ネットワーク問題プログラミングを支援するソフトウェア・ツール

加地　郁夫

0. はじめに

ネットワーク問題に関するプログラムを書くとき，あまり本質的ではない事柄にかかわらなくてもよいように，プログラミングの援助をするツールがあると便利である．たとえば，ネットワークの基本要素である頂点や辺にはいろいろな属性とかメソッドを必要に応じて付与するのが普通である．しかし，これにともなって複雑なデータ構造の詳細に立ち入ってプログラムを書き直すとか，また要素を集合にまとめるのに使うリスト処理でリンク操作の詳細に手を入れたりすることはできるだけ避けたい．また，ある部分をネットワークに対して計算処理を行なうとき，その部分グラフに固有な内部処理番号を使って処理を終えた後，再び元の頂点番号に戻るといった自動化機能も欲しい．さらに，要素のデータ・フィールドへのアクセス，要素集合の走査方式などの基本操作は明快で単純なものにしたい．これらの要望が満たされることを目標に立て，オブジェクト指向言語がもつ継承，クラスの階層構成，および生成的クラスの概念を使ったプログラミング支援ツールを試作してみた．

使用言語はC++(Ver 2.0)で，ツールはヘッダー・ファイル gtool. h とライブラリ・ファイル gtool. obj からなり，小型の普及型パソコンでも充分使用可能なプログラム・サイズである．

1. ツールの目標と設計方針

ネットワーク問題が扱う分野は，最短経路問題，最大流量問題，最小木問題など理論的に興味のある問題に加えて，応用の分野でもＰＥＲＴ－ＣＰＭ，信頼性グラフ，順序づけとラインバランシング，施設問題，輸送，交通，組織，作業計画など広範かつ多様である．しかし，問題分野での多様性にもかかわらず，プログラミングの立場から見れば，使用されるデータ構造や制御構造のパタンに共通な側面が多く見いだされる．

まず，ネットワーク問題の基本的オブジェクトである頂点や辺には多くの属性値が付与されており，その種類も多様である．また問題によっては，属性の異なるオブジェクトを同時に取り扱うこともあり，さらにプログラム開発過程で属性の変更が必要になることもある．このような多属性の導入が，プログラムを多様で個別化されたものにし，ネットワーク問題プログラミングを複雑にする原因の１つになっている．

一方，多属性にかかわる詳細を捨象し抽象化した立場では，基本オブジェクトに対して集合を定義しその上で各種の操作をする，複合構造としてのグラフを生成する，頂点と辺の接続関係を作りだすなどすべてのネットワーク問題にとって共通の概念である．これらのよく使われる処理を共通の枠組みとして用意しておくことはプログラミングの効率化のために望ましい．

頂点集合に属する頂点の総数をnとするとき，１からnまでの連続的な番号を各頂点に一意に付与し，これを各頂点のインデックスまたは内部名という．頂点と辺の接続関係を用いる算法では，インデックスの使用が算法表現や計算効率の面から有利である．他方，多種類の頂点集合，辺集合およびグラフについての集合論的処理が主体となる場合には，頂点について一意な認識番号（外部名）を使用する方が都合がよい．頂点と辺の接続関係を使う環境に入るとき，処理番号を認識番号からインデックスに変換し，この環境から出るとき自動的に元の認識番号に戻る機能をツール化しておくと便利である．

以上の考察で示唆された問題点ま要望をオブジェクト指向プログラミングのもとで解決し，実用的なプログラミング支援ツールにまとめることが目標である．そのための設計方針は下記のとおりである．

まず，基本要素である頂点や辺が多属性をもつことにより生ずる問題は継承によって解決できる．属性に関する捨象により共通な特性のみを残すことによって抽象化した頂点や辺のクラスを基本クラスとしてツール内に定義し，これらからプログラム上必要となる属性を追加し

かじ　いくお　北海道工業大学
〒006 札幌市手稲区手稲前田419の2

た派生クラスを利用者プログラム内に定義する．

次に多属性の頂点や辺について集合を定義し，その上で各種の操作をする，複合構造としてのグラフを定義する，頂点と辺の接続関係を作りだすなどの操作を共通の枠組みとしてもつためには，頂点や辺を指すポインタをパラメターとする生成的クラスの概念を使う．

