# Contrôle Classant – Informatique 431

Dominique Perrin *

3 juillet 2002

*Tous les documents du cours sont autorisés. On attachera une grande importance à la concision, à la clarté, et à la précision de la rédaction.*

## Partie I – Graphes eulériens

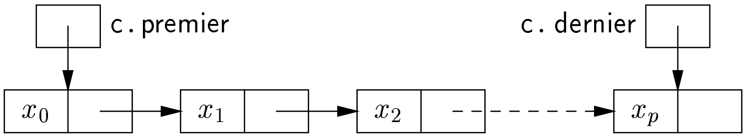
On considère des graphes orientés dont les $n$ sommets ($n > 0$) sont des entiers. On représente les graphes comme dans le cours par un tableau de listes de successeurs.

```
class Graphe {
  Liste [ ] succ;
  Graphe (int n) {
    succ = new Liste[n];
  }
}
```

```
class Liste {
  int val;
  Liste suivant;
}
```

```
class Chemin {
  Liste premier;
  Liste dernier;
}
```

Un chemin $\langle x_0, x_1, \ldots x_p\rangle$ de $x_0$ à $x_p$ est représenté par la liste des sommets $x_0$, $x_1$, $\ldots x_p$ par lesquels il passe ($p \geq 0$). Sa longueur est $p$. Pour faciliter la concaténation (destructive) de deux chemins, un objet de la classe Chemin contiendra deux champs premier et dernier indiquant le début et la fin de la liste des sommets par lesquels il passe. Le chemin $c$ vide au sommet $x$ correspond au cas où `c.premier == c.dernier`.




**Question 1** Ecrire les constructeurs suivants :

a) `Chemin(int x)` retournant le chemin vide au sommet $x$.

b) `Chemin(int x, Chemin c)` retournant le chemin qui démarre en $x$ puis emprunte le chemin $c$.

c) `Chemin(Chemin c, int x)` retournant le chemin qui commence par $c$ puis qui se termine en $x$.

**Question 2** Ecrire une fonction `concat(Chemin c, Chemin d)` qui retourne, en temps $O(1)$, le chemin obtenu par concaténation des deux chemins $c$ et $d$.

**Question 3** Ecrire une méthode `toString()` dans la classe Chemin qui retourne, pour tout chemin $c = \langle x_0, x_1, \ldots x_p\rangle$, une chaîne de caractères `c.toString()` contenant tous les sommets $x_i$ ($0 \leq i \leq p$) figurant sur $c$ ($p \geq 0$). Expliquer pourquoi on peut alors écrire des ordres d'impression tels que `System.out.println(c)`.

Pour un sommet $x$, on note $d^+(x)$ le nombre d'arcs d'extrémité $x$, et $d^-(x)$ le nombre d'arcs d'origine $x$ ; on pose $\delta(x) = d^+(x) - d^-(x)$. On dit qu'un graphe est *eulérien* si, pour chaque sommet $x$, on a $\delta(x) = 0$.

**Question 4** Indiquer les graphes eulériens parmi les graphes de la figure 1.

* avec les participations de Georges Gonthier, Jean-Jacques Lévy, Luc Maranget

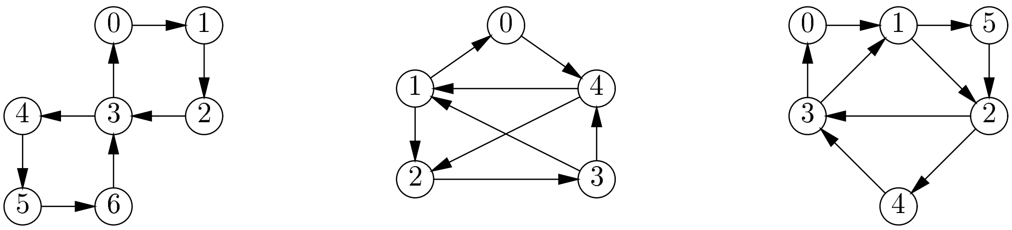


FIG. 1 – Graphes $(a)$, $(b)$, $(c)$

**Question 5** Ecrire les fonctions suivantes :

a) `delta (Graphe g)` qui retourne un tableau donnant la valeur de $\delta(x)$ pour tout sommet $x$.

b) `estEulerien (Graphe g)` qui teste si le graphe $G$ est eulérien. Quelle est sa complexité ?

