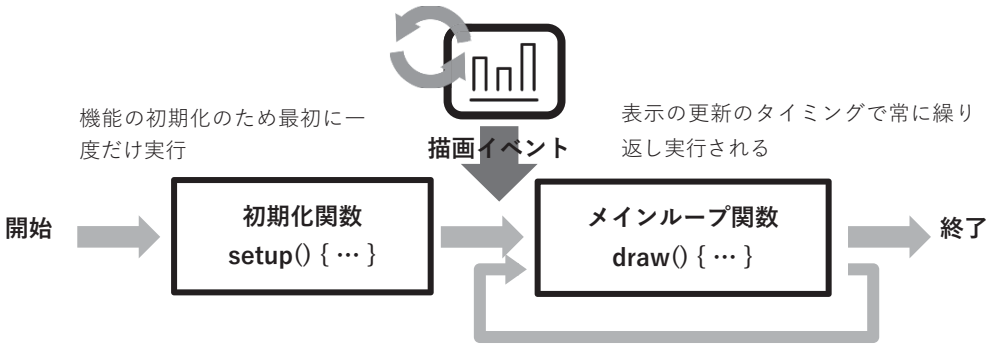

Fig. 2　描画イベントで駆動されるProcessing, p5.jsのプログラム

```
1   // JavaScriptによる一般的なバブルソートルーチン
2   function bubble_sort() {
3     for (let i = 0; i < N - 1; i++) {
4       for (let j = N - 1; j > i; j--) {
5         if (data[j - 1] > data[j]) {
6           let temp = data[j - 1];
7           data[j - 1] = data[j];
8           data[j] = temp;
9         }
10      }
11    }
12  }
```

```
1   // バブルソートをp5.jsの表示関数draw()化する
2   function draw() {       // 描画イベントによりループ
3     background(220);
4     show();               // 配列データの表示
5     if (i >= N - 1) {     // 外側ループ(i)の条件判断
6       noLoop;             // 終了(画面更新の停止)
7       return;
8     }
9     if (j <= i) {         // 内側ループ(j)の条件判断
10      i++;
11      j = N - 1;
12    }
13    if (data[j - 1] > data[j]) {
14      let temp = data[j];
15      data[j] = data[j - 1];
16      data[j - 1] = temp;
17    }
18    j--;
19  }
```

List 1　バブルソートプログラムをp5.jsのdraw()関数に変換した例

ソースリスト1の左側はJavaScriptで記述したバブルソートのルーチン（関数），右側はp5.jsで配列の内容を図2に示すp5.js形式の描画プログラム形式に変換したものである．左側では二重の繰り返し構造になること，2つの変数の更新とその条件を容易に説明し理解させることできるが，右側のプログラムが左側の同じ機能であることを理解することは難しい．Processingやp5.jsが優れたグラフィックス機能を持ち合わせていてもアルゴリズム学習にそれを活用することができない．

3・2　学びの構成

従来からのアルゴリズム記述スタイルを損なわず，動作を視覚的に表現したい．学習者の負担を減らすためデータ構造の表示機能をあらかじめ用意することが考えられるが，この学習カリキュラムに特化したものとなるであろう．なるべく単純な仕組みで構成し，理解しやすくかつ応用可能な手段を考えたい．

ここで扱いたい表現の対象は，データの並び，木構造，グラフ構造等であり，複雑な階層構造をもつものではない．基本図形を規則正しく並べる，線でつなぐ，色で塗り分けるような内容であり，それ自体初級者向きの内容である．一方，メインテーマであるアルゴリズムも，基礎的な検索や整列といった内容である．機能分離して作成でき，容易に連携ができれば，動作の仕組み理解も含めて興味をもてる学習カリキュラムが期待できる．

4．アルゴリズム処理と表示の並行動作

4・1　自然な並行処理

アルゴリズム本体から描画を独立させるためには，ルーチン相互のスイッチが必要となる．文字端末上であれば，出力関数やプローブの挿入で容易に表示ができる．しかし，描画イベントで動作している処理と，連続的に動作している本体処理は独立しており，表示のタイミング連携動作をさせなくてはいけない．２つのルーチンをスレッド（軽量プロセス）として並列動作させ，同期機能によりタイミング合わせることは可能であろう．しかし，主目的は図３のＢのように並列（parallel）に動作させる必要はなく，図３のＡのようにプログラムのコンテキストを分離するために交互に並行（concurrent）して動作する機能が利用できればよい．

