解説

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

浅野　哲夫　北陸先端科学技術大学院大学情報科学研究科
小保方 幸次　北陸先端科学技術大学院大学情報科学研究科
Kurt Mehlhorn　Max Planck Institute für Informatik

はじめに

計算機の進歩に伴って扱う問題のサイズが飛躍的に増大し，以前から使ってきた単純なアルゴリズムに基づくプログラムでは処理しきれない状況が現実のものとなってきています．少しでも計算時間の見積もりをしたことのあるプログラマならアルゴリズムの重要性はよく分かっていると思いますが，さて自分のプログラムに組み込むまでの労力を考えると，なかなか決心がつかないということが多いと思います．問題は何かというと，教科書にアルゴリズムの記述はあっても，そもそもがアルゴリズムの基礎を理解している人向けに記述されていることが一般的なので，たとえば最も基本的なデータ構造であるスタックの実現方法などは常識として片づけられていることです．抽象的なアルゴリズムだけ理解できても，具体的にどんなデータ構造を用いるかが分かっていないとプログラムが書けないのが実情です．実際，どんなデータ構造を用いるかによってプログラムの効率は大きく異なりますが，その辺りの事情を分かりやすく解説している教科書が「アルゴリズム＋データ構造＝プログラム」[2]です．しかし，実際にプログラムを書くにはこれでも十分とはいえないでしょう．たとえば，データ構造として平衡2分探索木を使うとよいことが分かっても，今度はそのデータ構造のプログラムが必要になるからです．

ドイツのザールブリュッケンにあるマックスプランク研究所で開発されたLEDA（正式の名称は，Library for Efficient Data types and Algorithms）は，「アルゴリズム＋LEDA＝プログラム」の実現を目指したソフトウェアライブラリです．このライブラリは約10年前に開発が始まって以来，改良が加えられてきていますが，最初の目標は，アルゴリズムとプログラムの差を縮めることでした．LEDAはC++言語で書かれたライブラリですが，教科書でアルゴリズムが記述されているのとほぼ同じ感じでプログラムが書けるように工夫されています．詳細は後で説明しますが，さまざまな繰り返し構造やクラスが用意されています．次の目標は，豊富なデータ構造を実現して置いておくことです．これによって，プログラマは努力をせずに既存のデータ構造を使うことができます．これ以外にも，誤差なし計算をサポートする仕掛けなどを用意することでプログラマのさまざまな意味での負担を著しく軽減することに成功しています．

本稿では，まず具体的な例題に即してLEDAの威力を説明します．教科書ではアルゴリズムを擬似言語で記述することが一般的ですが，LEDAによるアルゴリズムの記述がいかにテキストレベルのものに近いかをDijkstra（ダイクストラ）の最短経路発見アルゴリズムを例にとって説明します．また，LEDAでは多数のアルゴリズムが関数の形で用意されていますが，特に，グラフと計算幾何に関してどんなアルゴリズムが用意されているかを述べた後，ユーザサポートやインストールの方法についても説

LEDA : A MAGIC TOOL FOR EASY

明します．

計算幾何のアルゴリズム例：いかに良いアルゴリズムを設計するかではなく，いかに良いアルゴリズムを選ぶか

具体的な例をあげて説明することにしましょう．計算幾何学における重要な概念の1つにボロノイ図というものがあります．平面上に多数の点が与えられているとき，平面をどの点に最も近いかによって領域に分割したものです．たとえば，2点だけの場合，垂直2等分線を引いて平面を2分割することに対応しています．多数の点がある場合には，比較的近い点対について垂直2等分線を引くことによって平面を分割していくことになります．ボロノイ図を構成するアルゴリズムはいくつか知られていますが，計算幾何学の分野で最初に提案されたものは分割統治法に基づいたものでした．つまり，与えられた点集合を1本の直線でほぼ同じ点数の部分集合に2分割した後，それぞれの部分集合に対するボロノイ図を再帰的に構成して，それらを中央部で統合するというものです．記述自身は簡単ですが，この記述だけでプログラムが書ける人は天才というべきでしょう．

