アルゴリズム演習のためのグラフ理論ライブラリ

松下 浩明*

Graph Theory Programming Library for Use in Exercises of Graph Algorithms

Hiroaki MATSUSHITA*

Abstract

We have developed a library for graph theory to use by lessons or exercises. This program library has comparatively simple structures so that we can master it in a short time. This library has the function of editing graphs in order to make creation of graph data easy. Moreover, this library has the functions in which graph data can be saved to files, be inputted from files and draw Graphs. We can describe intelligibly the Graph search functions, such as depth first search, using this library. As a result of our using this library by lessons, we confirmed that the opinions obtained were for the most part positive.

Keywords：Graph Theory，Programming，Library，Graph Algorithm

1．まえがき

グラフ理論のグラフは頂点集合とそれらを結ぶ辺集合で定義されるものである．グラフを用いると都市間の交通網やインターネットの通信網，人と人の関係，作業工程表などさまざまな現実問題が厳密にモデル化できるため，グラフは工学上有用な概念である[1)]．

情報工学や情報科学のカリキュラムの中では，「データ構造とアルゴリズム」や「情報数学」などの科目で，グラフ上のアルゴリズムの観点から講義される場合が多い[2)]．また，中学生を対象としたグラフ理論の講義例もある[3)]．

グラフのアルゴリズムを講義するとき，二つの方法がある．一つは理論的見地に沿って講義する方法であり，数学的素養を養う．もう一つは実際にプログラミングを行い，創造的素養を養う方法である．

工業系大学や高等専門学校など，実践的高度専門技術者の養成を目的とする学校では，理論的素養を基礎にして，プログラミングなどの演習に力点を置く．

*香川高等専門学校詫間キャンパス情報工学科

グラフのアルゴリズムを実際にプログラミング言語で表現する場合，一からグラフのデータ構造を構築するには大変な労力を要する．限られた講義時間や演習時間のなかでそれを行うのは得策でない．そのため，グラフの基本的操作を用意したグラフ理論ライブラリを利用する価値が生じる．

代表的なグラフ理論ライブラリとして，C++言語上で動作する Boost Graph Library や LEDA，Java 言語上で動作する JGraphT がある[4)〜6)]．Boost Graph Library は機能が豊富でさまざまな実現法に対応しているため，実用性の高いライブラリである．LEDA はグラフ理論，計算幾何を含む広範囲のライブラリである．記述しやすく，実用，教育の両面に対応している．JGraphT はグラフ理論の操作に特化し，比較的単純な構造をもつライブラリであるため，理解しやすい．

一方，演習でグラフ理論ライブラリを使用する場合，Boost Graph Library に代表される従来のグラフ理論ライブラリでは十分でない．最初の問題点はライブラリで用意すべき関数の揃え方についてである．

深さ優先探索の応用として，関節点問題を解くアルゴリズムの演習を考えてみる．関節点アルゴ

リズムは複数存在するので, 効率的ではないが理解しやすいアルゴリズムから理解しにくいが効率的なアルゴリズムへと授業展開を進めるのが妥当であろう. そのとき, プログラミングの複雑さも易しいものから難しいものへ連なっていると都合がよい. しかし, 従来のグラフ理論ライブラリを用いるとそのようにならない.

Boost Graph Library や JGraphT では visitor やそれと類似な概念を導入し, 深さ優先探索をどのような使い方にも対応できるようにしている[9]. この汎用性は教育上の使用においてはむしろ欠点になる. 授業展開上, 先に教える理解しやすいアルゴリズムのコーディングがあまり簡単にならないからである.

関数の揃え方についてのもう一つの観点は, 入出力関数と描画関数である. グラフの入出力関数や描画関数はグラフアルゴリズムの学習を容易にさせるために必要である. 従来のグラフ理論ライブラリは入出力関数や描画関数を別のライブラリやライブラリ利用者に委ねている. これは頂点, 辺への付加データを含むグラフデータの入出力関数や描画関数を汎用的に記述する難しさに起因している.

従来のライブラリにおける第 2 の問題点はプログラム誤りの見つけやすさに関するものである. 頂点や辺の削除について考えてみる. 繰り返し構文の中で, 何も考慮せず削除を行うと実行時エラーになる[9]. これは削除により, グラフの内部データ構造が変化し, その結果繰り返しが正しく行われなくなったためである. このエラーへの対処法は既知であり, ライブラリの使用に慣れた利用者はこのエラーを回避するプログラムをつくることができる. しかし, 授業の学習者にとっては余分な注意をしなければならなくなり, 学習者の注意が学習の本来の流れからずれることになる.

プログラム誤りの見つけやすさはグラフクラスの揃え方にも関係する. 連結グラフというグラフクラスは学習上よく現れるクラスである. しかし, 従来のライブラリでは無向グラフや単純グラフなどのクラスは用意されているが, 接続関係で特徴づけられる連結グラフというクラスは用意されていない.

学習者が連結グラフに対するアルゴリズムの演習を行うとき, 入力グラフに誤って非連結グラフを用いることは起こり得ることである. そのとき, プログラムの誤りがアルゴリズムに起因するのか, 入力データに起因するのか, 学習者にとって判断が難しい.

従来のライブラリにおける第 3 の問題点はアルゴリズムの途中結果を表示しないことである.

