LEDA＋アルゴリズム＝プログラム

浅野　哲夫，小保方　幸次，Kurt Mehlhorn

1. はじめに

アルゴリズムの重要性はよく分かっていても，自分のプログラムに組み込むまでの労力を考えると，素朴な方法で我慢してしまうことが多い．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付けられている．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ある．こ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] である．しかし実際にはこれでも十分ではない．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ある．

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[1] 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ある．まず，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ある．LEDA は $C++$ 言語で書かれたライブラリであるが，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る．詳細は後で述べるが，様々な繰り返し構造やクラスが用意されている．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，プログラマーが既存のデータ構造を使いやすくしていることである．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ユーザの様々な意味での負担を著しく軽減することに成功している．

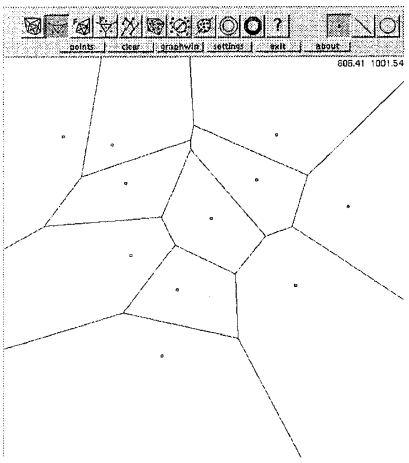

図 1: ボロノイ図（LEDA の画面）

具体的な例をあげて説明しよう．都市工学では都市の商圏に相当するボロノイ図（図１参照）をサブルーティンとしてよく使うが，このボロノイ図を計算するプログラムを組もうとすると，修士レベルの学生でも１週間で組むのは相当困難であるが，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う．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る．図１は，デモ用のプログラム voronoi_demo.c を実行して作成したものである．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一般的だと思われるが，今では効率の良いアルゴリズムを如何に利用するかの方が重要になってきている．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る．そのための道具の一つとして LEDA を考えることができる．そういう意味で，ここではアルゴリズムを利用するだけの人達を対象として解説をする．

あさの　てつお，　おぼかた　こうじ　北陸先端科学技術大学院大学情報科学研究科
〒 923-1292 石川県能美郡辰口町旭台 1-1
クルト　メルホーン，Max Planck Institute for Information, Saarbrücken, Germany

[1] 正式の名称は，Library for Efficient Data types and Algorithms.

2. LEDAの威力

今でも鮮明に記憶しているが，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内の2人（浅野とMehlhorn）が共に参加していた．1週間の予定のワークショップの2日目だったと思うが，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った．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，というものである[1]．人間にとっては比較的簡単な仕事であるが，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なかった．Nina達は，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た．どんな方法かと言うと，まず与えられた点集合Sに対してボロノイ図VD(S)を構成する．ここで生じたボロノイ図の頂点の集合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る．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もつ三角形に分割したものであるが，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るので，多数の三角形分割が存在する．その中でもデローネイ三角形分割とは，大雑把に言うと，三角形の内角の最小値が最大になるものである．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る．

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，残りの部分は余り難しくない．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るので，実に簡単にプログラムが仕上がってしまう．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ある．具体的なプログラムは以下の通りである．

```
#include <LEDA/graph.h>
#include <LEDA/graphwin.h>
#include <LEDA/map.h>
#include <LEDA/float_kernel.h>
#include <LEDA/geo_alg.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);
      // delete voronoi vertices and edges
         incident to them
}
int main()
{  list<point> S;
   window W("Connecting Points Naturally");
   W.init(0,1000,0,25);
   W.display();
   point p;
   while(W >> p) {  //input points in the window
      S.append(p);
      W.draw_circle(p, 2, green);
   }
   GRAPH<point, int> G;
   CRUST(S, G);
   edge e;
   forall_edges(e, G){
      point p1 = G[source(e)];
      point p2 = G[target(e)];
      W.draw_segment(p1.to_point(),
         p2.to_point(),blue);
   }
   W.screenshot("crust.ps");
   //output the screen image to a ps file.
   leda_wait(2.0);  //wait 2 seconds
```