最低必要なのは，2点に関する垂直2等分線の求め方，2本の線分（直線）の交点の計算方法でありますが，線分の連結関係を表現するデータ構造をどうするかはもっと大きな問題です．計算幾何学では常識になっていることですが，計算誤差によって交差判定で意外に簡単に間違った判定結果が出てしまいます．そのためにプログラムの暴走を招いたりするので，これは深刻な問題です．後でも詳しく説明しますが，LEDAでは誤差なし計算を採用することによってこの問題を解決しています．このように，ボロノイ図を計算するプログラムを組むのは並大抵なことではありません．修士レベルの学生でも1週間で組むのは相当困難だと思いますが，LEDAなら1日でボロノイ図を計算する関数の呼び出し方が習得できてしまいます．具体的には，VORONOI( ) という名前の関数を呼び出すだけでボロノイ図の計算ができてしまいます．**図-1** は，LEDAで用意されているデモ用のプログラムvoronoi_demo.cを実行して作成したものです．このプログラムでは，画面上にマウスで点を指定すると，上部にあるボタンをクリックするだけでボロノイ図だけでなく，ユークリッド最小木や凸包などを求めることができるようになっています．

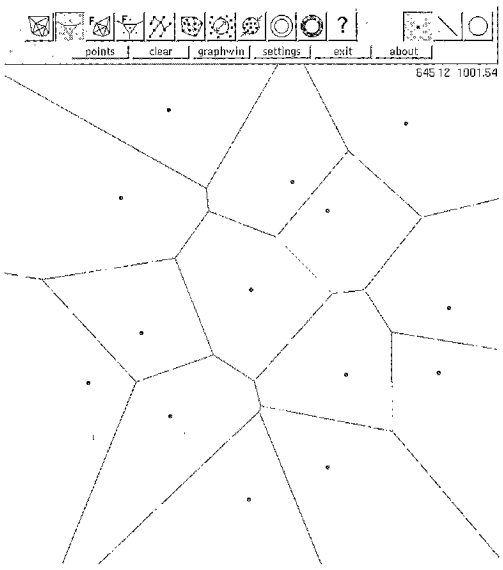

図-1 ボロノイ図（LEDAの画面）

今までアルゴリズムというと，新たなアルゴリズムを設計し，解析するものだという考え方が支配的ではなかったかと思いますが，今では効率のよいアルゴリズムをいかに利用するかの方が重要になってきています．アルゴリズムを理解していなくても，入力と出力の関係が分かっていれば，誰でもアルゴリズムは使えます．そのための道具の1つとしてLEDAを考えることができます．

LEDAだとこんなに簡単

今でも鮮明に記憶していますが，1997年にドイツのダグシュツールで計算幾何学のワークショップに著者のうちの2人（浅野とMehlhorn）が共に参加していました．1週間の予定のワークショップの2日目だったと思いますが，夕方にMehlhornが自らLEDAプロジェクトの進展状況を説明することになっていました．ちょうどその日の午前中にNina Amenta（現テキサス大学）による非常に興味深いトークがありました．問題は，平面上に点集合が与えられたとき，これらの点を適当な順で結んで，自然に見える図形を構成せよ，というものです[1]．人間にとっては比較的簡単な仕事ですが，これまでこの問題が理論計算機科学者の関心を引くことはありませんでした．Ninaたちは，この問題に計算幾何学の道具を持ちこんで，実に華麗なアルゴリズムを考案しました．どんな方法かというと，まず与えられた点集合Sに対してボロノイ図VD（S）を構成します．ここで生じたボロノイ図