学習者はアルゴリズムを理解するために, アルゴリズムの最終実行結果のみならず, アルゴリズムの途中の実行結果も知りたい. しかし, 従来のライブラリではアルゴリズムの最終結果しか知りえない.

このような状況のもとで, グラフアルゴリズムの演習に利用可能な教育用グラフ理論ライブラリを開発した[10]. 本グラフ理論ライブラリはプログラミング言語 C#および Java から呼び出し可能なライブラリであり, 上記問題点を次のように解決している.

(1) 単純な構造

グラフの内部構造を頂点, 辺間の隣接リストで実現するものだけに限定した. このことにより, 短い授業時間内でも習得できるよう比較的単純な構造をもたせることができた.

(2) 学習項目に対応した関数群

汎用的な 1 個の関数を用意するのでなく, 学習項目に対応した複数の関数群を用意した.

たとえば, 深さ優先探索に関しては先行順の頂点列を戻す関数, 後行順の頂点列を戻す関数, 深さ優先探索木を戻す関数, 先行順, 後行順の入り混じった頂点, 辺列を戻す関数, 汎用的な深さ優先探索関数を用意し, 学習段階に応じて適切に関数を使い分けできるようにした.

(3) グラフエディタと入出力関数

グラフデータは原理的には頂点間の隣接行列で表現できる[7]. しかし, 隣接行列で表現されたグラフデータは誤りやすく, また, データの内容を判別するのが困難である. そこでグラフデータの作成を容易にするために, さまざまなグラフエディタが開発されている[8]. 本論文でも, 本グラフ理論ライブラリを用いてグラフエディタを開発した. また, グラフエディタで作成したグラフデータを入出力する関数を用意した.

(4) 描画関数

アルゴリズムの動作確認を容易にするために, 描画関数を用意した. このことにより, アルゴリズムの実行途中や実行終了時のグラフの様子を視覚的に見ることができる.

(5) グラフクラス

汎用的なグラフクラスのほか，学習でよく使われるグラフクラスを 11 種類用意した．これらのグラフクラスを用いることにより，グラフデータの誤りに起因したプログラムエラーを減少させることができる．

(6)　アルゴリズムの途中動作の確認

アルゴリズムの途中まで実行できる関数群を用意した．これにより，アルゴリズムの最終結果のみならず，途中の実行結果を得ることができる．また，これらの関数群をグラフエディタに組込むことにより，アルゴリズムのステップごとの動作を見ることができる．

(7)　頂点，辺データの拡張性

ライブラリを利用して応用プログラムを作成するとき，頂点，辺に付加データが必要な場合がある．本グラフ理論ライブラリでは，頂点，辺に付加データを容易につけることができる．また，付加データつきのグラフの入出力を行うことができる．

本論文では，2.でアルゴリズム演習，特にグラフアルゴリズム演習の目標と方法について述べる．3.で本グラフ理論ライブラリの基本的な構成と機能を述べる．4.で本グラフ理論ライブラリによるグラフアルゴリズム演習の支援例について述べる．5.でグラフ理論ライブラリを実際の授業に適用した実施例とその評価について述べる．

2. アルゴリズム演習の目標と方法

コンピュータプログラムのアルゴリズムの題材としては，ソーティングアルゴリズムとグラフアルゴリズムがよく取り上げられている [2)]．これらのアルゴリズムは時間計算量の解析など，理論的側面を教授する場合が多いが，実際にプログラムを組み，その動作を実地に理解することも大切である．特に高等専門学校などの実践的高度専門技術者を養成することを目的とする学校においては，アルゴリズム演習によって，アルゴリズム設計能力，アルゴリズムに沿ったプログラミング能力，検査能力を涵養することが目標になる．

ところで，ソーティングアルゴリズムは，それをプログラミングする場合，配列や再帰プログラミングなどプログラミングの基礎を理解しておれば，おおかた用が足りる．

一方，グラフアルゴリズムでは，グラフ，頂点，辺など，扱うデータの抽象度がソーティングアルゴリズムの扱うデータより高いため，プログラミングをより困難なものにしている．

グラフアルゴリズムを題材として，アルゴリズム演習を構成するとき，演習の展開方法としては，アルゴリズムの理解向上を目的に展開する方法とアルゴリズムの設計能力向上を目的に展開する方法が考えられる．

(1)　アルゴリズムの理解向上を目的とする演習

アルゴリズムの理解向上を目的とする演習の授業展開例を表１に示す．分節１では学習者にアルゴリズムを説明し，教師が例題を解く．分節２では教師があらかじめ用意したグラフに対し，学習者に机上で課題を解かせる．分節３ではコンピュータを用いて課題のグラフデータを作成する．分節４ではコンピュータによる動作シミュレーションを行い，分節２で学習者が解いた解と一致するか調べさせる．グラフが複雑な場合や授業時間が短い場合はグラフデータを教師があらかじめ用意することにより，分節３を省略してもよい．

グラフアルゴリズムの題材の代表例の１つである最小木問題では，プリムのアルゴリズムとクラスカルのアルゴリズムの２つのアルゴリズムがあるので，分節２から文節４を２回繰り返すことになる．このような授業展開を実施するためにはまず，グラフデータを容易に作成できることが必要である．次にアルゴリズムの最終的な実行結果ばかりでなく，アルゴリズムの途中の実行結果がステップごとに確認できなければならない．

