Chapitre 15

La classe Graphe

15.1 Les choix

On fait souvent l'hypothèse que les sommets d'un graphe sont numérotés de 0 à $n-1$, de façon à utiliser des implantations à base de matrices d'adjacence ou bien de pouvoir faire des calculs. Toutefois, de nombreux algorithmes sur les graphes sont écrits en manipulant des sommets abstraits. Il nous a paru plus proche d'une certaine réalité d'en faire autant.

Après réflexion, écrire une classe générique de graphe est une tâche quasi-impossible, car chaque algorithme qui opère sur les graphes utilisent des structures de données auxiliaires et on ne peut envisager tous les cas possibles. Néanmoins, une certaine abstraction est possible. Cela a conduit à définir une classe `Sommet` qui code un sommet, dont le nom est une chaîne, et qui contient un marqueur, qui servira par exemple à modéliser l'état d'un sommet lors d'un parcours (bfs ou dfs). Il serait facile de modifier ce type `Sommet` en fonction des applications. Pour aller au-delà, la classe `Arc` s'est imposée, un arc générique étant formé de deux sommets (origine et extrémité) et doté d'une valeur (un entier). On peut alors réaliser des listes de sommets, des piles d'arcs, etc.

On peut alors définir une classe abstraite `Graphe` qui nous permettra d'écrire tous les algorithmes du poly. Cette classe est accompagnée de deux implantations : `GrapheMatrice` (pour utiliser des matrices d'adjacence) et `GrapheListe` qui utilise des tableaux de listes de voisin. La figure 15.1 donne un aperçu graphique du contenu du package et des liens entre les différentes classes.

Tout a été écrit dans `Graphe` de façon à privilégier les parcours sur les voisins d'un sommet, sans savoir quelle représentation sera finalement prise pour un problème donné. Pour faciliter le traitement des sommets du graphe, on introduit une classe numérotation qui permet d'assurer un ordre sur les sommets par exemple. Pour en comprendre l'intérêt, on regardera le code de la fonction `deFichier` de la classe `GrapheListe`.

Remerciements : les classes mises au point ont bénéficié de l'aide irremplaçable de P. Chassignet. Elles ont été testées en même que le poly a été écrit. Elles sont disponibles

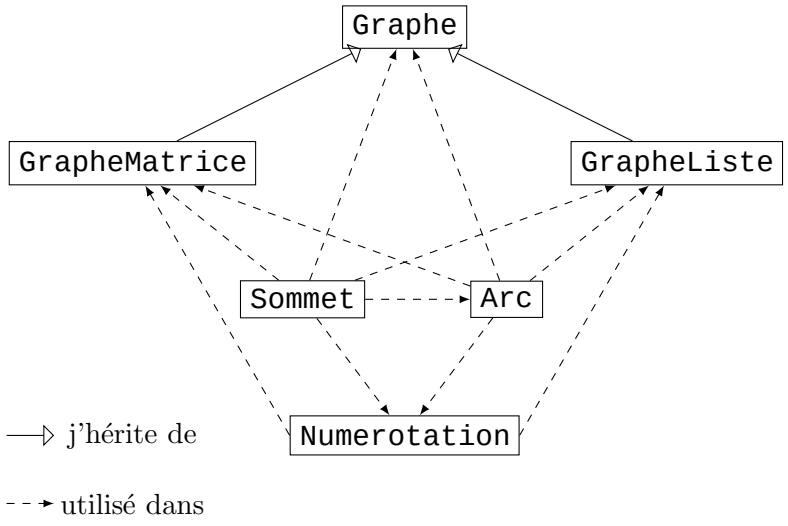

FIG. 15.1 – Panorama des classes du package `grapheX`.

sur la page web du cours.

15.2 grapheX comme exemple de package

Il est apparu naturel de regrouper tout cela dans un **`package`** au sens de Java, auquel le nom de **grapheX** a été donné. Cela permet de donner un exemple réel d'utilisation de package.

15.2.1 Comment ça marche ?

Après récupération sur le réseau[1], on exécute les commandes :

```
tar zxvf grapheX.tgz
```

Cela crée un répertoire `grapheX`, qui contient les fichiers sources :

```
Arbre.java  Graphe.java       GrapheMatrice.java  Sommet.java
Arc.java    GrapheListe.java  Numerotation.java
```

Notez la commande **`package`** `grapheX;` présente dans chacun des fichiers.