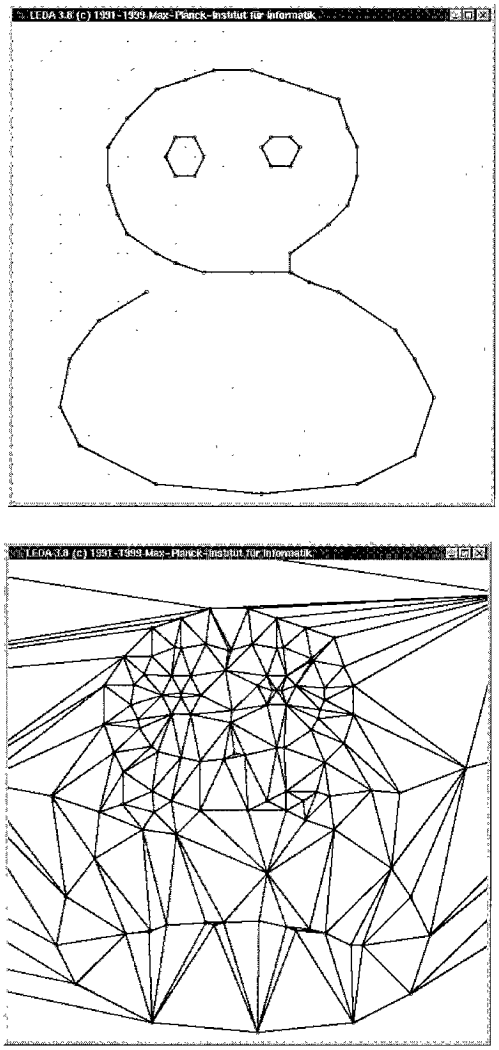

図-2　プログラムCRUSTの出力例（上）と対応するデローネイ三角形分割（下）

の頂点の集合Vと元の点集合Sの和集合$L = S \cup V$に対してデローネイ三角形分割を求めます．点集合Lの三角形分割とは，Lの凸包（Lの点をすべて内に含む最小の凸多角形）の内部をLの点だけを頂点として持つ三角形に分割したものですが，任意の2点を結ぶ線分を1辺とする三角形分割が考えられますので，多数の三角形分割が存在することになります．その中でもデローネイ三角形分割とは，大雑把にいうと，三角形の内角の最小値が最大になるものです．この三角形分割に含まれる辺のうち，元の点集合Sの点どうしを結ぶ辺だけを残して，残りの辺とボロノイ図の頂点を消し去ってできる図形が，このアルゴリズムの出力となります．

上に述べた方法は，ボロノイ図と（その双対として与えられる）デローネイ三角形分割さえ計算できれば，残りの部分はあまり難しくありません．上にも述べたように，LEDAではボロノイ図とデローネイ三角形分割を求める関数が用意されていますので，実に簡単にプログラムが仕上がってしまいます．実際，Mehlhornは午後の間にプログラミングをして，夕方にはデモを披露したのです．具体的なプログラムは以下のとおりです．

```
#include <LEDA/graph.h>
#include <LEDA/map.h>
#include <LEDA/float_kernel.h>
#include <LEDA/geo_alg.h>
#include <LEDA/window.h>
void CRUST (const list<point>& S,
GRAPH<point, int>& G)
{
   list<point> L = S;
   GRAPH<circle, point> VD;
   VORONOI (L, VD);
   // add Voronoi vertices and mark them
   map<point, bool> voronoi_vertex (false);
   node v;
   forall_nodes (v, VD)
   {
      if (VD.outdeg(v) < 2) continue;
      point p = VD[v].center();
      voronoi_vertex[p] = true;
      L.append(p);
   }
   DELAUNAY_TRIANG(L, G);
   forall_nodes(v, G)
      if (voronoi_vertex[G[v]]) G.del_node(v);
}
int main()
{
   list<point> S;
   window W; W.display();
   point p;
   while(W>>p)  //input points in the window
      S.append(p);
   GRAPH<point, int> G;
   CRUST(S, G);
   node v; edge e;
   W.clear();
   forall_nodes(v, G) W.draw_node(G[v]);
   forall_edges(e, G)
      W.draw_segment(G[source(e)], G[target(e)]);
   W.screenshot("crust.ps");
   //output the screen image to a ps file.
   leda_wait(2.0); //wait 2 seconds
   return 0;
}
```