(2)　アルゴリズムの設計能力向上を目的とする演習

アルゴリズムの設計能力向上を目的とする演習の授業展開例を表２に示す．分節１では学習者にアルゴリズムを箇条書きレベルの疑似コードで説明する．この疑似コードは自然言語に近いものとする．また，アルゴリズムとは関連づけずに疑似コードとプログラミングコードの対応関係を説明する．分節２では学習者に机上でプログラム記述を行わせる．分節３では学習者がコンピュ

表１　授業展開１

分節	授業内容	授業形態
1	アルゴリズムの説明	一斉
2	実行課題の解答	個別（机上）
3	グラフデータの作成	個別（PC）
4	動作シミュレーション	個別（PC）

表 2　授業展開 2

分節	授業内容	授業形態
1	アルゴリズムの説明	一斉
2	作成課題の解答	個別（机上）
3	グラフデータの作成	個別（PC）
4	プログラミング	個別（PC）

ータを用いて，テストデータとして適切なグラフデータを作成する．分節４ではコンピュータ上でプログラムコードの入力とコンパイル作業を行わせ，エラーを訂正する．分節４において，アルゴリズムと直接関係ないコード部分についてはあらかじめ学習者に知らせておく．

このような授業展開を実施するためにはまず，グラフデータはアルゴリズムが前提としたデータであるか否かを容易に検査できる必要がある．また，グラフ，頂点，辺などのグラフオブジェクトが容易に作成できなければならない．さらに，アルゴリズムが正しく動いていることを知るためにアルゴリズムの動作を直感的に把握できなければならない．

3．グラフ理論ライブラリの構成と機能

本グラフ理論ライブラリには C# 言語版と Java 言語版の２種類があるが，本論文の説明は主に C# 言語版で行う．

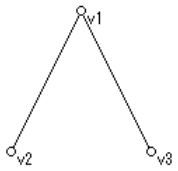
本グラフ理論ライブラリはループ，多重辺をもつ擬グラフを扱う．また，無向辺と有向辺がまじった混合グラフを扱う．

3.1　グラフ，頂点，辺の構築

グラフの型は Graph<V, E> である．ここで V は頂点の型を表すパラメータである．頂点の基底クラス BaseVertex とインタフェース IVertex を継承している．E は辺を表すパラメータで，頂点と同様な継承関係がある．頂点，辺の既定の型として Vertex，Edge を用意している．

グラフ，頂点，辺に共通の属性には名前，位置，重みがある．また，辺のみに存在する属性として，２つの端点および無向辺か有向辺かを示すタグがある．

グラフ G に新たな頂点を構築するには，G.NewVertex()を用いる．G に頂点 v1 と v2 を結ぶ無向辺を構築するには G.NewEdge(v1, v2)を，



(a) Graph

```
static void Main(string[] args)
{
   Graph<Vertex, Edge> G =
       new Graph<Vertex, Edge>();
   Vertex v1 = G.NewVertex("v1");
   Vertex v2 = G.NewVertex("v2");
   Vertex v3 = G.NewVertex("v3");
   v1.X = 100; v1.Y =  50;
   v2.X =  50; v2.Y = 150;
   v3.X = 150; v3.Y = 150;
   Edge e1 = G.NewEdge(v1, v2);
   Edge e2 = G.NewEdge(v1, v3);
}
```

(b) Program

図 1　グラフの構築

有向辺を構築するには G.NewDiEdge(v1, v2)を用いる．図１(a)のグラフを構築するプログラムを図１(b)に示す．

3.2　グラフ，頂点，辺の基本操作

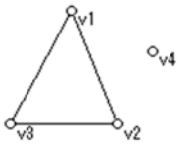
グラフ，頂点，辺の基本操作関数にはグラフへの頂点，辺の挿入，グラフからの頂点，辺の削除，頂点，辺がグラフに含まれているか否かの判定がある．

また，グラフ内の全頂点，全辺の参照は G.Vertices()，G.Edges()で行う．それぞれ List<V>, List<E>型である．また，G.VertexCount, G.EdgeCount で全頂点の総数，全辺の総数を表す．

3.3　削除操作を含む処理の繰り返し処理

グラフに属する全頂点，全辺を対象とした繰り返し処理において，各処理のなかに削除操作を行う場合，プログラムの記述に特別な方法が必要であり，学習者に過多な注意をしいる．

グラフのなかから孤立点（他のどの頂点とも隣接していない頂点）を削除する処理を例に取り，



(a) Graph

```
グラフ G の孤立点を取り除くアルゴリズム
(1) グラフ G を入力する
(2) グラフ G の各頂点 v に対し
      v の次数が 0 ならば
        G から v を削除する
```

(b) Pseudo Program

```
static void Main(string[] args)
{
   Graph<Vertex, Edge> G =
      new Graph<Vertex, Edge>();
   G.Input("graph.dat");
   foreach (Vertex v
             in G.EachVertex) {
      if (G.DegreeOf(v) == 0)
         G.Remove(v);
   }
}
```

(c) Pseudo Program

図 2　孤立点の削除処理

説明する．図 2 (a)のグラフにおいて，頂点 v4 が孤立点である．

孤立点を削除する処理は，グラフの全頂点をたどっていき，頂点の次数が 0 ならば，それを削除する方法になる．図 2 (b)に疑似言語による孤立点削除処理を示す．図 2 (c)は本グラフ理論ライブラリを用いて，コード化したものである．このプログラムは実行時エラーを起こす．Boost Graph Library や JGraphT を用いて，同様のプログラムを作成しても，やはり実行時エラーを起こす．