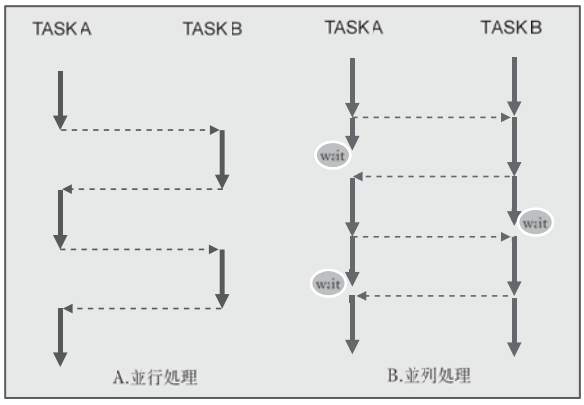

Fig. 3　並行処理と並列処理

小学生へのプログラミング教育で話題になっている Scratch[8]では基本機能として並行動作を積極的に利用する．Scratch は公開講座や高大連携授業でも教材として扱っているが，難しい概念抜きに同時に動かしたいスクリプトを普通に記述できる．クリティカルなリソースへのアクセス競合といった問題が重要でなれば難しい考え方ではない．図３のＡの並行同時処理ができれば，繰り返し処理の構造の途中で中断して表示を行うということが実現できる．

4・2　言語の並行処理機能

並行処理は機械語レベルでは決して難しくなく，コールフレームを切り替えるだけである．このような機能はコルーチン（co-routine）と呼ばれ1960年代の論文[6]で紹介されている．古いプログラミング言語にはコルーチン機能が実装されることがあったが，その後の汎用の言語への実装はなくなった．しかし近年，無限リスト，ジェネレータ／イテレータ，協調連携処理が注目され仕組みとして見直されている．

プログラミング言語におけるコルーチンやその応用機能の実装の状況を表１に示す．コルーチン機能は『ファイバ』や『ジェネレータ』というプログラミング機能として提供されることが多い．コルーチンは相互に中断が可能な並行動作であるが，ファイバやジェネレータはデータを生成側と消費側が順次データを渡しながら繰り返し処理を進める機能を提供する．繰り返しを抽象化するイテレータ機能と組み合わせることで自然にストリームや無限データ構造を扱うことができ，コーディングスタイルに変化を生み出している．

Table 1　主要言語のコルーチン対応

言語	対応年	対応バージョン	機能
C#	2005	C# 2.0	ジェネレータ
Python	2006	Python 2.5	ファイバ
Java	2014	Java8	ジェネレータ
ECMAScript(JavaScript)	2015	ECMAScript 2015 (ES6)	ジェネレータ
C++	2020	C++20	コルーチン

5．並行処理機能の利用

5・1　p5.js でのジェネレータの利用

各プログラミング言語におけるコルーチン機能の実装の形式はさまざまである．想定する利用目的の相違もあり，初級レベルのプログラミングに導入できる実装と，手続きが煩雑な実装がある．JavaScript，Ruby，Python などのスクリプト言語はダイナミックな実行機構がベースとなっており利用は，機能を部品化する関数の拡張として扱える．

p5.js は Processing を JavaScript 環境にポートしたものであるが，Web ブラウザの安定した環境の上に構築されており，クラウド上に開発環境が提供される点など，学習への利用については本家の Processing 以上の特徴・利点がある．そして，JavaScript には ECMAScript 2015（ES6）[7]からジェネレータ記述が言語仕様のレベルで実装されている．

プログラムの動きを表示したい（アルゴリズム処理をビジュアライズしたい）手続きをジェネレータとして記述し，表示データ列の生成を行う．描画ルーチン側は画面更新タイミングに従ってそれを順次表示していく．