上記のプログラムの実行例を示したのが**図-2**です．同

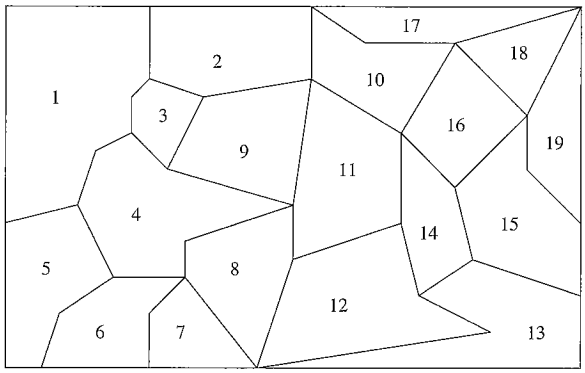

図-3　地図の例

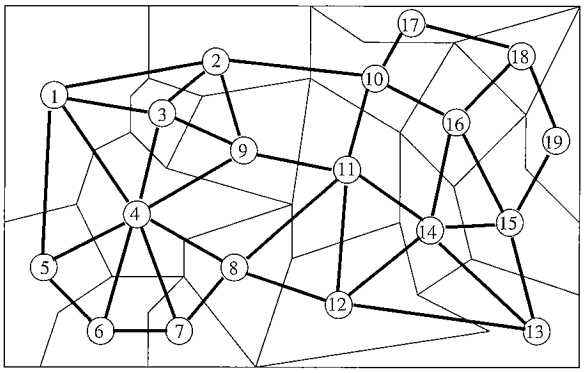

図-4　国の隣接関係を表すグラフ

図下には，実行の途中で構成されたデローネイ三角形分割を示しています．

ツールとしても使えるLEDA

アルゴリズムのことを知らない人でも，平面グラフの4色定理については新聞や本で読んだことがあるのではないでしょうか．これは，地図の上で隣接する国を区別するために，隣接する国には必ず異なる色で塗り分けるとしたとき，実は4色で十分だという定理のことです．この定理は長年未解決でしたが，計算機を駆使した証明によってついに肯定的に解決されて世間を騒がしたことをご記憶の読者も多いことでしょう．したがって，理論的には存在証明があるのですが，実際に白地図を与えて，4色だけで条件を満たすように彩色してほしい，と言われると，なかなか難しいものです（たとえば**図-3**の例）．

計算機に上記の問題を解かせようとすると，まず問題を抽象的に定式化する必要があります．この問題の場合にはグラフを用いるのが最も適切でしょう．問題は隣接する国に異なる色を割り当てないといけないのですが，国の形がどのようなものであっても関係はないわけです．そこで，国を1つの点で表し，2つの国が隣接しているときには対応する2点間を線で結ぶことにします．すると1つの図形ができますが，この図形をグラフといいます（**図-4**参照）．実は，うまく線を引くと，このグラフは必ず平面的に描くことができます．つまり，どの2つの線も端点以外で交差しないように描くことができるのです．逆に，うまく線を引けば，交差しないように描くことができるような隣接関係を表したグラフのことを平面的グラフといいます．

先の彩色問題は，実は平面的グラフの点に彩色を施す問題だということができます．制約は，線で結ばれている2点には必ず違う色を彩色しないといけないというものです．4色定理では4色だけで十分だと保証しているのですが，どのように4色を割り当てるべきかを求めるのは難しい問題です．しかし，5色を使ってもよいということになると，効率のよいアルゴリズムが知られています．ただ，そのアルゴリズムを使って彩色しようとすると，そのアルゴリズムを勉強して，細部に至るまで動作を理解した上でプログラム化する必要があります．グラフ理論を勉強したことがない人にとって，これは大変なことです．

もっと難しい問題があります．上に平面的グラフを定義しましたが，今度は任意にグラフを与えて，それが平面的かどうかを問うのです．分かりやすくいうと，点には1からnまでの番号がついているものとします．グラフを指定するには，どの点とどの点が隣接するかを指定すればいいわけですから，$n \times n$の行列で隣接関係を表すことができます．つまり，i番目の点とj番目の点が隣接するなら，行列の(i, j)および(j, i)要素を1とし，そうでなければ0とするように決めればいいわけです．このような行列が与えられて，その行列が表すグラフが平面的かどうかを判定しなければならないわけです．人間なら，紙の上に適当に点を並べて，適当に番号をつけ，行列の値に従って結ぶべき2点間に線を引いていきます．最後まで交差なく線が引けたら，平面的だということができます．でも，少し隣接関係が複雑になると，本当は交差なく引けるのに，その方法を見つけるのが非常に難しくなります．**図-5**の例を見てください．適当に点を並べて単純に直線で結んだために，交差が生じていますが，うまく点の位置を変えると交差をなくすことができます．あなたはできますか？

上に述べた問題は，グラフの平面性判定問題といって，グラフ理論のテキストなら必ず載っている重要な問題です．グラフ理論の方では，グラフが平面的であるための必要十分条件が求められていますが，その条件を確かめるという形でプログラムを書くことはかなり困難です．そこで，アルゴリズムの分野では，いかにして効率よく平面性を判定するかが研究されてきました．幸いなこと