Au même niveau que `grapheX` se trouve un fichier de menu étendu qui montre comment utiliser les fonctions contenues dans le package (fichier `TestGraphe.java`). Ce fichier se compile par :

```
javac TestGraphe.java
```

et ira compiler tout seul dans le répertoire `grapheX` les différentes classes, les fichiers `.class` restant dans `grapheX`.

[1] `http://www.enseignement.polytechnique.fr/informatique/INF431/`

15.2.2 Faire un .jar

Si on le souhaite, on peut faire un `.jar`. Nous en profitons pour donner la syntaxe (en Unix), qui n'est pas si évidente que cela :

```
javac grapheX/*.java # il faut les .class!
jar -cf grapheX.jar grapheX
```

On peut alors compiler `TestGraphe.java` :

```
javac -cp grapheX.jar:. TestGraphe.java
```

Pour exécuter, il suffit de taper

```
java -cp grapheX.jar:. TestGraphe ../Data/prim1.in Prim
```

15.3 Sommets

Un sommet d'un graphe sera supposé contenir comme information une chaîne de caractères. On a ajouté une marque, qui sera utilisée (à travers ses méthodes d'accès), lors des explorations, pour gérer l'état du sommet. On peut facilement afficher un sommet `s` à l'écran par

```
    System.out.println(s);
```

Nous aurons besoin de piles ou de tables de hachage de sommets. Pour permettre une gestion agréable, nous avons spécialisé les méthodes `equals` et `hashCode`. Cela permettra d'écrire du code :

```
import java.util.*;
import grapheX.*;

public class Test{
  public static void main(String args[]){
    HashSet<Sommet> HS = new HashSet<Sommet>();
    Sommet s = new Sommet("a", 0);
    HS.add(s);
    if(HS.contains(new Sommet("a", 1)))
        System.out.println(true);
  }
}
```

et qu'il affiche **`true`**. Notons que la valeur de hachage dépendra du nom du sommet, pas de l'état du marqueur.

```
package grapheX;

/**
```

```
   Classe de Sommets

   @author FMorain (morain@lix.polytechnique.fr)
   @author PChassignet (chassign@lix.polytechnique.fr)
   @version 2006.11.27
*/

public class Sommet{
    String nom;
    private int marque;

    public Sommet(String nn, int mm){
        nom = nn;
        marque = mm;
    }

    public Sommet(Sommet s, int mm){
        nom = s.nom;
        marque = mm;
    }

    public int valeurMarque(){
        return marque;
    }

    public void modifierMarque(int m){
        marque = m;
    }

    public boolean equals(Object o){
        return nom.equals(((Sommet)o).nom);
    }

    public int compareTo(Object o){
        Sommet s = (Sommet)o;
        return nom.compareTo(s.nom);
    }

    public String toString(){
        return ""+nom;
    }

    public int hashCode(){
        return nom.hashCode();
```

```
    }
}
```

15.4 La classe Arc

Nous utilisons les mêmes principes que pour la classe `Sommet` en permettant la gestion facile de pile ou tables d'Arcs.

```
package grapheX;

import java.io.*;
import java.util.*;

/**
   Classe d'arcs

   @author FMorain (morain@lix.polytechnique.fr)
   @author PChassignet (chassign@lix.polytechnique.fr)
   @version 2007.01.30
*/

// L'arc o -> d avec valeur val
public class Arc{
    private Sommet o, d;
    private int val;

    public Arc(Sommet o0, Sommet d0, int val0){
        this.o = o0;
        this.d = d0;
        this.val = val0;
    }

    public Arc(Arc a){
        this.o = a.o;
        this.d = a.d;
        this.val = a.val;
    }

    public Sommet destination(){
        return d;
    }

    public Sommet origine(){
```

```
        return o;
    }

    public int valeur(){
        return val;
    }

    public void modifierValeur(int vv){
        this.val = vv;
    }

    public boolean equals(Object aa){
        Arc a = (Arc)aa;
        return o.equals(a.o) && d.equals(a.d) && (val == a.val);
    }

    public String toString(){
        return "("+this.o+", "+this.d+")";
    }

    public int hashCode(){
        String str = ""+this;
        return str.hashCode();
    }
}
```

Une remarque sur la méthode `hashCode`. On veut qu'un arc soit repérable par ses deux sommets, pas par sa valeur. On pourrait changer ce comportement si besoin est.

15.5 La classe abstraite Graphe et deux implantations

Le but de la classe `Graphe` est de permettre de manipuler facilement des graphes, comme on peut le lire dans les chapitres du poly concernant les graphes. Notons en passant que nous utiliserons systématiquement des itérateurs sur les sommets d'un graphe ou les voisins d'un sommet.