ジェネレータ側は描画情報列というストリームデータの生成を行い，描画ルーチン（p5.js では draw 関数）はそのデータのコンシューマの立場となる．プログラミングの初級者が容易に理解できる概念とは思えないが，データ構造を中心とした抽象化の考え方として後に意味をもつと思われる．

5・2　関数からジェネレータへの変換

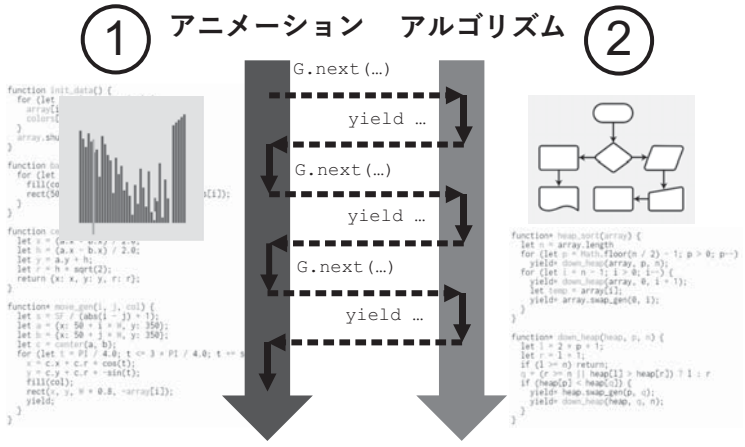

Fig. 4　表示部とアルゴリズム部の並行処理による分離

繰り返しデータ構造に対する繰り返し処理の抽象化をイテレータとよび，Python や Ruby 言語に導入され一般的な概念になって．イテレータに対応して動的にデータを生成するストリームを提供機能がジェネレータである．イテレータは配列やそれに類する集合的データ構造を対象として繰り返し機能を提供するが，ジェネレータは動的にデータを生成するデータ構造として抽象化される．その実装はコルーチンであり，生成側（ジェネレータ）と繰り返し側（イテレータ）は実行権を相互に受け渡しながら並行動作をする（図 4 ）．

アルゴリズム動作側は表示データのジェネレータであり，表示側（draw ルーチン）は表示データストリームのイテレータとして機能すると考えられる．言語の機能からみれば，コルーチン，ファイバ，ジェネレータと抽象度によりさまざまな名前とインターフェースが与えられているが，今回の目的には簡単に利用できることが重要で，初級者にその

概念の理解を期待するものではない．

JavaScript では言語の文法レベルでジェネレータ機能が実装されている．関数では return 命令により値を返してコンテキストは終了廃棄されるが，ジェネレータは yield 命令で動作を中断し値を呼び出し側に渡ことができる．呼び出し側ではジェネレータ・インスタンスに対する next メソッドを呼ぶことで，値を渡し動作を継続できる．関数呼びと異なる点はコンテキストを表現するジェネレータのインスタンスを呼び出し側で明示的に扱う必要があり，最初にインスタンスを生成後，next メソッドにより起動する．

JavaScript のジェネレータは言語レベルの実装であり関数定義とは区別される．関数定義は function キーワードから始まるが，ジェネレータは function*であり，ジェネレータ・インスタンスを生成する関数が定義される．最初の起動には next メソッドが必要であること，戻り値は yield 命令による中断と return 命令による終了を区別するために value と done の 2 つの属性をもつ連想配列であることには注意を必要とする．

関数からジェネレータへの変換は具体的な対応は別として形式的に変換が可能である．作業指示の例として以下のように説明をすることができる．

⑴ 関数（function）をジェネレータ（function*）に変更する．
⑵ 初期化ルーチンでジェネレータ関数を呼び出し，ジェネレータ・インスタンスを生成する．
⑶ ジェネレータ側で表示が必要なポイントに yield 命令を挿入する．
⑷ 描画ルーチンで画面の更新タイミングに合わせて next メソッドを呼び出す．