その理由はグラフの全頂点を対象とした繰り返し処理の中で，グラフの内部データ構造を変更したためである．このような処理においては，全頂点のコピーを作成し，コピーされた頂点に対し，繰り返し処理を行わなければならない．しかしながら，このようなプログラムコーディング上の注意事項は煩雑であり，また，実行時エラーが起きたとき，その原因が分かりにくく，学習者に余分な負担をしいることになる．

```
foreach (Edge e in G.EdgesOf(v))
{
   Vertex w = e.Opposite(v);
}
```

図 3　隣接頂点の参照

本グラフ理論ライブラリでは，グラフの内部データ構造を直接アクセスして，全頂点，全辺をたどる G.EachVerex, G.EachEdge 関数と，全頂点，全辺のコピーを作り，このコピー上の頂点，辺をたどる G.Vertices, G.Edges 関数を用意し，前者を応用プログラム作成用，後者を学習用に提供した．学習者は後者を用いることにより，アルゴリズムの理解に集中できるようにした．

3.4　接続辺，隣接頂点の参照

頂点 v に接続する辺集合を参照するには G.EdgesOf(v)を用いる．G.EdgesOf(v)の値はそれぞれ List<E>型である．また，G.EdgeCountOf(v)で頂点 v に接続する辺集合の総数を表す．v が辺 e の端点であるとき，e.Opposite(v)で辺 e のもう一つの端点を表す．図 2 に頂点 v から，接続辺を経由して隣接頂点を参照するプログラムを示す．

3.5　グラフの入出力

グラフのファイルからの入力はファイル名を入力ダイアログで指定する関数とファイル名およびパスを文字列で指定する関数がある．本グラフ理論ライブラリの Java 版では，インターネット上での利用を考えて，ユニフォームリソースロケータを指定できる関数も用意している．

グラフ G のファイルへの出力はファイル名を出力ダイアログで指定する関数とファイル名およびパスを文字列で指定する関数で行う．

3.6　グラフの描画

グラフの描画はグラフィックスを指定して描画する関数が基本である．そのほか，基準点を指定する描画関数や頂点，辺の色，大きさなどを指定できる描画関数がある．

4. アルゴリズム演習の支援例

4.1 グラフデータの作成支援

グラフデータは頂点間の隣接関係を表現する隣接行列や, 頂点と辺間の接続関係を表現する接続行列で表現することができる. しかし, グラフがすこし大きくなると間違いやすくなる.

グラフデータの作成のために, グラフエディタを用意すると便利である. グラフデータの作成支援のために, 本グラフ理論ライブラリを用いて作成したグラフエディタを用意した. グラフエディタは本グラフ理論ライブラリ用のグラフを作成, 編集することができる.

方眼紙や地図を下敷きにする下絵機能によって, 規則的なグラフや地図情報にもとづいたグラフの作成が容易である. また, ランダムグラフを自動生成する. 図４にグラフエディタの外観を示す.

4.2 アルゴリズムの動作シミュレーション

アルゴリズムの動作を理解するには, ステップごとの動作シミュレーションが有効である.

グラフエディタには下方にトラックバーがついており, アルゴリズムを選択すれば, エディタで編集中のグラフに対し, 動作シミュレーションを行うことができる.

トラックバーのスライダを左端に合わせると, 全実行が行われる. ２番目の目盛りにスライダを合わせるとアルゴリズムの実行前の状態になる. 2 + i 番目の目盛りにスライダを合わせるとアルゴリズムの i ステップ後の実行状態になる.

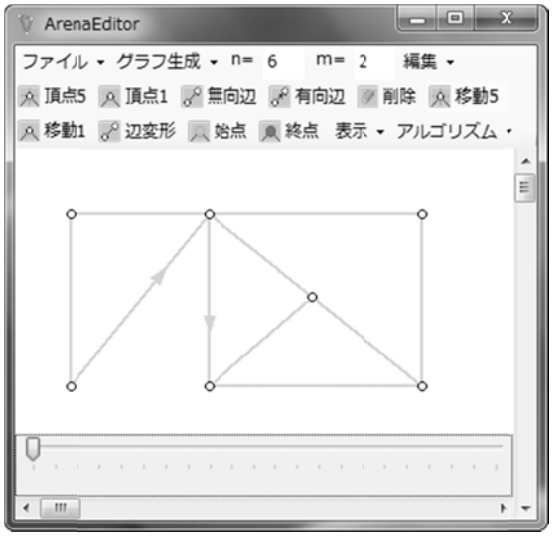


図４　グラフエディタ

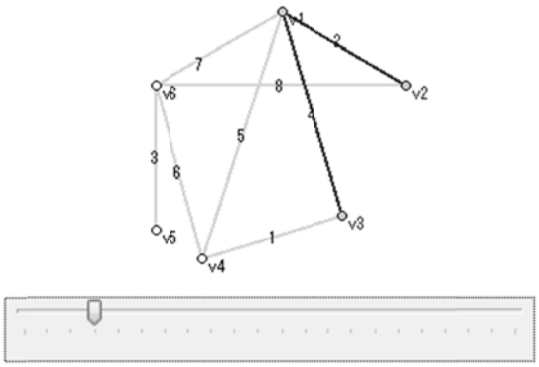
図５(a) は最小木問題を解くプリムのアルゴリズムの２ステップ目の実行表示である. 頂点集合 {v1} から出る重み最小の辺（v1, v2）を最小木の辺として選び, 次に頂点集合 {v1, v2} から出る重み最小の辺（v1, v3）を最小木の辺として選んだところである.