Un *circuit eulérien* est un chemin $\langle x_0, x_1, \ldots x_p \rangle$ tel que $x_0 = x_p$ et qui utilise exactement *tous* les arcs du graphe une seule fois. Ainsi $\langle 0, 1, 2, 3, 1, 5, 2, 4, 3, 0 \rangle$ est un circuit eulérien dans le graphe $(c)$ de la figure 1. Donc si $E$ est l'ensemble des arcs d'un graphe, un circuit eulérien est de longueur $|E|$ où $|E|$ est la cardinalité de $E$. Nous allons démontrer qu'un graphe $G$ admet un circuit eulérien si et seulement s'il est eulérien et fortement connexe.

Une composante connexe de $G$ est une composante connexe du graphe non-dirigé obtenu en oubliant l'orientation des arcs de $G$. Un *chemin eulérien* dans le graphe $G$ est un chemin qui utilise une seule fois tous les arcs d'une composante connexe. Par exemple $\langle 3, 1, 2, 3, 4, 1, 0, 4, 2 \rangle$ est un chemin eulérien dans le graphe $(b)$ de la figure 1, mais n'est pas un circuit eulérien.

**Question 6** Montrer les propositions suivantes :

a) Un graphe admettant un circuit eulérien est un graphe eulérien.

b) Un graphe eulérien est fortement connexe si et seulement s'il est connexe.

On note par $d^G_{x,y}$ un chemin eulérien dans $G$ menant de $x$ à $y$. On écrit $c^G_x$ dans le cas où $x = y$ (un cycle eulérien). Remarques : un graphe eulérien n'est pas forcément connexe, un cycle eulérien n'est pas non plus forcément un circuit eulérien.

**Question 7** Soit $e$ un arc de $x$ à $y$ dans $G$. Soit $H$ le graphe obtenu à partir de $G$ en supprimant l'arc $e$. Soit $K$ le graphe obtenu à partir de $H$ en supprimant les arcs contenus dans la composante connexe de $y$ dans $H$. Montrer que :

a) $c^G_x = x \cdot d^H_{y,x}$ est un cycle eulérien quand $G$ est eulérien et $d^H_{y,x}$ est un chemin eulérien de $H$.

b) $d^G_{x,z} = c^K_x \cdot d^H_{y,z}$ est un chemin eulérien quand $G$ vérifie $\delta(x) = -1$, $\delta(z) = 1$, $\delta(s) = 0$ pour tout autre sommet $s$ de $G$, et quand $c^K_x$ et $d^H_{y,z}$ sont des cycles et chemins eulériens de $K$ et de $H$.

**Question 8** Ecrire une fonction `euler (Graphe g, int x)` qui retourne comme résultat un circuit eulérien du graphe $G$ issu du sommet $x$. Quelle est sa complexité ?

## Partie II – Cycles de De Bruijn

On se propose maintenant de résoudre le problème suivant : on veut énumérer dans le meilleur ordre possible tous les codes d'un digicode. Les touches du digicode sont les chiffres et les lettres “A” et “B”. On considérera que le code à trouver est un mot de $n$ lettres sur un alphabet de $k$ lettres. Typiquement $k = 12$ et $n = 5$. Pour résoudre le problème, on remarque que si on a composé les $n$ numéros d'un code, il n'est pas la peine de recomposer complètement le code suivant, puisque le digicode conserve les valeurs des $n - 1$ dernières touches composées, comme indiqué sur la figure 2.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 3 | A | 2 | 4 | | | |
| | 3 | A | 2 | 4 | B | | |
| | | A | 2 | 4 | B | 5 | |
| | | | 2 | 4 | B | 5 | 1 |

FIG. 2 – Quatre tentatives successives sur le digicode ($k = 12$, $n = 5$).