Le parti pris est celui d'une collection d'arcs, donc d'un graphe *a priori* orienté. Dans de rares cas, l'utilisation d'une classe plus spécifique d'arête au lieu d'arc pourrait être envisagée.

15.5.1 Définitions

Les quelques méthodes décrites suffisent à implanter tous les algorithmes décrits dans le cours.

```
package grapheX;

import java.io.*;
import java.util.*;

/**
   Classe abstraite de graphes

   @author FMorain (morain@lix.polytechnique.fr)
   @version 2007.01.30 [ajouterArc devient public]
*/

public abstract class Graphe{

    public abstract int taille();
    public abstract Graphe copie();

    public abstract void ajouterSommet(Sommet s);
    public abstract boolean existeArc(Sommet s, Sommet t);
    public abstract void ajouterArc(Sommet s, Sommet t, int val);
    public abstract int valeurArc(Sommet s, Sommet t);
    public abstract void enleverArc(Sommet s, Sommet t);

    public abstract Collection<Sommet> sommets();
```

15.5.2 Numérotation

Cette classe est utilisée dans les deux implantations de la classe abstraite, car nous avons souvent besoin (en interne) d'un ordre sur les sommets.

Il est important, en interne, d'avoir un moyen de numéroter les sommets lors de leur création. C'est le rôle de cette classe, qui sera utilisée dans les deux classes `GrapheMatrice` et `GrapheListe`. Pour que l'utilisateur ne soit pas tenté d'utiliser cette numérotation, tous les champs sont privés. On peut récupérer des itérateurs sur les sommets, au moyen de la méthode `elements()`.

```
package grapheX;

import java.io.*;
import java.util.*;

/**
   Numérotation des graphes

   @author FMorain (morain@lix.polytechnique.fr)
```

```
   @author PChassignet (chassign@lix.polytechnique.fr)
   @version 2006.11.22
*/

public class Numerotation{
    private int compteur;
    private Hashtable<Sommet,Integer> HSI;
    private Vector<Sommet> VS;

    public Numerotation(int n){
        compteur = -1;
        HSI = new Hashtable<Sommet,Integer>();
        VS = new Vector<Sommet>(n);
        VS.setSize(n);
    }

    public int taille(){
        return VS.size();
    }

    public boolean ajouterElement(Sommet s){
        if(!HSI.containsKey(s)){
            compteur++;
            HSI.put(s, compteur);
            VS.set(compteur, s);
            return true;
        }
        return false;
    }

    public int numero(Sommet s){
        return HSI.get(s);
    }

    public Sommet elementAt(int i){
        return VS.elementAt(i);
    }

    public Collection<Sommet> elements(){
        return VS;
    }
}
```