図５(b) はクラスカルのアルゴリズムの２ステップ目の実行表示である. 全体の辺のうち, 重み最小の辺（v1, v2）を選び, 次に２番目に重みが小さい辺（v3, v4）を選んだところである.

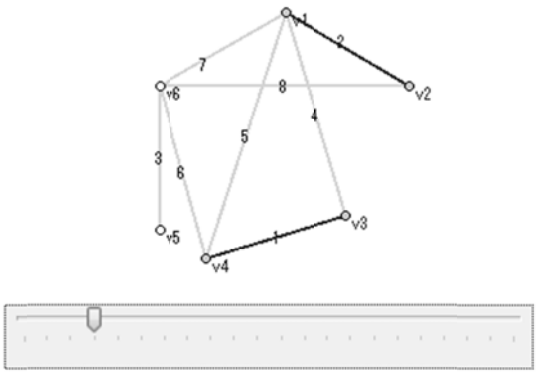
このようにステップごとの動作を表示すれば, 学習者は個々のアルゴリズムの動作ばかりでなく, 異なるアルゴリズムの動作の違いをたやすく理解することができる. ステップごとの動作シミュレーションは現在, 深さ優先探索や最小木問題, 最短経路問題, 巡回セールスマン問題などで実現されている.

4.3 深さ優先探索学習への支援

深さ優先探索はグラフ内の頂点や辺を接続関係にしたがって探索する基本的方法であり, グラフアルゴリズムの題材の代表例の１つである.

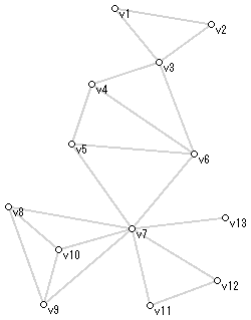


(a) Prim Algorithm



(b) Kruskal Algorithm

図５　アルゴリズムの動作シミュレーション



(a) Graph

```
static Graph<Vertex, Edge> G =
  new Graph<Vertex, Edge>();
static List<Vertex> preorder =
  new List<Vertex>();
static   Dictionary<Vertex,   bool>
visit = new
  Dictionary<Vertex, bool>();
static void Main(string[] args)
{
    G.Input("graph.dat");
    Vertex start =
      G.GetVertex("v4");
    foreach (Vertex v
             in G.Vertices)
        visit[v] = false;
    DFS(start);
}
static void DFS(Vertex v)
{
    visit[v] = true;
    preorder.Add(v);
    foreach (Edge e
             in G.EdgesOf(v)) {
        Vertex w = e.Opposite(v);
        if (!visit[w]) DFS(w);
    }
}
```

(b) Algorithm

```
v4 v3 v2 v1 v6 v7 v5 v8 v9 v10 v11 v12 v13
```

(c) Output

図 6　深さ優先探索アルゴリズム

それに基づいたさまざまなアルゴリズムが提案されている. 深さ優先探索を用いるとさまざまな角度から議論を展開できるため, アルゴリズム演習の題材としては適している.

深さ優先探索の学習は大きく２つに分かれる. １つ目は深さ優先探索アルゴリズムの学習であり, ２つ目は深さ優先探索を応用したアルゴリズムの学習である.

4.3.1　深さ優先探索アルゴリズムの学習

深さ優先探索アルゴリズムの学習では, 深さ優先探索の概念の学習と深さ優先探索の実現法の学習が行われる. 前者は表１の授業展開１で, 後者は表２の授業展開２で教授する. 深さ優先探索の実現法には再帰を用いる方法とスタックを用いる方法がある.

図 6(b) は本グラフ理論ライブラリを用いて, 深さ優先探索を再帰法で実現した１例である. 図６(a) のグラフに対し, 図６(c) の出力が得られる.

このように, 本グラフ理論ライブラリを用いれば, 深さ優先探索の実現法を実際にプログラミングして確かめることができる.

4.3.2　深さ優先探索を応用したアルゴリズムの学習

深さ優先探索を応用したアルゴリズムとしては, 無向グラフの連結成分を求めるアルゴリズム, 無向グラフの関節点を求めるアルゴリズム, 無閉路有向グラフに対するトポロジカルソートアルゴリズムなどがある.

ここでは, 無向グラフの関節点を求めるアルゴリズムを例にとり, 本グラフ理論ライブラリを用いた学習支援法を述べる.

グラフの関節点とは, グラフからその頂点とその頂点に接続する辺を取り除いたとき, グラフの連結成分の個数が増加する頂点のことである. 図６(a) のグラフでは頂点 v3, v7 が関節点である. 関節点を求めるアルゴリズムは３つ考えられる. 定義に基づいた方法, 深さ優先探索の性質に基づいた方法, 効率のよい方法である. それらを関節点アルゴリズム１, ２, ３と呼ぶことにする. 表３に関節点アルゴリズムの授業展開例を示す.