本来ジェネレータ機能はデータ列の順次生成であり，yield により順次データを渡すが，単純なデータ構造の表示（配列構造，木構造，グラフ構造，その他描画用アニメーションデータ）であれば，共通の大域データとして扱ってしてもかまわない．その場合，yield 命令，next メソッドは同期命令としての役割をする．

p5.js プログラムの構成はおおむね図 5 に示すパターンとなる．初期化関数 setup と描画ループ関数 draw は p5.js の標準関数である．draw 関数の中から next メソッド呼び出し制御をアルゴリズム部に移し，表示用のデータを得る．next メソッドの戻り値は連想配列（{value: [戻り値], done: [終了値(true/false)]}）を返す．done 属性が true の場合が関数における return による終了であり，yield 命令による中断の場合は false を返す．その場合の受け渡される戻り値は value 属性の値として返される．

```
// p5.js の初期化関数
function setup() {
          :
  g = some_algorithm_generator(...);  // アルゴリズム表示ジェネレータの生成
          :
}

// p5.js の表示イベントループ関数
function draw() {
          :
  if ((state = g.next()).done) noLoop();  // ジェネレータとのスイッチ, done 属性で終了を判断
  show(state.value);   // ジェネレータ側からの状態を元に表示を行う
          :
}

// アニメーション化したいアルゴリズムの表示ジェネレータ
function* some_algorithm_generator(...) {
          :
   yield some_value;
          :
}

// draw から呼ばれる表示関数
function show(param) {
  // データ構造の表示プログラム
          :
}
```

Fig. 5　アルゴリズム表示プログラムの基本フレーム

このような部分は初級者にとっては理解が難しい部分となるが，最初の段階でテンプレートとして提示することで対

応できるであろう．

5・3　関数呼び出し・再帰プログラムへの対応

関数呼び出しをジェネレータ形式に変換することで yield 命令による中断ができ，呼び出し側とジェネレータ側のタスクには相互に中断・継続を行うことができる．しかし，プログラムがさらに下位の関数を呼ぶ場合や再帰呼び出しをする場合も考慮しておく必要ある．JavaScript（ES2015以降）のジェネレータ機能は言語の拡張となっており，yield 命令は function*によるジェネレータ定義内に制限をされる．内部から別の関数（下位の関数）を呼ぶ場合，呼び出される関数内での中断はしない（yield 命令は使わない）か，呼び出す関数についてもジェネレータ化するかしなくてはいけない．後者の場合は呼び出しごとにジェネレータのインスタンスを生成し，呼び出し後は yield 命令による中断を中継しなくてはならない．これは単純な関数呼び出しであっても，ソースコードを難解な構成にしてしまう．特に再帰呼び出しをするようなプログラムはこの点が問題となる．

次にこの点について補足をする．これは JavaScript 処理系の実行効率を考えた仕様によるものであり対策はとられている．ジェネレータからさらに下位のジェネレータを呼び出す場合，下位ジェネレータに機能を移譲し，yield 機能を中継する呼び出し形式として yield*式が用意されている．ソースリスト 2 の例は再帰呼び出しを利用する例としてクイックソートのソースコードを載せている．この例では再帰的に quick_sort_generator を呼ぶ部分と，配列の要素の交換をする部分を，yield*を使用して機能を下位のジェネレータに移譲している．表示側に制御をスイッチする部分は swap_generator の中に記述されている．

```
1  // ジェネレータ化し表示中断可能なクイックソート
2  function* quick_sort_generator(array, min, max) {
3    if (min >= max) return;
4    let mid = Math.floor((min + max) / 2);
5    let pivot = array[mid];
6    let i = min, j = max;
7    for (;;) {
8      while (array[i] < pivot) i++;
9      while (array[j] > pivot) j--;
10     if (i >= j) break;
11     yield* swap_generator(array, i, j);
12     i++, j--;
13   }
14   yield* quick_sort_generator(array, min, i - 1);
15   yield* quick_sort_generator(array, j + 1, max);
16 };
17
18 // 配列の要素交換をジェネレータ化
19 function* swap_generator(array, a, b) {
20   yield [a, b];        // 交換前の表示
21   let temp = array[a];
22   array[a] = array[b];
23   array[b] = temp;
24   yield [b, a];        // 交換後の表示
25 }
```