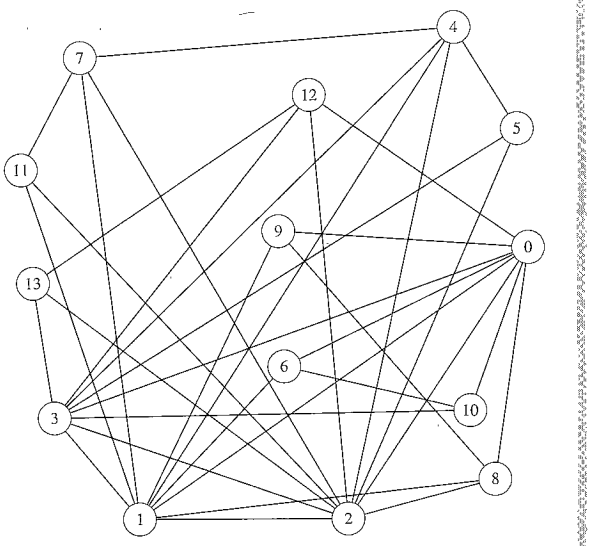

図-5　このグラフは平面的か？

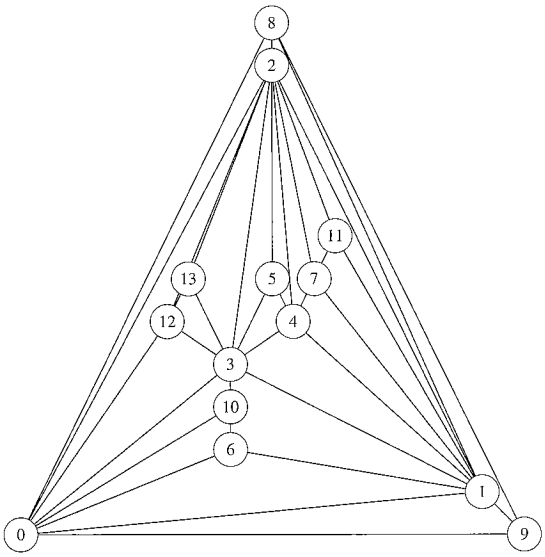

図-6　図-5のグラフの平面描画

に，点の個数に比例する時間で平面性を判定するアルゴリズムがHopcroftとTarjanによって1974年に求められていますので，アルゴリズム理論としては解決がついているのですが，実際にそのアルゴリズムをプログラムの形に実現するのは，よほどグラフとアルゴリズムの知識を持ち合わせていないと難しいでしょう．事実，アルゴリズムの分野では平面性判定に関してはいくつかの方法が提案されていますが，やはりプログラムを組むとなると，相当の覚悟が必要になります．

LEDAにはグラフの平面性を判定するプログラムが組み込まれているだけではなく，平面的であるグラフを実際に平面的に，すなわち辺が交差しないように描画する関数が組み込まれています．たとえば，デモ用のプログラムgw_plan_demoを実行しますと，グラフ作成用のウィンドウが開かれて，そこでグラフを作成・編集することができます．このグラフを平面描画するメニューを選んで実行すると**図-6**に示す結果が得られます．このように，LEDAはプログラムを作成するツールであると同時に，論文を書く際の作図の道具としても使えるのです．出力結果をpsファイルとして出力することも簡単です．実際，図-6もその機能を使ったものです．

では，**図-7**に示したようなグラフはどうでしょう．平面的に描画するメニューを選ぶと，今度はグラフは平面的ではないという出力があり，さらに証明をするかどうかが尋ねられます．ここで証明する方を選択すると，**図-8**のような結果が示され，このグラフにKuratowskiグラフ（この場合は，$K_{3,3}$）が部分グラフとして含まれていることが陽に示されます．

ここにもLEDAの基本思想が現れています．求められているのは，グラフの平面性判定であったとしても，出力が平面的かどうかだけであれば，出力の正しさを証明することは非常に難しいわけですが，問題を少し変更して，「与えられたグラフが平面的かどうかを判定し，もし平面的なら実際に平面上に描画し，そうでなければKuratowskiグラフが部分グラフとして含まれることを示せ」とすると，今度はどちらの結果になっても出力の正しさを検証するのは簡単です．このように，出力の正しさを検証できる形で問題を設定することは非常に重要です．

LEDAをインストールしよう

LEDAを実際に使うためには，C++言語のコンパイラが必要ですが，それに加えてLEDAをインストールする必要があります．インストールはいたって簡単です．まず，インターネットを通してダウンロードします．ダウンロードは

http://www.mpi-sb.mpg.de/LEDA/download/

からできます．ダウンロードするにはユーザ登録が必要ですが，上記ホームページから登録できます．ホームページにはUnix系のSparc-Solaris，Mips-Irix，i386-linuxなどとWindows系のVisual C++，Borland C++など用のオブジェクトパッケージが用意されています．使用する環境がこれらと合えば，パッケージをダウンロードした後で展開し，必要なファイルを/usr/local/libや/usr/local/includeなど，環境に合わせてコピーすればよいわけです．環境が用意されたパッケージと合わない場合のためにソースコードパッケージが用意されています．ソースコードパッケージを展開した後でコンパイル（make）すれば，オブジェクトパッケージと同じものが作られます．