表３　授業展開３

分節	授業内容	授業形態
1	関節アルゴリズム１の説明	一斉
2	作成課題１ を解く	個別（机上）
3	グラフデータの作成	個別（PC）
4	プログラミング１	個別（PC）
5	関節アルゴリズム２の説明	一斉
6	作成課題２ を解く	個別（机上）
7	プログラミング２	個別（PC）
8	関節アルゴリズム３の説明	一斉
9	作成課題３ を解く	個別（机上）
10	プログラミング３	個別（PC）

(1)　定義に基づいた方法とそれに対する学習支援

定義に基づいた方法では，グラフから頂点を取り除き，連結成分の個数が増えるかどうか調べる．図７にそのアルゴリズムを示す．

本グラフ理論ライブラリでは，連結成分を調べるために深さ優先探索関数G.VerticesDFS(v) が用意されている．G.VerticesDFS(v) は頂点 v に連結している頂点を深さ優先探索でたどり，その頂点列を戻す．

(2)　探索木に基づいた方法とそれに対する学習支援

探索木に基づいた方法では，頂点 v を開始点とした深さ優先探索木を構成し，探索木における頂点 v の次数が２以上ならば頂点 v は関節点，そうでなければ関節点でないと判定する．図８に探索木に基づいたアルゴリズムを示す．

本グラフ理論ライブラリには，探索木を構成するのに深さ優先探索関数 G.GraphDFS(v) が用意されている．G.GraphDFS(v) は頂点 v に連結している頂点を深さ優先探索でたどり，その探索木を戻す．

(3)　効率のよい方法とそれに対する学習支援

効率のよい方法では，グラフ G の各頂点 v に対し，次に定義する値 low[v] を深さ優先探索順に計算し，関節点か否か調べる．

low[v] の定義

low[v] は次の (a)～(c) のうちの最小値である．

(a) v の探索番号

(b) w の探索番号，ただし，w は辺 (v, w) が後退辺である任意の頂点

(c) low[z]，ただし，z は v の任意の子

頂点 v がグラフ G の関節点か否か判定する
アルゴリズム 1
(1) v が孤立点ならば，偽を戻す
(2) グラフ G のコピーH をつくる
(3) v に隣接する頂点のひとつを w とする
(4) H から v を削除する
(5) w から G および H 内を深さ優先探索でたどり，それぞれの連結成分 C(G)，C(H)を求める
(6) C(G) の頂点数 - 1 が C(H) の頂点数と等しければ偽を戻す．
そうでなければ，真を戻す

図７　関節点アルゴリズム１

頂点 v がグラフ G の関節点か否か判定する
アルゴリズム 2
(1) v を開始点として G を深さ優先探索し，探索木 T を得る
(2) T において，v の次数が 2 以上ならば関節点，そうでなければ関節点でないと判断する

図８　関節点アルゴリズム２

グラフ G のすべての関節点を求める
アルゴリズム 3
(1) 深さ優先探索を行い，探索番号，探索木情報，辺の種類情報を得る
(2) G の各頂点 v に対し，low[v] を計算する
(3) G の頂点 root が関節点か否かを，アルゴリズム 2 を用いて調べる
(4) G の root 以外の各頂点 v については，v の子 w で，low[w] ≧ v の探索番号 なる w が存在すれば，v は関節点である

図９　関節点アルゴリズム３

ここで，探索番号は深さ優先探索を行った順に１からつけた番号である．図９に効率のよいアルゴリズムを示す．

効率のよいアルゴリズムでは，各頂点の探索番号や各辺が探索木を構成する辺か，後退辺かを知る必要がある．

本グラフ理論ライブラリには，探索番号やグラ

フの辺の種類を調べる深さ優先探索関数 G.ResultDFS(v) が用意されている．

4.4 グラフのクラスとアルゴリズム作成支援

グラフアルゴリズムが入力するグラフは連結グラフであるとか無閉路グラフであるとか制約条件のついている場合が多い．ところがアルゴリズム演習で入力グラフを学習者に個々に作成させると，制約条件に反するグラフを作成することがある．

アルゴリズム演習において，作成したプログラムをデバッグするとき，生じたエラーがアルゴリズムの誤りにのみに起因する場合は，学習者は比較的容易にエラーに対処できる．しかし，入力グラフが制約条件に反している場合，学習者がエラーの原因をつかむのは難しい．

このような状況を回避するために，本グラフ理論ライブラリではさまざまな制約条件をもつグラフのクラスをあらかじめ定義している．表４に本グラフ理論ライブラリで用意されているグラフのクラスを示す．

本グラフ理論ライブラリで扱うグラフのクラスは二分類される．ひとつは Graph で，もう一方は ImmutableGraph およびそれから継承されたクラスである．Graph はすべての基本操作を実行することができる．ImmutableGraph は基本操作のうち，頂点，辺の追加，削除を実行することができない．

Graph から制約条件のあるグラフへの変換は構築子で行われ，その変換時に，グラフの制約が満たされているか否かが検査される．学習者は検査済みの入力グラフに対し，アルゴリズムを作成する．このことにより，学習者はアルゴリズムの誤りにのみ起因したエラーに対処することができる．

表４　グラフのクラス

グラフクラス	意　　味
Graph	混合，擬グラフ
ImmutableGraph	混合，擬グラフ
Pseudograph	無向，擬グラフ
Multigraph	無向，多重グラフ
MixedGraph	混合，単純グラフ
SimpleGraph	無向，単純グラフ
DirectedGraph	有向，単純グラフ
ConnectedGraph	無向，単純，連結グラフ
Block	無向，単純，２重連結グラフ
Dag	有向，単純，無閉路グラフ
Forest	無向，単純，無閉路グラフ
Tree	無向，単純，連結，無閉路グラフ