List 2　クイックソートジェネレータの例

6．教育カリキュラムとしてのアプローチ

6・1　学習の進め方

現時点ではこのメカニズムを用いた専用のフレームワークや学習のための専用ソフトウェアの開発は考えていない．ジェネレータへの自動変換や，データ構造の表示機能をあらかじめ用意しておくことも考えられる．しかし，中途半端な演習ツールを提供しても，ツールを使用するだけの学習になってしまうこと，教育専用の環境となり実際のプログラミングの作業と乖離してしまう可能性が高い．

提案するメソッドは 3 つのプログラム部分をそれぞれ学生が考えてプログラミングしながら順に学んでいく．

(1)　基本的となるデータ構造の表示プログラム
(2)　本来の学習対象アルゴリズムの記述と変換

(3)　効果的な表示・アニメーション化プログラム

最初のステップ(1)はデータ構造の表示である．処理の対象となるデータ構造（配列，線形リスト，木構造等）を表示する部分として図５のテンプレートでは show 関数を作成しシンプルな形で表示するプログラムを作成する．ソート（整列）のような問題であれば図６のような四角形を要素の数だけ並べ，配列の値を表示させるだけでもいい．データの入れ替えのないような問題に場合は指定した要素に色を付けることも準備が必要かもしれない．簡単なグラフィックス命令と，繰り返し処理もこの段階で学べるであろう．

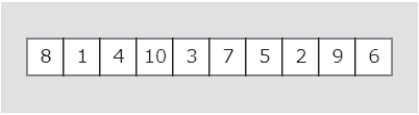

Fig. 6　基本的な配列表示

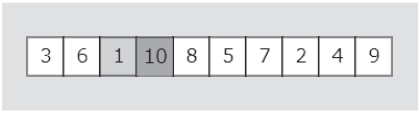

Fig. 7　交換位置の色による明示

次のステップ(2)はデータの基本操作である．ソート問題ではデータの入れ替えを行うコードをテストすることになるだろう．特定の要素を対象にした操作から，繰り返し文を利用した連続操作に移行する．この段階で表示部分と操作部分（アルゴリズム部分）を分離し，連続表示の機能を扱うことになる．プログラムが簡単な段階で記述方法を説明しておくことができれば，その先の作業結果を自然にモニタできる．そこからは目的とするアルゴリズムの学習ステップとなる．最低限のデータ表示はされているため，動作を確認しながらプログラム化をしていくことができ，間違いがあってもその問題個所を見つけることができる．

次のステップ(3)は，より効果的な表現方法を試すことになる．図６のような表示方法では，プログラムが連続動作していること，最終結果が正しいことは確認できるが，実際のプログラムの動きはわからない．図７のように操作箇所に色をつけてハイライトする課題を設定することでプログラムの詳細な動きを意識させることができる．さらに配列の要素を棒グラフで表示させるよういすれば図８，さらに要素を増やして要素を点で表現すれば図９のような表現を実現することができる．

このような視覚を実現する課題は，単純なアルゴリズムの演習課題を工夫できる課題に変え，さらにその観察によってアルゴリズムの理解を深める，あるいはアルゴリズムの改良といった思考に至ることが期待できる．