最後に多属性の頂点と辺について接続関係を構成し，その関係を利用する環境下でインデックス値（内部名）による操作を許し，環境から出るときには元の認識番号に自動的に戻す機能は，環境そのものをインスタンスとする生成的クラスとして実現する．

2. ツールのプログラム構造

ネットワークの基本構成要素である頂点と辺のクラスは，抽象，基底，応用の3つのレベルに分けて階層的に構成される．はじめの2つのレベルのクラスはツール内に実現されており，最後の応用レベルのクラスは前二者からの派生クラスとして利用者が自分のプログラム内に実現しなければならない．

抽象レベルでは要素を単に集合を形成できる対象としてとらえている．このレベルで取り込まなければならない必須のメソッドは2つの要素の同等性の判定である．

基底レベルでのクラスは頂点と辺に関する2つのクラスからなり，抽象レベルのクラスから導出される．それぞれ頂点と辺を特徴づける必須データ項目と，基本的メソッドをメンバ関数としてもつ．これらは，すべてのネットワークに共通な特性のみを残して抽象化された頂点や辺をインスタンスにもつクラスである．

応用レベルのクラスは，プログラムが要求するすべての属性とメソッドをもつクラスとして，基底レベルのクラスから利用者プログラム内で派生する．

1) ツール内に存在する抽象，基底レベルの頂点，辺のクラスの継承関係を示すと，

```
class object;
class baseNode : public object;
class baseEdge : public object;
```

ここで object は抽象レベル，baseNode，baseEdge は基底レベルにある頂点および辺クラスの名前である．また，これらクラスのオブジェクトを指すポインタ変数はツール内システム変数として次のように定義されている．

```
typedef object* ent;
typedef baseNode* entV;
typedef baseEdge* entE;
```

また名前 idNode と idxNode は頂点の認識番号とインデックス値の型名で，これらもツール内で次のように定義している．

```
#define idxNode unsigned int
#define idNode unsigned int
```

2) 利用者は baseNode, baseEdge を基本クラスとして，必要な属性値とメソッドをもつ応用レベルのクラスを利用者プログラム内に派生することができる．これら導出クラスの名前を，それぞれ仮の名前 derivedBaseNode, derivedBaseEdge で表わす．また，これらクラスのインスタンスを指すポインタ変数も，それぞれ仮の名前 type1 および type2 で表わす．実際のプログラミングではこれらの導出クラスと対応するポインタ変数は要求された個数だけ，利用者プログラム内に実名で定義しなければならない．そして仮の名前をこの実名で置き換える．

```
class derivedBaseNode : public baseNode;
class derivedBaseEdge : public baseEdge;
deftype derivedBaseNode* type1;
deftype derivedbaseEdge* type2;
```

仮の名前 type 1 および type 2 はツール内で定義されるすべての生成的クラスの仮パラメターとして使われている．これらに実パラメターを割り当てることにより生成的クラスの実例が具体的に定まる．

3) 利用者が自分のプログラム内に定義した derived BaseNode 型の頂点，derivedBaseEdge 型の辺について，集合を定義し，その上で各種の操作をする，複合構造としてのグラフを生成する，頂点と辺の接続関係を作り出すなどの操作を共通の枠組みとしてもつために，type1 または type2 をパラメターとする，頂点および辺の集合型，グラフ型，接続環境型を与える生成的クラス

```
Vlist(type1), Elist(type2), graph(type1,type2),
adj(type1,type2)
```

がツール内に存在する．これらのパラメターを与えられた頂点または辺を指すポインタ変数名で置き換えることによって，生成的クラスの実例が具体的に定まる．これらの生成的クラスはツールプログラムの中で，より抽象度の高い基本クラスから階層的派生関係によって作りだされている．その様子を示すと下記のようになる．

```
class slink;
class slist;
```

```
class alist : slist;
class vlist : public alist;
class elist : public alist;
class Vlist(type1) : public vlist;
class Elist(type2) : public elist;
class graph(type1, type2) : public Vlist(type1),
      public Elist(type2);
```

