Contrôle Hors-Classement — Cours INF 431

Jean Berstel Jean-Jacques Lévy

Ecole polytechnique, 29 Avril 2003

Avertissement : la rédaction doit être claire, concise et précise.

Hauteur d'un *dag*

On suppose les graphes représentés par des listes de successeurs (comme dans le cours). La longueur d'un chemin est le nombre d'arcs contenus dans ce chemin. Soit G un graphe orienté sans cycle (*dag*). Le *rang* de tout sommet est la longueur du plus long chemin issu de ce sommet. Ainsi le rang de tout sommet est le rang maximal de ses sommets successeurs, augmenté de un. La *hauteur* de G est le rang maximal de ses sommets.

Question 1 Ecrire une fonction hauteur(g) qui retourne la hauteur du graphe G. Quelle est la complexité de cette fonction ?

Représentation des ensembles

On représente les ensembles finis d'entiers positifs ou nuls par une classe Ensemble et par héritage en implémentant un type disjonctif selon la décomposition suivante pour tout ensemble E :

$$\begin{array}{rcll} E & ::= & \emptyset & (E \text{ ensemble vide}) \\ & \mid & \{x\} \cup E' & (E \text{ ensemble non vide, } x \notin E') \end{array}$$

Question 2 Ecrire la déclaration de la classe Ensemble, de ses éventuelles sous-classes associées et de ses constructeurs.

Question 3 Ecrire les méthodes card, contient et ajouter telles que, pour tout ensemble E représenté par l'objet e, les expressions e.card(), e.contient(x) et e.ajouter(x) valent respectivement le nombre d'éléments n de E, le booléen vrai si x appartient à E et faux sinon, et l'ensemble $\{x\} \cup E$. Dans le dernier cas, la méthode ajouter ne change pas e quand x appartient à E. Ces méthodes pourront avoir une complexité en $O(n)$.

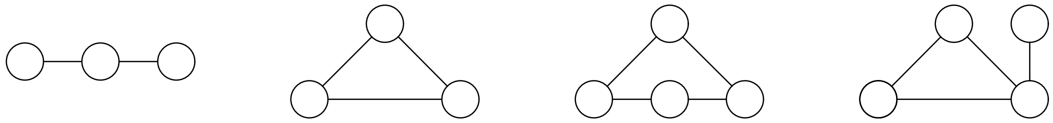
Question 4 Ecrire la méthode prochain telle que e.prochain(i,k) vaut le plus petit entier x tel que $x \notin E$ et $i < x < k$. Si cet entier n'existe pas, la valeur retournée sera -1.

Question 5 Ecrire la méthode toString telle que e.toString() est une chaîne de caractères donnant une représentation imprimable de E.

Coloriages de graphes non-orientés

Un graphe non-orienté est coloriable avec k couleurs s'il existe une fonction c de l'ensemble des sommets dans $\{0, \ldots, k-1\}$ (les *couleurs*) telle que $c(s) \neq c(t)$ s'il existe un arc de s à t. Savoir si un graphe est coloriable avec k couleurs est un problème NP difficile. Mais, on peut écrire facilement un programme (de complexité exponentielle) qui liste tous les coloriages possibles d'un graphe avec k couleurs.

Question 6 Donner des coloriages (en affectant k couleurs appropriées aux sommets) pour les graphes suivants, avec k minimum pour chaque graphe.



Question 7 Montrer que, pour tout graphe G et k suffisamment grand, il existe toujours un coloriage dès que G ne contient pas de boucles (une boucle est un arc dont l'origine et l'extrémité coïncident).

On suppose à présent que le graphe considéré n'a pas de boucles, et que le coloriage c est représenté par un tableau d'entiers couleur tel que $c(x) = \text{couleur}[x]$ pour tout sommet x ($0 \leq x < n$, $\text{couleur}[x] \geq 0$).

Pour énumérer tous les coloriages de G avec k couleurs, une première technique consiste à énumérer tous les n-uplets de k^n et à ne lister que ceux correspondants à des coloriages (licites). Une technique un peu plus rapide consiste à ne choisir, pour tout sommet, que des couleurs qui ne sont pas dans l'ensemble des couleurs déjà affectées à ses voisins. C'est cette deuxième technique qu'il s'agit d'utiliser dans la question suivante.

Question 8 Ecrire une fonction colorier telle que colorier(g,k) imprime tous les coloriages possibles du graphe G avec k couleurs. (On supposera qu'il existe une fonction imprimerSolution telle que imprimerSolution(couleur) imprime le tableau couleur sur une ligne.)