15.5.3 Implantation par matrice

C'est là une implantation proche des matrices d'adjacence, même si on utilise des vecteurs de vecteurs.

```
package grapheX;

import java.io.*;
import java.util.*;

/**
   Graphes implantés dans des "matrices"

   @author FMorain (morain@lix.polytechnique.fr)
   @version 2007.01.12
*/

public class GrapheMatrice extends Graphe{
    private Vector<Vector<Arc>> M;
    private Numerotation numerotation;

    public int taille(){
        return M.size();
    }

    public GrapheMatrice(int n){
        numerotation = new Numerotation(n);
        M = new Vector<Vector<Arc>>(n);
        M.setSize(n);
    }

    public void ajouterSommet(Sommet s){
        if(numerotation.ajouterElement(s)){
            int n = taille();
            Vector<Arc> vs = new Vector<Arc>(n);
            vs.setSize(n);
            M.set(numerotation.numero(s), vs);
        }
    }

    public boolean existeArc(Sommet s, Sommet t){
        int si = numerotation.numero(s);
        int ti = numerotation.numero(t);
        return M.get(si).get(ti) != null;
    }
```

```
    private boolean existeArc(int i, int j){
        return M.get(i).get(j) != null;
    }

    public void ajouterArc(Sommet s, Sommet t, int val){
        ajouterSommet(s);
        ajouterSommet(t);
        int si = numerotation.numero(s);
        int ti = numerotation.numero(t);
        M.get(si).set(ti, new Arc(s, t, val));
    }

    public int valeurArc(Sommet s, Sommet t){
        int si = numerotation.numero(s);
        int ti = numerotation.numero(t);
        return M.get(si).get(ti).valeur();
    }

    private int valeurArc(int i, int j){
        return M.get(i).get(j).valeur();
    }

    public void enleverArc(Sommet s, Sommet t){
        int si = numerotation.numero(s);
        int ti = numerotation.numero(t);
        M.get(si).remove(ti);
    }

    public void modifierValeur(Sommet s, Sommet t, int val){
        int si = numerotation.numero(s);
        int ti = numerotation.numero(t);
        M.get(si).get(ti).modifierValeur(val);
    }

    public LinkedList<Arc> voisins(Sommet s){
        LinkedList<Arc> l = new LinkedList<Arc>();
        int si = numerotation.numero(s);

        for(int j = 0; j < taille(); j++)
            if(existeArc(si, j))
                l.addLast(M.get(si).get(j));
        return l;
    }
```

```
    public Collection<Sommet> sommets(){
        return numerotation.elements();
    }

    public GrapheMatrice copie(){
        int n = taille();
        GrapheMatrice G = new GrapheMatrice(n);
        for(int i = 0; i < n; i++)
            G.ajouterSommet(numerotation.elementAt(i));
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                if(M.get(i).get(j) != null)
                    G.ajouterArc(numerotation.elementAt(i),
                                 numerotation.elementAt(j),
                                 valeurArc(i, j));
        return G;
    }

    public static GrapheMatrice deMatrice(int[][] M){
        int n = M.length;
        GrapheMatrice G = new GrapheMatrice(n);

        for(int i = 0; i < n; i++)
            G.ajouterSommet(new Sommet(i+"", 0));
        for(int i = 0; i < n; i++){
            for(int j = 0; j < M[i].length; j++)
                if(M[i][j] == 1)
                    G.ajouterArc(G.numerotation.elementAt(i),
                                 G.numerotation.elementAt(j),
                                 1);
        }
        return G;
    }

}
```