Un *cycle de De Bruijn* d'ordre $n$ sur $k$ lettres est un mot de longueur $k^n$ qui contient circulairement tous les mots de longueur $n$ sur l'alphabet $\{0, 1, \ldots, k-1\}$. Un tel cycle fournit donc un moyen d'énumérer économiquement tous les mots de longueur $n$. Par exemple, pour $k = 2$ et $n = 3$, le mot $x = 00010111$ est un cycle de De Bruijn d'ordre 3. En effet, les 8 mots binaires de longueur 3 apparaissent successivement dans $x$ lu circulairement dans l'ordre $(000, 001, 010, 101, 011, 111, 110, 100)$. Nous allons utiliser les graphes eulériens pour construire des cycles de De Bruijn.

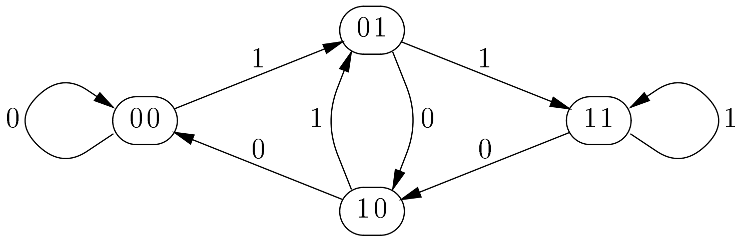



FIG. 3 – Graphe de de Bruijn d'ordre $n = 2$ pour $k = 2$.

Le *graphe de De Bruijn* d'ordre $n$ sur $k$ lettres est le graphe étiqueté noté $B(n, k)$ dont les sommets sont les mots de longueur $n$ sur l'alphabet $\{0, 1, \ldots, k-1\}$ et les arcs tous les triplets $(ax, b, xb)$ où $a, b$ sont des lettres et $x$ est un mot de longueur $n - 1$.

On supposera les graphes de De Bruijn représentés comme les graphes précédents, sauf pour la classe `Liste` où un champ supplémentaire `etiquette` détermine la lettre correspondant à cet arc.

**Question 9** Ecrire un constructeur `Graphe(int n, int k)` qui construit le graphe $B(n, k)$. On identifiera chaque sommet de $B(n, k)$ à un entier en le considérant comme écrit en base $k$.

**Question 10** Utiliser la fonction `euler` pour écrire une fonction `deBruijn(int n, int k)` qui rend un mot de De Bruijn d'ordre $n$ sur $k$ lettres.

## Partie III – Mots premiers

Un mot sur l'alphabet $A$ est dit *premier* s'il est plus petit dans l'ordre alphabétique $\prec$ que tous ses suffixes propres. Ainsi, pour $A = \{0, 1\}$, le mot $x = 00101$ est premier, mais $y = 01001$ ne l'est pas. On va voir que les mots premiers donnent un moyen de construire des cycles de De Bruijn de façon très économique.

**Question 11** Donner la liste des mots premiers de longueur 5 sur l'alphabet $A = \{0, 1\}$.

On suppose donnée la classe suivante `Mot` pour représenter les mots :

```
class Mot {
  Mot (int n) {...}                    // construit le mot formé de n zéros
  int length() {...}                   // renvoie la longueur du mot
  int charAt(int i) {...}              // renvoie l'entier à la position d'indice i
  void setCharAt(int i, int c) {...}   // affecte la position d'indice i
  public String toString() {...}       // pour l'affichage du mot
}
```

Ainsi, le mot `0101` est obtenu en effectuant les instructions suivantes :

```
Mot x = new Mot(4);
x.setCharAt(0, 0); x.setCharAt(1, 1); x.setCharAt(2, 0); x.setCharAt(3, 1);
```

**Question 12** Ecrire les deux fonctions suivantes :

a) `comparer(Mot x, int i)` qui rend $-1$ si $x \prec y$, $0$ si $x = y$ et $1$ si $x \succ y$, où $y$ désigne le suffixe de $x$ commençant au caractère d'indice $i$.

b) `estPremier(Mot x)` qui teste, en utilisant la fonction précédente, si un mot $x$ est premier. Quelle est sa complexité ?

Soient $M_{k,n}$ et $M_{k,\leq n}$, respectivement, les ensembles des mots de longueur $n$, et de longueur inférieure ou égale à $n$, sur l'alphabet $\{0, 1, \dots k-1\}$, avec $k > 1$.