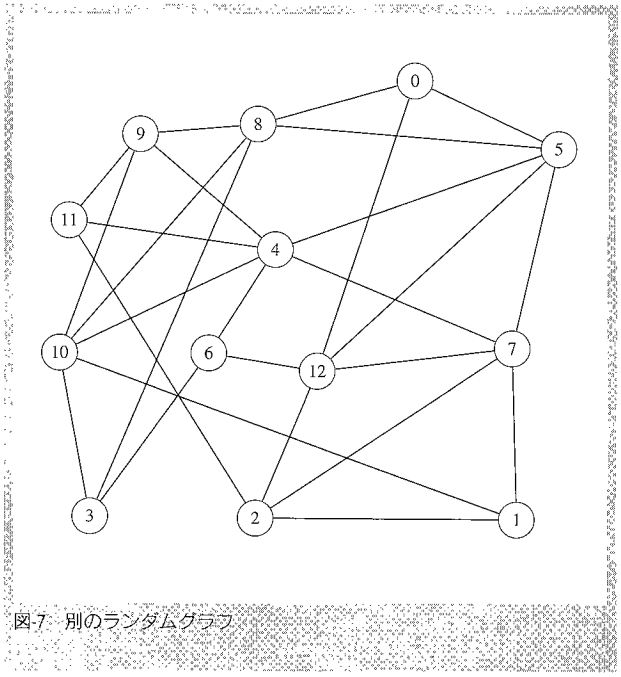

図-7　別のランダムグラフ

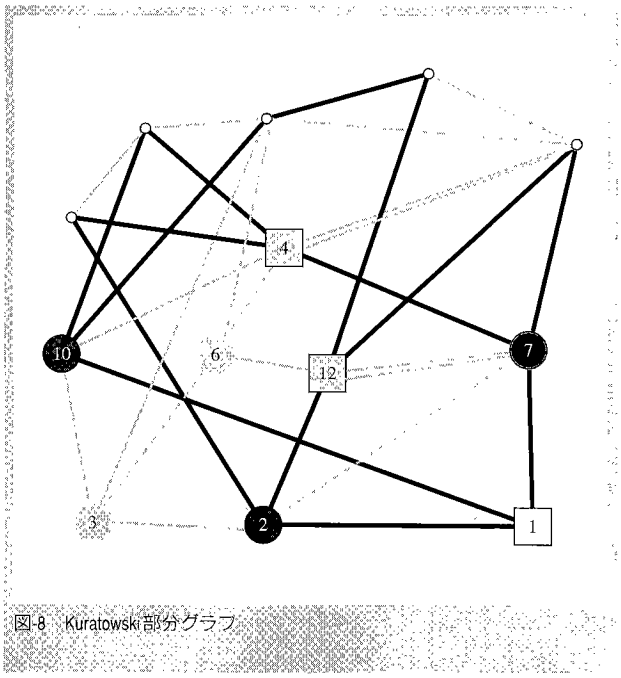

図-8　Kuratowski部分グラフ

LEDAのサポート体制

LEDAは現在世界中で10年以上にわたって使われており，バグはかなりなくなってきていますが，商用に使おうとすると信頼できるユーザサポートは欠かせません．従来，LEDAはアカデミックな機関で管理されてきましたが，ユーザサポートを本格的に行うためには限界があるということで，現在は会社組織になっています．LEDAはアカデミック機関のユーザに対しては無償ですが，ソースレベルでのユーザサポートを望む商用ユーザは，サポートの質によっていくつかのレベルで契約できることになっています．

ユーザサポートが完備してからユーザ数は飛躍的に増大し，現在では50カ国の1,500の機関がユーザ登録をしています．特筆すべきことは，そのうちで計算機科学関連のユーザは半数以下であり，さらに計算機科学のユーザの中でもアルゴリズムを専門としているユーザ数は1/5に満たないということです．つまり，ほとんどのユーザはアルゴリズムを考案する側ではなく，単にアルゴリズムを利用するだけの側にいるのです．これは，LEDAがアルゴリズム専門家のために作られたライブラリではなく，非専門家を対象として作られたものであることを如実に物語っています．

LEDAの商用ユーザ

上記のユーザサポートに助けられて，LEDAは産業界でも着実にユーザ数を増やしてきています．主だったところをあげると：MCI (USA)，Comptel (Finnland)，France Telecom，ATR (Japan)，Siemens (Germany)，Sun Microsystems (USA)，Daimler Benz (Germany)，Ford (USA)，Silicon Graphics (USA)，DEC (USA)，Minolta (USA) 等々です．