15.5.4 Implantation avec des listes

```
package grapheX;

import java.io.*;
import java.util.*;
```

```
/**
   Graphes implantés dans des "listes"

   @author FMorain (morain@lix.polytechnique.fr)
   @version 2007.01.30 [propagation modifs de Arc]
*/

public class GrapheListe extends Graphe{
    private Vector<LinkedList<Arc>> L;
    private Numerotation numerotation;

    public int taille(){
        return L.size();
    }

    public GrapheListe(int n){
        numerotation = new Numerotation(n);
        L = new Vector<LinkedList<Arc>>(n);
        L.setSize(n);
    }

    public void ajouterSommet(Sommet s){
        if(numerotation.ajouterElement(s))
            L.set(numerotation.numero(s), new LinkedList<Arc>());
    }

    public boolean existeArc(Sommet s, Sommet t){
        for(Arc a : L.get(numerotation.numero(s)))
            if((a.destination()).equals(t))
                return true;
        return false;
    }

    private boolean existeArc(int i, int j){
        Sommet t = numerotation.elementAt(j);
        for(Arc a : L.get(i))
            if(a.destination().equals(t))
                return true;
        return false;
    }

    public void ajouterArc(Sommet s, Sommet t, int val){
        ajouterSommet(s);
```

```
        ajouterSommet(t);
        int si = numerotation.numero(s);
        L.get(si).addLast(new Arc(s, t, val));
    }

    public void ajouterArc(int i, int j, int val){
        L.get(i).addLast(new Arc(numerotation.elementAt(i),
                                 numerotation.elementAt(j),
                                 val));
    }

    public int valeurArc(Sommet s, Sommet t){
        for(Arc a : L.get(numerotation.numero(s)))
            if(a.destination().equals(t))
                return a.valeur();
        return -1; // convention
    }

    public int valeurArc(int i, int j){
        Sommet t = numerotation.elementAt(j);
        for(Arc a : L.get(i))
            if(a.destination().equals(t))
                return a.valeur();
        return -1; // convention
    }

    public void enleverArc(Sommet s, Sommet t){
        int si = numerotation.numero(s);
        Arc a = null;
        for(Arc aa : L.get(numerotation.numero(s)))
            if(aa.destination().equals(t)){
                a = aa;
                break;
            }
        if(a != null)
            L.get(numerotation.numero(s)).remove(a);
    }

    public void modifierValeur(Sommet s, Sommet t, int val){
        for(Arc a : L.get(numerotation.numero(s)))
            if(a.destination().equals(t)){
                a.modifierValeur(val);
                return;
            }
```

```
    }

    public LinkedList<Arc> voisins(Sommet s){
        return L.get(numerotation.numero(s));
    }

    public Collection<Sommet> sommets(){
        return numerotation.elements();
    }

    public GrapheListe copie(){
        int n = taille();
        GrapheListe G = new GrapheListe(n);
        for(int i = 0; i < n; i++)
            G.ajouterSommet(numerotation.elementAt(i));
        for(int i = 0; i < n; i++){
            // recopie dans le même ordre
            LinkedList<Arc> Li = G.L.get(i);
            for(Arc a : L.get(i))
                Li.addLast(a);
        }
        return G;
    }

    // retourne vrai si le caractère c est dans str
    private static boolean option(String str, char c){
        for(int i = 0; i < str.length(); i++)
            if(str.charAt(i) == c)
                return true;
        return false;
    }

    public static GrapheListe deFichier(String nomfic){
        try{
            Scanner scan =
                new Scanner(
                    new BufferedReader(new FileReader(nomfic)));
            System.out.println(scan.next());
            int n = scan.nextInt();
            GrapheListe G = new GrapheListe(n);
            String str = scan.next();
            boolean estValue = option(str, 'v');
            boolean estSym = option(str, 's');
            boolean avecCouples = option(str, 'c');
```

```
            System.out.println("n = "+n);
            for(int i = 0; i < n; i++){
                Sommet s = new Sommet(scan.next(), 0);
                G.ajouterSommet(s);
            }
            if(avecCouples){
                System.out.println("Avec couples");
                // on lit des lignes "i j dij" ou "i j"
                while(scan.hasNext()){
                    Sommet s = new Sommet(scan.next(), 0);
                    Sommet t = new Sommet(scan.next(), 0);
                    int si = G.numerotation.numero(s);
                    int ti = G.numerotation.numero(t);
                    if(estValue)
                        G.ajouterArc(si, ti, (int)scan.nextInt());
                    else
                        G.ajouterArc(si, ti, 1);
                }
            }
            else{
                // format "s ns t0 t1 ... t{ns-1}"
                // ou     "s ns t0 v0 t1 v1 ... t{ns-1} v{ns-1}"
                System.out.println("Avec listes, estvalue="+estValue);
                for(int r = 0; r < n; r++){
                    Sommet s = new Sommet(scan.next(), 0);
                    int si = G.numerotation.numero(s);
                    int nj = (int)scan.nextInt();
                    for(int k = 0; k < nj; k++){
                        Sommet t = new Sommet(scan.next(), 0);
                        int ti = G.numerotation.numero(t);
                        if(estValue)
                            G.ajouterArc(si, ti,
                                         (int)scan.nextInt());
                        else
                            G.ajouterArc(si, ti, 1);
                    }
                }
            }
            System.out.println("G="+G);
            if(estSym)
                // on doit symétriser G
                for(Sommet s : G.sommets())
                    for(Sommet t : G.sommets())
```

```
                        if(G.existeArc(s, t)
                           && !G.existeArc(s, t))
                            G.ajouterArc(s, t,
                                         G.valeurArc(s, t));
            return G;
        }
        catch(Exception e) { }
        return null;
    }

}
```

La méthode `deFichier` donne un exemple d'utilisation des différentes primitives. Donnons un exemple de fichier d'entrée. Le fichier :

```
#prim1
4 vs
u v w x
u 2 v 16 x 29
v 3 u 16 x 20 w 25
w 2 v 25 x 10
x 3 u 29 v 20 w 10
```

La première ligne est un commentaire qui n'est pas traité par le programme. La seconde contient le nombre de sommets du graphe. Suivent des codes comme `v` pour graphe valué, `s` pour graphe symétrique (non orienté). La ligne suivante contient les noms des 4 sommets du graphe. Ils seront stockés dans cet ordre dans une table, ce qui fait que

```
for(Sommet s : sommets())
    System.out.println(s);
```

affichera

```
u
v
w
x
```

Suivent alors 4 lignes. Chacune commence par un nom de sommet, puis par le nombre de voisins k, puis k paires nom de sommet suivi de la valeur de l'arc. Ce fichier correspond au graphe de la figure 8.6.

Un graphe non valué est stocké sous une forme plus simple :

```
#pda1
6 s
a b c d e f
a 4 b d c f
```

```
b 2 a d
c 3 a e f
d 2 a b
e 2 a c
f 1 c
```

C'est le graphe représenté à la figure 6.5.