**Question 13** Ecrire une fonction `longueurSuivant(int k, Mot x)` qui retourne comme résultat la longueur du mot $y \in M_{k,\leq n}$ successeur immédiat de $x \in M_{k,n}$ dans l'ordre alphabétique. Cette fonction retourne 0 si $x$ est maximal dans $M_{k,\leq n}$.

**Question 14**

a) Donner tous les mots premiers de longueur 1 de $M_{k,\leq n}$.

b) Quels sont les premier et dernier mots dans l'ordre alphabétique qui sont des mots premiers de $M_{k,n}$ $(n > 1)$.

c) Même question dans $M_{k,\leq n}$ $(n > 1)$.

Un mot *primaire* est un préfixe non vide d'un mot premier. L'*extension* $E_n(x)$ de longueur $n$ d'un mot non-vide $x$ est le préfixe de longueur $n$ des mots $x^i$ ($x$ répété $i$ fois) pour $i$ assez grand. On admet les résultats suivants :

- L'extension de longueur $n$ de tout mot premier de $M_{k,\leq n}$ est un mot primaire de $M_{k,n}$.
- Tout mot primaire de $M_{k,n}$ est l'extension d'un (unique) mot premier de $M_{k,\leq n}$.
- Soient $x$ et $y$ deux mots premiers de $M_{k,\leq n}$ tels que $x \prec y$. Alors $x \prec E_n(x) \prec y$.
- Le mot obtenu en incrémentant la dernière lettre d'un mot primaire est premier.

**Question 15**

a) Donner les premier et dernier mots primaires dans l'ordre alphabétique dans $M_{k,n}$ $(n > 1)$.

b) Donner dans l'ordre alphabétique les mots primaires dans $M_{2,\leq 4}$. Indiquer les mots premiers dans cette liste.

**Question 16**

a) Ecrire une fonction `extension (int i, Mot x)` qui modifie $x \in M_{k,n}$ en le remplaçant par l'extension de longueur $n$ de son préfixe de longueur $i$.

b) Ecrire une fonction `primaireSuivant (int k, Mot x)` qui modifie le mot primaire $x \in M_{k,n}$ en le remplaçant par le mot primaire $y$ qui suit $x$ dans l'ordre alphabétique dans $M_{k,n}$, et qui retourne la longueur du mot premier dont $y$ est l'extension, ou bien 0 si $x$ est maximal.

c) Ecrire la fonction `primaires(int k, int n)` qui imprime les mots primaires de $M_{k,n}$ dans l'ordre alphabétique.

Le mot formé par la suite des mots premiers de longueur divisant $n$ dans l'ordre alphabétique est un mot de De Bruijn. Par exemple, le mot de De Bruijn d'ordre 5 ci-dessous est obtenu ainsi.

0 00001 00011 00101 00111 01011 01111 1.

**Question 17** Utiliser cette propriété pour écrire une fonction `fM(int n, int k)` qui imprime un mot de De Bruijn de longueur $n$ sur $k$ lettres sans construire le graphe de De Bruijn.

# Corrigé

**Question 1**

```
Chemin (int x) {
  premier = new Liste (x, null);
  dernier = premier;
}

Chemin (int x, Chemin a) {
  premier = new Liste (x, a.premier);
  dernier = a.dernier;
}

Chemin (Chemin a, int x) {
  premier = a.premier;
  dernier = a.dernier.suivant = new Liste (x, null);
}
```

**Question 2**

```
static Chemin concat (Chemin a, Chemin b) {
  a.dernier.suivant = b.premier;
  a.dernier = b.dernier;
  return a;
}
```

**Question 3** Respectivement dans les classes Chemin et Liste.

```
public String toString () {
  return (premier == null) ? "" : premier.toString();
}
public String toString () {
  String r = "";
  for (Liste x = this; x != null; x = x.suivant)
    r = r + x.val + " " ;
  return r;
}
```