4.5 付加データと応用プログラム作成支援

グラフ理論ライブラリを少し規模の大きい応用プログラムの作成に利用しようとすると，頂点や辺に付加データを加えることが必要になる．

本グラフ理論ライブラリでは，頂点に付加データを付け加える場合，頂点の基底クラス BaseVertex および頂点インタフェース IVertex から継承したクラスを定義し，そのクラスのメンバ変数として，付加データを定義する．また，辺に付加データを付け加える場合，辺の基底クラス BaseEdge<V>および辺インタフェース IEdge から継承したクラスを定義し，そのクラスのメンバ変数として，付加データを定義する．

AVertex，AEdge が BaseVertex，IVertex および BaseEdge<V>，IEdge から継承したクラスであるとき，AVertex，AEdge はメンバ関数として，引数のない構築子，付加データを頂点，辺に設定する関数，頂点，辺上の付加データを文字列に変換する関数を定義しなければならない．本グラフ理論ライブラリはこれらの関数を用いて，付加データのついたグラフのファイル入出力を行う．

図 10 に付加データを用いた応用プログラムの例を示す．このプログラムは京都市内の観光ルート検索プログラムである．学生が卒業研究として作成したものである．

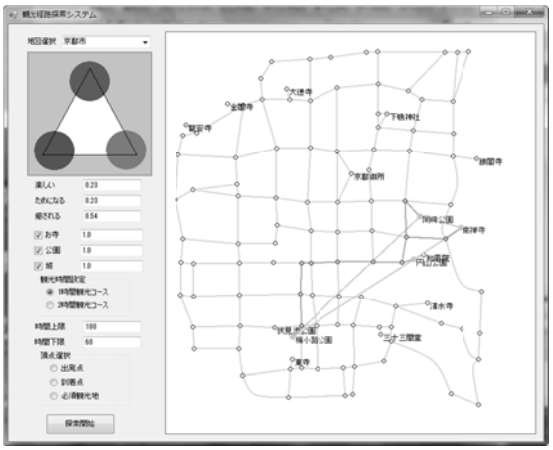


図 10　応用プログラム

5. 授業への適用

5.1 授業の概要

本グラフ理論ライブラリを高等専門学校専攻科１年の授業で 2012 年前期に使用し，評価した．実施した授業科目は「データ構造とアルゴリズム」である．この科目は１回の講義時間が 90 分であり，半年 15 回の授業を行う．

授業は２部からなる．第１部では，まず，講義形式でプログラミング言語 C#の解説と基本的データ構造を解説する．次に，「C#解説本」の作成演習を１班 4〜5 名からなるグループで行う．平均 30 ページの「C#解説本」ができあがる．

第２部では，まず，講義形式でソートアルゴリズムやグラフアルゴリズムを解説する．次に，グラフアルゴリズムの演習を行う．

2012 年度のグラフアルゴリズムの演習課題は最短経路問題である．１班 2~3 名からなるグループで，グループごとに異なる国を対象として，ある都市から他の都市への最短経路問題を解く．本グラフ理論ライブラリはこの演習で用いる．最短経路関数についてはライブラリで用意されている関数ではなく，学生が自らコーディングした関数を用いる．

図 11 はあるグループのプログラム作成例である．フランスを対象とした都市間の距離と移動に要する時間を求め，また，移動経路を表示している．

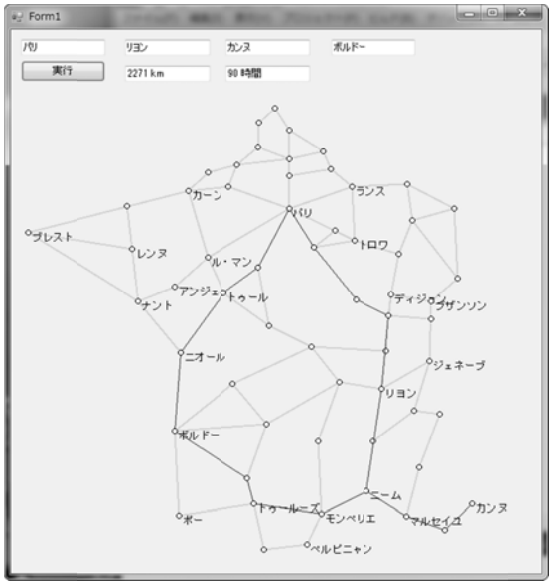


図 11　巡回路

問１　グラフエディタを用いた地図データ作成

評価段階	5	4	3	2	1
人数	7	5	7	0	0

問２　本グラフ理論ライブラリライブラリを使用するための参照設定

評価段階	5	4	3	2	1
人数	7	5	6	1	0

問３　グラフ，頂点，辺オブジェクトに対する理解

評価段階	5	4	3	2	1
人数	8	5	5	1	0

問４　ダイクストラの最短経路アルゴリズムの理解

評価段階	5	4	3	2	1
人数	5	8	5	1	0

問５　ダイクストラの最短経路アルゴリズムを，本グラフ理論ライブラリを用いて作成するプログラム演習

評価段階	5	4	3	2	1
人数	10	3	5	0	1

図 12　アンケート集計結果　1

問６　授業内容は講義と演習ではどちらがよいか
5 を演習の方がよい，1 を講義の方がよいとする

評価段階	5	4	3	2	1
人数	10	3	5	0	1

問７．授業において難しいと感じた項目

A．講義内容
B．プログラミング言語 C#の理解
C．地図データの作成
D．本グラフ理論ライブラリの設定
E．本グラフ理論ライブラリによる最短経路プログラムの作成
F．特になし