Corrigé

Question 1 La fonction hauteur(g) prend un temps en $O(V + E)$.

```
static int hauteur (Graphe g) {
  int n = g.succ.length; int[ ] rang = new int[n];
  int h = -1;
  for (int x=0; x < n; ++x) rang[x] = -1;
  for (int x=0; x < n; ++x) {
    if (rang[x] == -1)
      calculerRangDe (g, x, rang);
    h = Math.max(h, rang[x]);
  }
  return h;
}

static void calculerRangDe (Graphe g, int x, int[ ] rang) {
  int r = 0;
  for (Liste ls = g.succ[x]; ls != null; ls = ls.suivant) {
    int y = ls.val;
    if (rang[y] == -1)
      calculerRangDe (g, y, rang);
    r = Math.max (r, 1 + rang[y]);
  }
  rang[x] = r;
}
```

Question 2

```
abstract class Ensemble { }

class Vide extends Ensemble { }

class NonVide extends Ensemble {
  int elt;
  Ensemble reste;
  NonVide (int x, Ensemble e) { elt = x; reste = e; }
}
```

Question 3

```
abstract class Ensemble {
  abstract int card ();
  abstract boolean contient (int x);
  abstract Ensemble ajouter (int x);
}

class Vide extends Ensemble {
  int card () { return 0; }
  boolean contient (int x) {return false; }
  Ensemble ajouter (int x) {return new NonVide (x, this); }
  public String toString () { return ""; }
}

class NonVide extends Ensemble {
  int elt;
  Ensemble reste;
  NonVide (int x, Ensemble e) { elt = x; reste = e; }

  int card () { return 1 + reste.card(); }
```

```
  boolean contient (int x) {
    return elt == x  reste.contient(x);
  }

  Ensemble ajouter (int x) {
    if ( contient(x) ) return this;
    else return new NonVide (x, this);
  }
}
```

Question 4

```
abstract class Ensemble {
  int prochain (int i, int k) {
    do ++i;
    while (i < k && contient(i));
    return i < k ? i : -1;
  }
}
```

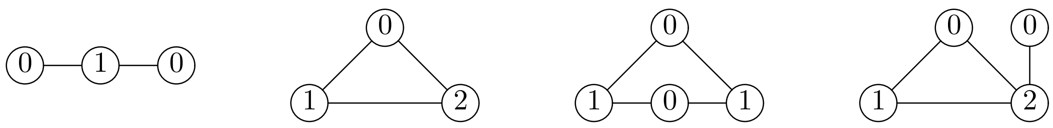
Question 5

```
abstract class Ensemble {...}

class Vide extends Ensemble {
  public String toString () { return ""; }
}

class NonVide extends Ensemble {
  public String toString () { return elt + " " + reste; }
}
```

Question 6



Question 7 Si k est suffisamment grand, on peut prendre comme coloriage $c(x) = x$ pour tout sommet x. C'est un coloriage s'il n'y a pas de boucle.

Question 8

```
  static void colorier (Graphe g, int k) {
    int n = g.succ.length;
    int[ ] couleur = new int[n];
    for (int x=0; x < n; ++x) couleur[x] = -1;
    colorier1 (g, k, 0, couleur);
  }

  static void colorier1 (Graphe g, int k, int i, int[ ] couleur) {
    int n = g.succ.length;
    if (i == n) imprimerSolution (couleur);
    else {
      Ensemble e = new Vide();
      for (Liste ls = g.succ[i]; ls != null; ls = ls.suivant) {
        int y = ls.val;
        if (couleur[y] != -1)
```

```
        e = e.ajouter (couleur[y]);
    }
    for (int c = e.prochain(couleur[i], k); c != -1; c = e.prochain(c, k) ) {
      couleur[i] = c;
      colorier1 (g, k, i+1, couleur);
      couleur[i] = -1;
    }
  }
}
```

Cette dernière fonction peut aussi s'écrire sous la forme suivante plus succincte (mais moins claire) :

```
static void colorier1 (Graphe g, int k, int i, int[ ] couleur) {
  int n = g.succ.length;
  if (i == n) imprimerSolution (couleur);
  else {
    Ensemble e = new Vide();
    for (Liste ls = g.succ[i]; ls != null; ls = ls.suivant) {
      int y = ls.val;
      if (couleur[y] != -1)
        e = e.ajouter (couleur[y]);
    }
    while ((couleur[i] = e.prochain(couleur[i], k)) != -1)
      colorier1 (g, k, i+1, couleur);
  }
}
```