**Question 4** $(a)$ et $(c)$ sont eulériens.

**Question 5**

```
static int[ ] delta (Graphe g) {
  int n = g.succ.length;
  int[ ] d = new int[n];
  for (int x = 0; x < n; ++x)
    for (Liste ls = g.succ[x]; ls != null; ls = ls.suivant) {
      int y = ls.val;
      ++ d[x]; -- d[y];
    }
}

static boolean estEulerien (Graphe g) {
  int[ ] d = delta (g);
  for (int x = 0; x < d.length; ++x)
    if (d[x] != 0)
      return false;
```

```
    return true;
  }
```

**Question 6**

a) Le chemin eulérien entre et sort de chaque sommet par des arcs tous distincts. Donc $\delta(x) = 0$ pour tout sommet $x$.

b) Fortement connexe implique trivialement connexe. Supposons que le graphe eulérien soit connexe. Comme les composantes fortement connexes forment un graphe dirigé sans cycle entre elles, il existe au moins une composante connexe avec plus d'arcs entrants que sortants. Impossible. Il ne peut donc exister plus d'une composante fortement connexe.

**Question 7**

a) Montrons que tout arc $e'$ de $G$ dans la composante connexe de $x$ est parcouru par $c_x^G$. En effet, soit $e' = e$ et alors $e$ est la première étape de $c_x^G$. Soit $e' \neq e$. Alors, comme $G$ est eulérien, toute composante connexe de $G$ est aussi fortement connexe. Il existe donc un chemin de $y$ à une des extrémités de $e'$ ne passant pas par $e$. Donc $e'$ est dans la composante connexe de $y$ dans $H$, et le chemin eulérien $d_{y,x}^H$ parcourt $e'$.

b) Soit $e'$ un arc de $G$ dans la composante connexe de $x$. Soit $e' = e$, c'est l'étape entre $c_x^G$ et $d_{y,z}^H$. Soit $e' \neq e$, et une extrémité de $e'$ est dans la composante connexe de $y$ dans $G$. Deux sous-cas : premier cas si $e'$ dans la composante connexe de $y$ dans $H$. Alors $e'$ est parcouru par $d_{y,z}^H$ par récurrence dans $H$. Deuxième cas : $e'$ n'est pas accessible de $y$ dans $H$. Il est donc dans la composante connexe de $x$ dans $K$. Alors il figure donc sur $c_x^G$.

**Question 8** En $O(|E|)$ opérations où $E$ est l'ensemble des arcs de $G$, on fait :

```
  static Chemin euler (Graphe g, int s) {
    int n = g.succ.length;
    Liste[ ] succ1 = new Liste[n];
    for (int x = 0; x < n; ++x) succ1[x] = g.succ[x];
    return cycleE(succ1, s);
  }

  static Chemin cycleE (Liste[ ] succ, int x) {
    // assert δ(x) = 0
    return cheminE (succ, x, x) ;
  }

  static Chemin cheminE (Liste[ ] succ, int x, int z) {
    //assert (δ(x) = 0 ∧ x = z) ∨ (δ(x) = −1 ∧ δ(z) = 1)
    if (succ[x] == null)
      // assert x = z
      return new Chemin(x);
    else {
      int y = succ[x].val;
      succ[x] = succ[x].suivant;
      Chemin c = CheminE (succ, y, z);
      return Chemin.concat (cycleE (succ, x), c);
    }
  }
```

**Question 9**

```
  Graphe (int n, int k){
    int nsommets = 1;
    for(int i = 0; i < n; ++i)
```

```
      nsommets = nsommets * k;
    succ = new Liste[nsommets];
    for (int p = 0; p < nsommets; ++p)
      for (int a = 0; a < k; ++a)
        succ[p] = new Liste (p, (k*p+a) % nsommets, a, succ[p]);
  }
```

**Question 10**

```
  static String deBruijn (int n, int k){
    Graphe g = new Graphe (n, k);
    Chemin c = euler(g, 0);
    String s = "";
    for (Liste ls = c.premier; ls != null; ls = ls.suivant)
      s = s + ls.etiquette;
    return s;
  }
```