クラス slist は抽象レベルの要素 object に対する環状リストと，その上での基本操作を与える．これにツール形成上必要とされる初等的メソッドを追加して派生したクラスが alist である．次のレベルの派生クラス vlist, elist では，それぞれ基底レベルの baseNode 型，baseEdge 型要素がリストの要素となる．また，これらを基本クラスとして導出した生成的クラスが Vlist (type1), Elist(type2) であり，応用レベルの derived BaseNode 型，derivedBaseEdge 型要素をリスト要素としてもつ．これらの生成的クラスから二重継承によって導出したのが生成的クラス graph (type1, type2) であり，応用レベルの頂点，辺に対するグラフを定義する．また，与えられた頂点集合に対して，頂点の認識番号（外部名）とインデックス値（内部名）の対応表とその上での操作を与えるクラスとして

```
class mutator;class Mu(type1):public mutator;
```

がある．前者は基底レベルの頂点に，後者は応用レベルの頂点に対応する生成的クラスである．また，頂点と辺の接続構造と環境に関連するクラスは次のような階層構造によって派生されている．

```
class adjacent;
class Adj(type2) : public adjacent;
class adj(type1, type2) : public Adj(type2),
      public Mu(type1);
```

クラス adjacent は基底レベルの baseEdge 型辺についての接続関係，また Adj (type2) は応用レベルの derivedBaseEdge 型辺についての接続関係を与えるクラスである．2つのパラメターをもつ生成的クラス adj (type1, type2) は応用レベルの頂点と辺について接続関係とインデックスによる計算環境を与える生成的クラスである．

3. ツールの使用法と実例

ツール内にクラス・ライブラリとして用意されている生成的クラス Vlist (type1), Elist (type2), graph (type1, type2), adj (type1, type2) の仮パラメターを実パラメターで置き換えるには declare 文を使用する．実パラメターとして使えるのは与えられた頂点または辺を指すポインタ名である．これを事例によって説明する．実パラメターの名前を Node, Edge とすると，置き換えのための宣言文は最初の3つのクラスに対して

```
declare(Vlist, Node); declare(Elist, Edge);
declare2(graph, Node, Edge);
```

となる．declare 文が置かれた場所に実例としてのクラス定義が作られる．また，クラス adj の定義には，クラス Mu と Adjをあらかじめ定義しておくことが必要である．

```
declare(Mu, Node); declare(Adj, Edge);
declare2(adj, Node, Edge);
```

ここで，実パラメター Node と Edge を具体的に定義するには，まず基底レベルのクラス baseNode と baseEdge から下記のクラス node と edge を導出する．

```
class node : public baseNode {
  public :
  int num, label;
  node() {}
  node(idNode Id); ~node() {}
};
class edge : public baseEdge {
  public:
    int cost;
    edge() {}
    edge(idNode Orig, idNode Dest, int Cost);
    ~edge() {}
};
```

これらから Node と Edge は次のように定義する．

```
typedef node* Node; typedef edge* Edge;
```

ここでツールを使う簡単なプログラムの実例として半順序集合の整列化のプログラムの一部の断片を示す．

```
#define top(idx) next(idx)
#define slot(idx, name) vtxPtr(idx)->name
void topsort(adj(Node, Edge)& A, idxNode& v){
  A.slot(v, num)=++i;
  for(Edge e=A.top(v); e; e=A.next(v)) {
      idxNode w=e→ idxDst();
      if(A.slot(w, num)==0) topsort(A, w);
      else
        if(A.slot(w, label)==0) exit(1);
    }
```

```
    A.slot(v,label) =--j;
}
```

4. 利用者に公開された主要なクラス・ライブラリとそのメンバ関数

クラス graph(type1,type2)のメンバ関数

graph(type1,type2)(); ~graph(type1,type2)();

コンストラクタとデストラクタ.

void getVset(Vlist(type1)& V);

void getEset(Elist(type2)& E);

グラフから頂点または辺集合を取りだす.