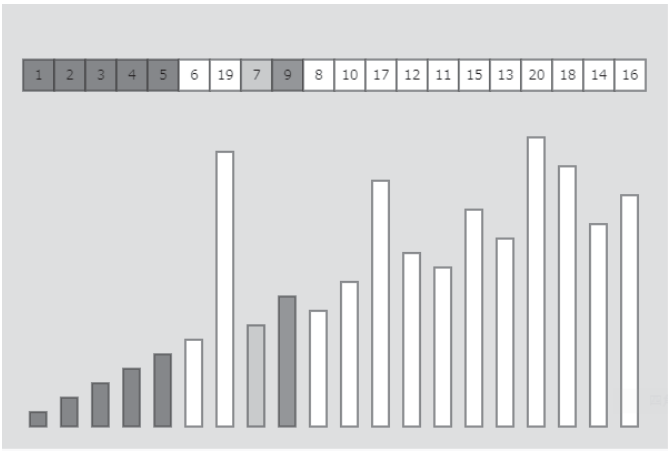

Fig. 8　棒グラフによるソート過程表示

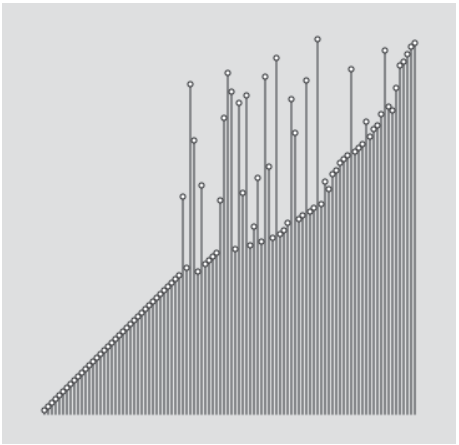

Fig. 9　ラインと点による配列要素の分布表示

６・２　例題と実践

本アプローチの例としてデータ構造とアルゴリズムで扱われる問題について，前述の手法に沿ってデータ構造とアルゴリズムの教科書で扱われる問題を実際にプログラム記述してみた．扱ったアルゴリズム手続きは10〜30行程度で記述できるものである．

- **探索（線形探索，２分探索）**

- ソート（整列過程表示，データ交換のアニメーション）
  単純ソート，バブルソート，シェーカーソート，選択ソート，挿入ソート，交換ソート，クイックソート，ヒープソート，その他
- 8 クィーン問題（バックトラック）
- ハノイの塔
- エラトステネスの篩（素数のフィルタリング）
- 線形リスト
- 2 分木の探索

ソートのアルゴリズムは種類が多く，同じスタイルで表示可能な内部ソートを実際に記述し動作させてみた．学習者がプログラムしその動作を知ることが目的ではあるが，データが収束する過程を比較してみせるだけでもデモ効果はあることがわかる．これらのアルゴリズムについての知識があれば，p5.js のプログラミング環境だけで簡単に作成できることも実感できた．表示部分はほぼ共通に利用でき，教科書通りにプログラムを記述するだけなので，教材作成方法としても負担にならない手法と云える．

探索の問題もデータの位置の変更がないので，探索の対象位置を見えるようにすることは，ソートよりも若干工夫が必要となる．表示のためのプログラムはスキル的にはソート問題と逆転するかもしれない．同様に，8 クィーン問題やエラトステネスの篩による素数のフィルタリング問題は，ちょっとした工夫によりデモ効果の高いプログラムに仕上がる例である．

ハノイの塔は再帰プログラムの純粋な問題であり対象とするデータ構造はない．表示のためのモデル作成が必要な例である．これは 3 D グラフィックス表示の導入することで楽しめる課題に発展させられるが，再帰アルゴリズムの学習に役立てるには別の視点が必要であろう．

6・3　並行理処理学習

本アプローチの主目的はアルゴリズムの学習へのグラフィックス機能の導入であり，そのメカニズムとして並行処理機能を活用することにあるが，並行処理の基礎事例を学ぶきっかけとして位置付けることができる．現在のコンピュータによる情報処理の基本が逐次処理化であったため，プログラミング技術の基礎が逐次処理として手続き記述することにあったと思われる．しかし，現代のコンピューティングにおいて並行処理・並列処理はたいへん多くの場面で現れ，その考え方やアプローチの重要度は増している．従来は高度で専門的なプログラミング技術として扱われてきたが，気がつけば子どもを対象にするプログラミング環境である Scratch には並行処理が取り入れられ，子どもは自然に理解してプログラムを作成している．コルーチンやその抽象化機能であるジェネレータやファイバのような機能が多くの言語に実装されていることからも，初級プログラミングの段階から導入することは新しいプログラミング学習カリキュラムの方向として考えられる．Scratch のようなプログラミング環境からの入門者にとっては理解ができる概念であり，それを生かしていくことは 1 つの課題となるだろう．