**Question 11** 00001 00011 00101 00111 01011 01111.

**Question 12**

a)

```
  static int comparer (Mot x, int i) {
    for (int j = 0; i+j < x.length(); ++j) {
      if (x.charAt(j) < x.charAt(i+j)) return -1;
      if (x.charAt(j) > x.charAt(i+j)) return 1;
    }
    return i == 0 ? 0 : 1;
  }
```

b) Avec une complexité en $O(n^2)$ :

```
  static boolean estPremier (Mot x) {
    for (int i = 1; i < x.length(); ++i)
      if (comparer(x, i) == 1) return false;
        return true;
  }
```

**Question 13** On doit incrémenter la lettre incrémentable la plus à droite.

```
  static int longueurSuivant(int k, Mot x) {
    int n = x.length() ;
    for (int i = n-1 ; i >= 0 ; i--)
      if (x.charAt(i) != k-1) return i+1 ;
    return 0 ;
  }
```

**Question 14**

a) Tous les mots de longueur 1 sont premiers, il s'agit donc des $k$ mots 0, 1, ..., $k-1$.

b) Pour $n \geq 2$, aucun mot premier ne peut finir par par 0, car 0 est minimal parmi les mots non-vides. Mais le second mot de la liste des mots de $M_{k,n}$ qui est $00\ldots01$ ($n-1$ fois zéro suivis de 1) est premier.

Par ailleurs, et toujours pour $n \geq 2$, aucun mot premier ne peut commencer par la lettre maximale $k-1$ (le mot formé de $n$ fois $k-1$ n'étant pas premier et tout mot commençant par $k-1$ et contenant une lettre $j < k-1$ ne l'étant pas non plus). Or, le mot $k-2k-1\ldots k-1$ ($k-2$ suivi de $n-1$ fois $k-1$) est premier et tous les mots qui le suivent ont $k-1$ en première position.

La réponse est donc $00\ldots01$ et $k-2k-1\ldots k-1$.

c) Le mot 0 est premier et minimal dans $M_{k,\leq n}$ De l'autre côté, le mot $k-1$ est premier et tous les mots à lui supérieurs dans $M_{k,\leq n}$ commencent par $k-1$, sont de longueur supérieure ou égale à deux et ne sont donc pas premiers. La réponse est 0 et $k-1$ (réponse d'ailleurs valable pour $n=1$).

**Question 15**

a) Le premier mot de $M_{k,n}$ est primaire (c'est un prefixe de $00\dots1$, premier mot premier de $M_{k,n+1}$). Le dermier mot de $M_{k,n}$ ($n$ fois $k-1$) est également primaire, car il est préfixe du mot premier $k-1k-1\dots k-1k$ de $M_{k+1,n+1}$. La réponse est donc $00\dots0$ et $k-1k-1\dots k-1$.

b) On dresse la liste des mots premiers de $M_{2,\leq 4}$ assez facilement, on l'ordonne et on insère les mots primaires.

| | | | |
|---|---|---|---|
| 0 | 0000 | 01 | 0101 |
| 0001 | | 011 | 0110 |
| 001 | 0010 | 0111 | |
| 0011 | | 1 | 1111 |

**Question 16**

a)
```
static void extension (int i, Mot x) {
  int n = x.length() ;
  for (int i = i ; j < n ; ++j)
    x.setCharAt(j, x.charAt(j-l)) ;
}
```
b)
```
static int primaireSuivant (int k, Mot x) {
  int n = x.length(), p = longueurSuivant(k, x) ;
  if (p > 0) {
    x.setCharAt(p-1, x.charAt(p-1) + 1) ;
    extension(p,x) ;
  }
  return p ;
}
```
c)
```
static void primaires (int k, int n) {
  Mot x = new Mot(n) ;
  do { System.out.println(x) ; }
  while (primaireSuivant (k, x) > 0) ;
}
```

**Question 17**
```
static void printPrefix (int p, Mot x) {
  for (int i = 0 ; i < p ; ++i)
    System.out.print(x. charAt(i)) ;
}

static void fM (int k, int n) {
  Mot x = new Mot(n) ;
  int i = 1 ;
  do {
    if (n % i == 0) printPrefix (i, x) ;
  } while ((i = primaireSuivant (k, x)) > 0) ;
}
```