教科書のアルゴリズムからプログラムへ

上に述べたように，LEDAには実にさまざまなアルゴリズムが実装されて用意されていますが，単に多数のアルゴリズムを寄せ集めただけではなく，さらに一歩踏み込んで，できるだけ少ない労力でアルゴリズムをプログラム化するための環境も与えています．ここでは，グラフ上で最短経路を求めるDijkstraのアルゴリズムがLEDAではどれほど簡潔に記述できるかを示しましょう．まず，下に示したのはアルゴリズムの教科書で見かける記述です．

Dijkstraの最短経路アルゴリズム

入力：有向グラフ $G = (V, E)$, s in $\in$, 始点,
　cost: $E \rightarrow N$, 辺のコスト
begin
　dist(s) = 0;
　dist(v) = ∞; for $v \neq s$;
　すべての頂点に「未到達」のラベルをつける;
　while「未到達」の頂点が存在する
　do uを「未到達」の頂点の中でdistの値が
　　最小のものとする;
　　uに「到達」のラベルをつける；
　　for uから出るすべての辺 $e = (u, v)$

```
    do dist(v) = min(dist(v), dist(u) + cost(e));
    od
  od
end
```

Dijkstraのアルゴリズム自身は非常に基礎的なものなので詳しい説明は省略しますが，上の記述には曖昧な記述が多くあります．たとえば，有向グラフはどのようなデータ構造で管理すべきかについては何も指定していません．頂点や辺についても同様です．最も難関であるのは，「未到達」の頂点の中でdistの値が最小のものを求めるところですが，アルゴリズムの知識を持った人なら，プライオリティキューが必要になることは分かるのですが，実際のデータ構造を考えようとすると結構面倒です．LEDAでは，頂点と辺の配列とグラフを定めるデータタイプと，節点に関するプライオリティキューのデータ構造をプログラムの先頭で宣言するだけです．紙数の都合上，詳細な説明はできませんが，下に示したLEDAプログラムは上記のアルゴリズムの記述にかなり近いことが理解してもらえるでしょう．

```
void DIJKSTRA(const graph &G, node s,
    const edge_array<double>& cost,
    node_array<double>& dist)
{
  node_pq <double> PQ(G);
  node v; edge e;
  forall_nodes(v, G)
  { if (v == s) dist[v] = 0;
      else dist[v] = MAXDOUBLE;
    PQ.insert(v, dist[v]);
  }
  while(!PQ.empty())
  {
    node u = PQ.del_min();
    forall_out_edges(e, u)
    {
      v = target(e);
      double c = dist[u] + cost[e];
      if (c < dist[v])
      { PQ.decrease_p(v, c); dist[v] = c; }
    }
  }
}
```

LEDAの構成

LEDAにはさまざまな基本的なデータタイプが用意されています．スタックやキューはもちろんのこと，たとえば連結リストもlist<int> L; のように宣言することで，整数値を持つ単方向連結リストを利用することができます．このほかにも各種のプライオリティキューや，有向および無向のグラフも簡単に扱えます．さらに，点，直線，多角形などの幾何対象物の扱いも非常に容易です．計算幾何では最近ヨーロッパの大学が中心となってCGALと呼ばれるライブラリが開発されていますが，それとの親和性も図られていますので，共用可能です．

LEDAにおける計算誤差対策

浮動小数点数を用いた計算では計算誤差対策が非常に重要です．特に，幾何データの処理においては，計算誤差が位相情報と矛盾するためにプログラムが暴走することが深刻な問題になります．LEDAでは，誤差なしの計算を効率よくサポートするためのデータタイプを用意しています．具体的には，C言語でサポートされているshort, int, long, float, doubleのほかに，任意桁数の整数を表すinteger，任意精度の浮動小数点数を表すbigfloat，有理数を表すrationalなどが用意されています．bigfloatでは，bigfloat::set_precision(2*m);のようにして仮数部の長さを指定したり，丸めの方式さえも指定できるようになっています．さらにLEDAに特徴的なデータタイプにrealがあります．これは，

```
real x = (sqrt(17) - 2) * (sqrt(17) + 2);
```