6・4　プログラミング教育への位置付けと課題

入門者にはなるべく新鮮な体験を与え，それを将来に向けての学習のモティベーションにしてあげたい．そのためのアプローチはさまざまであり，直観的で短時間で結果が得られるようなツールを提供することが課題なのかもしれない．高機能なツールが数多くある時代なので，抵抗感なくそれらを利用できることも重要である．本アプローチは，コンパクトな環境の中でステップを踏んで，多方面から自由度を広げていく方向でプログラミングの学ぶものである．教える側からは，学生の前で手品を演じるようにライブコーディングができるため，たいへん楽しいものであるが興味を持たない学生を置いてきぼりにしてしまうという問題もある．

十分な時間をとって，いろいろ試してみることができるようにするには，わかりやすいテキストや楽しめるポイントの設定が必要となるだろう．簡単な試行で短期的な結果を求めるよりは事例を重ねていくことが課題であろう．現在，1 年生を対象のセミナーでのアルゴリズム学習の中でこの手法を利用している．執筆時点では，p5.js の機能の習得から配列のデータの表示の段階であり成果を報告できる段階にはないが，実践をすすめて結果を得ていきたい．

7．結　　語

データ構造とアルゴリズム科目の学習に能動的に視覚化支援を取り入れる手段として，電子アートやビジュアルデザインを指向したプログラミング環境を活用する手段として，最近のプログラミング言語に取り入れられている並行実行の機能を活用する方法を提案した．

画面描画イベントに同期してループするアニメーション機能，ライブラリ導入を必要としない組み込みのグラフィックス命令群は，視覚的なフィードバックが即座に得られるメリットがある．しかし一方で，描画イベントのタイミングで起動されるスタイルが，従来からの基本的なアルゴリズムの記述とは相容れない問題があることがわかった．

データ構造のビジュアライズするプログラムと，アルゴリズムをわかりやすく表現するプログラム，この２つのニーズを共存させるため，最近の言語に積極的に取り入れられはじめたコルーチン機能を基盤におく並行処理機能を積極的に利用することで，描画とアルゴリズム記述を分離して扱えることを示した．

並行処理・並列処理は高度なスキルレベルを要求するという位置づけであったが，記述のパターンを限定し，初級者でも十分に扱える形にパターン化することで容易にプログラム記述ができること，①データ構造の表現，②アルゴリズム記述の学習，③より効果的な表示方法へのステップと考察，特別なツールを使用することなく自分の力で学んでいくカリキュラムを提供することを提案した．

教科書にある基本的なアルゴリズムについて，いくつかの例について対応できることを確認し，現在プログラミングの授業の中に取り入れながら効果を確認する段階にある．カリキュラムをテキスト化することと同時に，単純にデータ構造を表示するだけでなく，対話的な操作やより楽しめるアニメーションも学生と活動の中で実践をしていきたい．

文　　献

⑴　武田 寛昭，山下 英生，“C言語プログラムに対するアルゴリズム可視化システム”，広島工業大学紀要．研究編42巻（2008），pp. 247-253

⑵　大城 正則，永井 保夫，初学者向けプログラミング学習のための初等アルゴリズム視覚化システム”，情報教育シンポジウム論文集　No. 15（2018），pp. 104-111

⑶　鎌田 敏之，恩田 健司，本多 満正，兼宗 進 “中学・高校生に対し並行処理を意識させる自動化システム教材の開発”，日本情報科教育学会誌　No. 10- 1 （2017），pp. 33-44

⑷　Lauren McCarthy “p5.js Processing simplicity times JavaScript flexibility”, https://p5js.org/

⑸　Casey Reas, Benjamin Fry “Processing”, https://processing.org/

⑹　Melvin Conway “Design of a separable transition-diagram compiler” Communications of the ACM, Vol. 6, No. 7 (1963)

⑺　ECMA International, “ECMAScript 2015 Language Specification”, Standard ECMA-262 6th edition June 2015, http://www.ecma-international.org/ecma-262/6.0/ECMA-262.pdf

⑻　MIT, “Scratch”, https//scratch.mit.edu/

⑼　Kemeny, John G., Kurtz, Thomas E., “Back to BASIC: The History, Corruption and Future of the Language.”, (1985) Addison-Wesley Publishing Company, Inc. ISBN 0-201-13433-0