評価項目	A	B	C	D	E	F
人数	4	14	2	5	3	4

図 13　アンケート集計結果　2

5.2 アンケート集計結果とそれに対する考察

本グラフ理論ライブラリを使った演習の評価を行うために，以下の７項目について受講した学生 19 名に対してアンケートを行った．評価は５段階評価で行い，評価項目は問１から問５については５を容易，１を難しいとした．

集計結果を図 12 および図 13 に示す．問１ではグラフエディタの扱いやすさについて調査を行った．アンケート結果から，半数以上がデータ作成はどちらかといえば容易であると回答している．平均値が 4.0 であることから，グラフエディタは演習教材として向いているといえる．

次に，問２，問３，問５は本グラフ理論ライブラリを用いたプログラミング演習についての評価である．いずれも 3.8 から 4.0 という高い結果が得られた．

問４は講義によるダイクストラアルゴリズムの理解度に関するものであり，問５はプログラミングを行うことによるダイクストラアルゴリズムの理解度に関するものである．講義よりプログラミングによる理解度がやや増えている．ただし，１名はプログラミングによる理解度で最低評価をしている．演習を行えば，必ず理解が深まるといえないことは，演習を授業に組み込むときの留意点である．

問７の結果では，多くの学生が C#言語が難しいと答えていることから，プログラミング言語に対するサポートも考慮しなければならないことが分かった．

6. むすび

本論文では，学習者に比較的容易に扱えるグラフ理論ライブラリを構築し，アルゴリズム演習への適用可能性について調べた．

グラフアルゴリズムの演習のためにはグラフデータを容易に作成できる機能が必要であること，アルゴリズムのステップごとの動作確認でできる機能が必要なこと，グラフアルゴリズムを容易にプログラム化できること，グラフの操作，特に頂点，辺の削除操作に関し，学習者を混乱させないことを述べた．

上記の問題を解決するために，グラフエディタを作成し，その機能を説明した．また，深さ優先探索を例にとり，効率的ではないが理解しやすいアルゴリズムから効率的であるが複雑なアルゴリズムまで，本グラフ理論ライブラリを使用すると複数レベルのアルゴリズム演習が問題なく行えることを示した．

本グラフ理論ライブラリに対するアンケートでは，学習に役立つという評価を得ることができた．その一方で，アンケートではグラフ理論ライブラリの土台となるプログラミング言語に対する敷居が高かったという結果も出ている．プログラミングの初心者向けには指導支援ツールがいくつか提案されている [11)~12)]．このような先行研究を参考にしつつ，プログラミング言語の理解も含めたアルゴリズム演習を構築することが今後の課題である．

参 考 文 献

1) F. Harary, グラフ理論, 池田貞雄（訳), 共立出版, 東京, 1978.
2) A.V. Aho, J.E. Hopcroft, J.D. Ullman, データ構造とアルゴリズム, 大野義夫(訳), 培風館, 東京, 1999.
3) 柳本朋子, 中本敦浩, 桝田尚之, “グラフ理論の教育について －中学生を対象として－,” 大阪教育大学紀要第Ⅴ部門, vol.50, no.2, pp201-212, Jan. 2002.
4) J. Siek, L. Lee, A. Lumsdaine, "The Boost Graph Library," http://www.boost.org/doc/libs/1_52_0/libs/graph/doc/index.html, 2000.
5) 浅野哲夫, 小保方幸次, K. Mehlhorn, “LEDA: 複雑なアルゴリズムも簡単にプログラム化できる魔法のツール,” 情報処理, vol.41, no.7, pp.854-861, Jul. 2000.
6) B. Naveh, "JGraphT," http://www.jgrapht.org/, 2003.
7) 松原勇, " グラフ理論を用いた授業設計支援に関する一考察, "信学論 (A), vol.J69-A, no.9, pp.59-64, Sep. 1986.
8) 畠山誠, 佐藤公男, " グラフ理論学習用エディタの開発, "仙台電波工業高専研究紀要,vol.33, pp.20-27, 2003.
9) 松下浩明, "グラフ理論ライブラリの比較－Boost, JGraphT, Arena－, " http://www.di.kagawa-nct.ac.jp/~matusita/Arena/boost-jgrapht-arena/

index.htm, 2011.
10) 丸岡将大, 松下浩明, " グラフ理論ライブラリ Arena の構築と授業への適用, "信学技報, vol.111, no.473, ET2011-119, pp.107-112, Mar. 2012.
11) 中島秀樹, 高橋直久, 細川宣秀, " プログラミング学習のための QA サイクル －受講者の習得度に応じた問題提示メカニズム－, "信学論 (D-I), vol.J88-D-I, no.2, pp.439-450, Feb. 2005.
12) 谷川紘平, ディンドンフォン, 原田史子, 島川博光, " C 言語関数呼出しの記録を用いた演習過程での習得項目の把握, " 信学論 (D), vol.J95-D, no.12, pp.2079-2089, Dec. 2012.