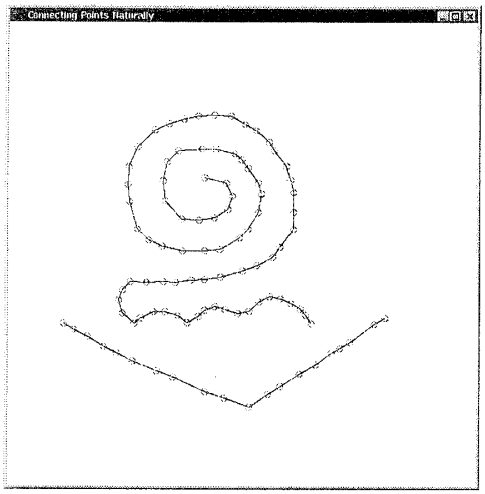

図 2: プログラム CRUST の出力例

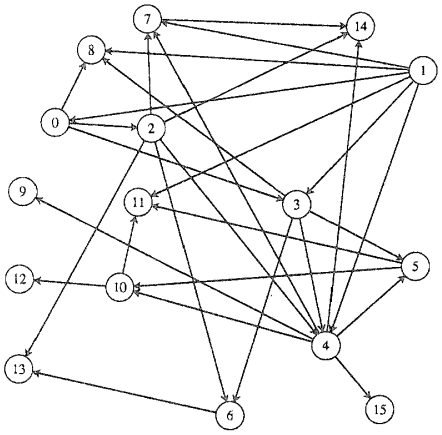

図 3: ランダムグラフの例

```
    return 0;
}
```

上記のプログラムの実行例を示したのが図 2 である.

3. 道具としての LEDA

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. たとえば, グラフに関する作図で Tgif などのツールを使ったとき, グラフが完成したところで, 頂点の位置を少しだけずらそうとすると, 接続している辺についても変更を余儀なくされる. しかし, LEDA ではデモ用のプログラム gw_plan_demo を実行すると, グラフ作成用のウィンドーが開かれて, そこでグラフを作成·編集することができる. メインメニューで Graph を選ん後, そのサブメニューにある random graph を選んで頂点数と辺数を指定すれば, ランダムに生成されたグラフが画面上に表示される. マウスを用いて頂点の位置などを調節することができるが, さらに結果を ps ファイルでセーブすることができる. 図 3 に一例を示す. 図 4 は, さらにこのグラフを直線で平面描画したものである. では, 図 5 に示したようなグラフはどうだろう. これは, メインメニュー Layout のサブメニューにある Simple Layout の中の circular layout を選択して, グラフの頂点を円周上に配置して表現したものである. これを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る. ここで証明する

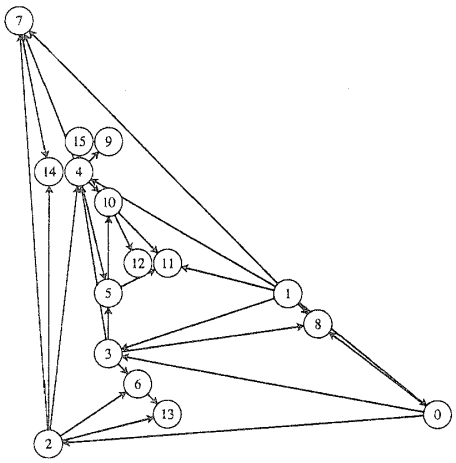

図 4: 図 3 のグラフの平面描画

方を選択すると, 下図 6 のような結果が示され, このグラフに Kuratowski グラフが部分グラフとして含まれていることが陽に示される. ここでも LEDA の基本思想がある. 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は一目瞭然である. 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ある.