のように使います．このように平方根や立方根などを用いて定義された値を持てるのがreal変数です．数学的には上式の値は$17-2^2=13$となりますが，LEDAのreal変数も同じ値を持つことになります．これはかなりすごいことです．これ以外にもさまざまなデータタイプが用意されていますが，詳しくは最近刊行されたLEDA Book[3]かLEDA Manual[4]を参照してください．基本的には，浮動小数点数の代わりに有理数の多倍長表現を用いて，誤差が生じないように計算するというものですが，単純な仕掛けでは実行時間が爆発してしまいます．そのために，浮動小数点数での計算結果をうまく利用しています．

グラフに関するアルゴリズム

LEDAにはグラフに関して実にさまざまなアルゴリズムが関数の形で用意されていますが，主だったものを列挙すると以下のようになります．

1. トポロジカルソート：

2. 深さ優先探索：訪れた節点に印だけをつける版と，番号づけをする版があります．

3. 幅優先探索：訪問したかどうかを示すラベルづけするだけの版と，始点からの距離をつける版とがあります．

4. 強連結成分：与えられた有向グラフのすべての節点について，その節点が属する強連結成分の番号を求めます．

5. 2連結成分：与えられたグラフを無向グラフと見なして，各節点が属する2連結成分の番号を求めます．

6. 推移的閉包：与えられた有向グラフの推移的閉包を計算して返します．

7. 最短経路：

7.1 単一始点最短経路アルゴリズム：Dijkstraのアルゴリズムと Bellman-Fordのアルゴリズムを含みます．

7.2 全点対最短経路アルゴリズム：すべての節点対について最短経路を求めます．

8. 最大フロー：preflow-pushに基づく Goldberg-Tarjanの最大フロー計算アルゴリズム．さまざまなヒューリスティクスを組み込んだ版も用意されています．

9. 最小コストフロー：capacity-scalingと最短経路アルゴリズムに基づいて最小コストフローを求めるアルゴリズム．これについてもいくつかの変形版が用意されています．

10. 2部グラフの最大マッチング：与えられた2部グラフに対して要素数最大のマッチングを求めます．

11. 2部グラフの最大重みマッチング：

12. 一般のグラフに対する最大マッチング：

13. 最小全域木：

14. グラフの平面性判定：線形時間のアルゴリズムのほかに，Kuratowskiグラフを部分グラフとして含むかどうかを判定するアルゴリズムもあります．

15. 平面グラフの三角形分割：

16. 平面グラフを5色で彩色するアルゴリズム：

17. グラフ描画アルゴリズム：さまざまな条件の下で平面グラフを直線などを用いて平面上に描画するアルゴリズム多数．

計算幾何学関係のアルゴリズム

グラフだけではなく，計算幾何学関係のアルゴリズムも多数用意されています．下に名前だけを列挙しておきます．

1. 凸包を求めるアルゴリズム

2. 点集合に対する三角形分割（デローネイ三角形分割を含む）

3. 制約つきの三角形分割

4. 多角形の三角形分割

5. ユークリッド最小木

6. ボロノイ図の計算

7. 最遠点ボロノイ図

8. 最大空円

9. 最小包含円

10. 真円度の計算：半径の差が最小である同心円の間にすべての点が入るような2つの円を求めます．

11. 与えられた点を自然に結ぶ（CRUST）

12. 線分交差列挙アルゴリズム

13. 最近点対

14. 最小包含長方形

15. 単純多角形かどうかの判定

日本で公開されているアルゴリズム

科学研究費特定研究「アルゴリズム工学」のグループでは，アルゴリズムの実用化研究を推進していますが，一般的普及のための重要な道具としてLEDAを位置づけています．具体的には，さまざまなアルゴリズムを利用可能な形で一般に公開することも重要な使命の1つですが，一部のソフトはLEDAで記述することが計画されています．現在はまだ整備中ですが，ぜひ下記サイトを訪問してください．

http://www-or.amp.i.kyoto-u.ac.jp/algo-eng/db/index.html

参考文献

1) Amenta, N., Bern, M. and Eppstein, D.: The Crust and the Beta-skeleton: Combinatorial Curve Reconstruction, Graphics Models and Image Processing, pp.125-135 (1998).
2) N. ヴィルト著，片山卓也訳: アルゴリズム＋データ構造＝プログラム，日本コンピュータ協会 (1979).
3) Mehlhorn, K. and Näher, S.: LEDA: A Platform for Combinatorial and Geometric Computing, Cmbridge Univ. Press (1999).
4) Mehlhorn, K. et al.: The LEDA User Manual Version 4.0, Max Planck Institute für Informatik, Germenay (1999).

(平成12年2月9日受付)

PROGRAMMING OF COMPLEX ALGORITHMS