クラス Vlist(type1) から継承された graph(type1,type2)のメンバ関数

void insertVtx(type1 v); void appendVtx(type1 v);

頂点vを頂点集合リストの先頭または末尾に追加する.

type1 getVtx(); (破壊的取り出し)

頂点集合リストの先頭要素を取り除く.

void resetVlist();

頂点集合リスト上の着目点をリストの先頭位置に戻し, リフレッシュする.

int lengthVlist();

頂点集合リストの長さを関数値としてかえす.

type1 nextVtx();

頂点集合リストで現在着目している頂点を取りだし, 着目点を次の位置に移動する.

type1 seeTopVtx();

頂点集合リストの先頭要素を読みだす.

void printVtx();

クラス vlist から継承された graph(type1,type2) のメンバ関数

int isMember(ent a);

指定された頂点と同じ認識名 (id値) をもつ要素が頂点集合リストの中にあれば真, なければ偽.

クラス Elist(type2) から継承された graph(type1,type2) のメンバ関数は Vlist(type) のメンバ関数と相似なので説明は省略する.

void insertEdge(type2 e);

void appendEdge(type2 e);

type2 getEdge(); void resetElist();

int lengthElist(); type2 nextEdge();

type2 seeTopEdge(); void printElist();

クラス elist から継承された graph(type1,type2) のメンバ関数として int isMember(ent a); がある. また, クラス alist から継承されたメンバ関数には次のものがある.

int isEmpty();

頂点集合リストまたは辺集合リストが空ならば真, そうでなければ偽をかえす.

virtual int isMember(ent a)=0;

任意の型の頂点または辺のリストに対して isMember 関数を派生させるための土台となる純粋仮想関数.

利用者に公開されたクラス class adi (type1,type2)のメンバ関数

void accept(Elist(type2)& E);

void printAdj (); adj(type1,type2)(Vlist(type1)& V, Elist(type2)& E);

頂点集合と辺集合から接続関係とインデックスによる計算環境を生成する構成子.

~adj(type1,type2) ();

クラス Adj(type2)から継承された adj(type1,type2) のメンバ関数

Elist (type2)* getElist (idxNode i);

インデックス値 i の頂点に接続する辺リストを指すポインタを取りだす.

void insert(idxNode i, type2 e);

インデックス値 i をもつリストの先頭に辺を挿入.

void append(idxNode i, type2 e);

インデックス i をもつリストの末尾に辺を追加.

type2 get(idxNode i);

インデックス i をもつリストの先頭要素を取り除く (破壊的取りだし)

type2 next(idxNode i);

インデックス i をもつリストの先頭要素を読みだす (非破壊的読みだし)

クラス adjacent から継承された adj (type1,type2)のメンバ関数

void clear(idxNode i);

インデックス値 i をもつリストを完全に除去する.

void reset(idxNode i);

インデックス値 i のリストの現在着目位置をリストの先頭にもどす.

int length(idxNode i);

void print ();

クラス Mu(type1)から継承された adj(type1, type2) のメンバ関数

int toIdx(type1 v);

type1 型要素により指される頂点のインデックス値をかえす.

type1 vtxPtr(idxNode i);

インデックス値 i の頂点のポインタ値をかえす.

void printMu();

クラス mutator から継承された adj(type1,type2) のメンバ関数

idxNode toidx(idNode id);

認識番号 id の頂点のインデックス値をかえす.

idNode toIdNum(idNode i);

インデックス値 i をもつ頂点の id 値をかえす.

5. あとがき

ツールの性能を確かめるために，単純閉路探索，二重連結成分，強連結成分などのグラフ算法をはじめ，最短経路問題，最大流量問題，ＰＥＲＴ計算などの典型的な属性依存問題，また現在研究中のネットワーク問題について，ツール支援下でプログラミングを行なってみた．その経験から，大筋でほぼ満足な結果を得ることができ，有効性の確認ができたと感じている．今後さらにツールを強化する方向として，Ｃ＋＋の処理系がすでに用意しているコンテナ・クラス・ライブラリとの連携をはかる，基本的グラフ算法が応用レベルのプログラム中で自由に利用できるようにする，を目標として考えている．

参考文献

1) プログラミング言語Ｃ＋＋
B．ストラウストラップ著

2) Borland C++20 Programmer's Guide Borland 社

会合記録

8月3日（火）表彰委員会　10名
8月19日（木）機関誌編集委員会　13名
8月19日（木）ＯＲ基本課題委員会　15名