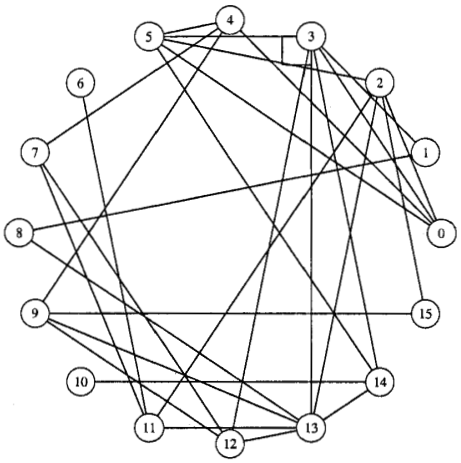

図 5: 別のランダムグラフ

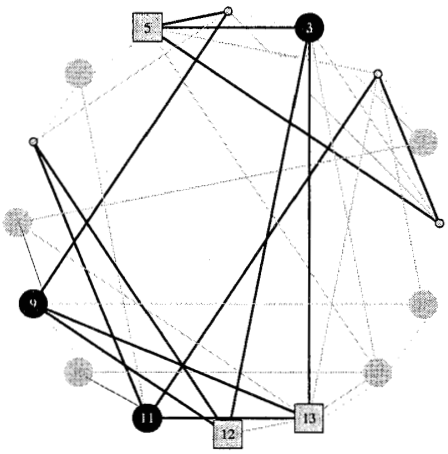

図 6: Kuratowski 部分グラフ

4. LEDA のインストール方法

LEDA を実際に使うためには、C++言語のコンパイラが必要であるが、それに加えてLEDA をインストールする必要がある。インストールは至って簡単である。まず、インターネットを通してダウンロードする。ダウンロードは http://www.mpi-sb.mpg.de/LEDA/download/からできる。ダウンロードするにはユーザ登録が必要だが、上記ホームページから登録できる。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る。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。環境が用意されたパッケージとあわない場合にはソースコードパッケージが用意されている。ソースコードパッケージを展開後コンパイル (make) すればオブジェクトパッケージと同じものが作られる。これをオブジェクトパッケージと同様に必要なファイルをコピーすればよい。

5. LEDA のユーザサポート

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るが，商用に使おうとするとユーザサポートは欠かせない．従来，LEDA はアカデミックな機関で管理されてきたが，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る．LEDA はアカデミック機関のユーザに対しては無償であるが，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幾つかのレベルで契約できることになっている．

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 か国の 1500 の機関がユーザ登録をしている．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５分の１に満たないということである．つまり，殆どのユーザはアルゴリズムを考案する側ではなく，単にアルゴリズム利用するだけの側にいるのである．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る．

6. 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る．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) 等々である．

7. 教科書のアルゴリズムをそのまま実行

上に述べたように，LEDA には実に様々なアルゴリズムが実装されて用意されているが，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る．ここでは，グラフ上で最短経路を求めるダイクストラのアルゴリズムがLEDAではどれほど簡潔に記述できるかを示そう．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ある．

ダイクストラの最短経路アルゴリズム
入力：有向グラフ $G=(V,E)$, $s \in V$, 始点,
　cost: $E \to N$, 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

ダイクストラのアルゴリズム自身は非常に基礎的なものであるので詳しい説明は省略するが，上の記述では曖昧な記述が多い．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ない．頂点や辺についても同様である．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所であるが，アルゴリズムの知識を持った人なら，プライオリティーキューが必要になることは分かるが，実際のデータ構造を考えようとすると結構面倒である．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ー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よい．紙数の都合上，詳細な説明はできないが，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あろう．

```
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; }
      }
    }
}
```

8. グラフアルゴリズム

LEDAにはグラフに関して実に様々なアルゴリズムが用意されているが，主だったものを列挙すると以下のようになる．

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，番号付けをする版がある．
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る．
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る．
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る．
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む
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る．

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．様々なヒューリスティックスを組み込んだ版も用意されている．

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．これについても幾つかの変形版が用意されている．

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る．

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他に，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る．

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: 様々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．

9. 幾何ルゴリズム

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る．下に名前だけを列挙しておく．

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つの円を求める．

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:

10. 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るが，一般的普及のための重要な道具としてLEDAを位置づけている．具体的には，様々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一つであるが，一部のソフトはLEDAで記述することが計画されている．現在はまだ整備中であるが，是非下記サイトを訪問されたい．

http://zagato.kuamp.kyoto-u.ac.jp
/labs/or/tokutei98/database/index.html

11. LEDAの数値誤差対策

浮動小数点数を用いた計算では数値誤差対策が非常に重要である．特に，幾何データの処理においては，数値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である．LEDAでは，誤差なしの計算を効率良くサポートするためのデータタイプを用意している．具体的には，int, float, doubleの他に，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る．これ以外にも様々なデータタイプが用意されているが，詳しくは最近刊行されたLEDA Book[3]かLEDA Manual[4]を参照されたい．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あるが，単純な仕掛けでは実行時間が爆発してしまう．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る．

参考文献

[1] N. Amenta, M. Bern, and D. Eppstein: "The crust and the β-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] K. Mehlhorn and S. Näher: "LEDA: A platform for combinatorial and geometric computing," Cmbridge Univ. Press, 1999.

[4] K. Mehlhorn et al.: "The LEDA User Manual Version 4.0," Max Planck Institute für Informatik, Germenay, 1999.