Contrôle Classant – Informatique 431

Claire Kenyon Jean-Jacques Lévy *

Ecole polytechnique, 30 juin 2005

On attachera une grande importance à la concision, à la clarté, et à la précision de la rédaction. Lorsqu'une question porte sur une complexité en temps ou en espace, seul l'ordre de grandeur en fonction du paramètre précisé est demandé, dont on demande une démonstration à chaque fois.

Une entreprise de distribution ouvre plusieurs magasins dans une nouvelle région à commercialiser. Les sites de ses magasins doivent être optimisés en fonction de plusieurs facteurs : 1/ la somme des distances parcourues par les clients pour se rendre au magasin le plus proche ; 2/ le coût d'ouverture des magasins ; 3/ le maximum des distances parcourues par les clients pour se rendre au magasin le plus proche.

Pour simplifier, on suppose qu'il n'y a qu'un seul client par site et que les magasins ne s'installent que sur les sites des clients. Ainsi, si $X = \{x_0, x_1, \ldots x_{n-1}\}$ est l'ensemble des n sites clients ($n > 0$), on veut ouvrir k magasins ($k > 0$) sur des sites y_j tels que l'ensemble $Y = \{y_0, y_1, \ldots y_{k-1}\}$ vérifie $Y \subseteq X$.

Soit $d(x_i, x_j)$ la distance entre les sites x_i et x_j et $\mathit{coût}(x_i)$ le coût d'ouverture d'un magasin sur le site x_i. On cherchera d'abord à minimiser, pour tous les sous-ensembles Y de k élements dans X, la quantité $S(Y)$ suivante :

$$S(Y) = d(x_0, Y) + d(x_1, Y) + \cdots + d(x_{n-1}, Y) + \mathit{coût}(Y)$$
$$\text{avec } d(x_i, Y) = \min\{d(x_i, y_j) \mid y_j \in Y\} \text{ et } \mathit{coût}(Y) = \sum_{y_j \in Y} \mathit{coût}(y_j)$$

où $d(x_i, Y)$ est la distance minimale parcourue par le client i pour se rendre à un magasin situé dans Y, et $\mathit{coût}(Y)$ est la somme des coûts d'ouverture des magasins situés en Y.

Dans ce problème, on utilisera les classes `Paire` (dans la partie I), `Liste` et `Graphe` (dans les parties III et IV) pour définir des paires d'entiers, des listes de paires, et des graphes non-orientés représentés par un tableau de listes qui donne pour chaque sommet s, la liste des paires (t, ℓ) pour chaque arc allant de s à t de longueur ℓ.

```
class Paire {
  int un; int deux;
  Paire (int x, int y) {
    un = x; deux = y;
} }

class Liste {
  Paire val; Liste suivant;
  Liste (Paire x, Liste ls) {
    val = x; suivant = ls;
} }
```

```
class Graphe {
  Liste[ ] succ;
  Graphe (int n) { succ = new Liste[n]; }
  ...
  void ajouterArc (int s, int t, int distance) {
    succ[s] = new Liste (
        new Paire (t, distance), succ[s]);
    succ[t] = new Liste (
        new Paire (s, distance), succ[t]);
  }
}
```

Partie I. Problème en une dimension

Tout se passe en une dimension, et on suppose qu'il faut ouvrir un unique magasin ($k = 1$). Un site client x_i est repéré par son abcisse entière naturelle x_i pour tout i ($0 \le i < n$), et on pose $0 \le x_i < x_j$ pour $0 \le i < j < n$ (On confond, sans ambiguité, site x_i et son abscisse x_i). Dans un premier temps,

*avec les participations de Robert Cori et Frédéric Magniez on suppose que le coût d'ouverture d'un magasin est le même sur tous les sites. Placer le magasin sur le site x_m est un choix optimal si et seulement s'il minimise la quantité $S(m)$ suivante :

$$S(m) = \sum_{i=0}^{n-1} |x_i - x_m|$$

Question 1 Calculer x_m optimal quand $X = \{0, 1, 2, 4, 7, 9\}$.

Question 2 Démontrer que ce minimum est atteint si et seulement si m est en position médiane, c'est-à-dire lorsque le nombre p de x_i inférieurs à x_m et le nombre q de x_i supérieurs à x_m vérifient $|p - q| \leq 1$.

A chaque configuration optimale, on associe un mot sur l'alphabet $\{\mathtt{c}, \mathtt{r}, \mathtt{m}\}$, obtenu en lisant la succession des points à coordonnées entières sur la ligne, à partir du premier client et jusqu'au dernier client. La lettre `c` signifie « client », `m` signifie « magasin »(et un client se trouve également sur ce site), `r` signifie « rien ». Dans l'exemple précédent, un mot associé à $x_m = 4$ est `cccrmrrcrc`.

Question 3 Donner le mot associé à $x_m = 2$ dans cet exemple.

Question 4 Donner une grammaire algébrique pour engendrer le langage des configurations optimales.

Question 5 Montrer que ce langage n'est pas rationnel (régulier).

On suppose à présent que le coût d'ouverture d'un magasin varie selon le site, et on souhaite choisir le lieu du magasin de façon à minimiser la somme du coût d'ouverture du magasin et de la distance parcourue par les clients. On veut donc minimiser la quantité :

$$S(m) = \sum_{i=0}^{n-1} |x_i - x_m| + co\hat{u}t(x_m)$$

Remarque : le minimum n'est plus forcément atteint quand m est en position médiane.

Informatiquement, les sites clients sont représentés par un tableau s de n paires $s[i] = (x_i, c_i)$, trié dans l'ordre croissant des x_i, tel que, pour chaque site, la position est en x_i et le coût c_i d'ouverture d'un magasin sur ce site est $co\hat{u}t(x_i)$.

Question 6 Ecrire la fonction `magasin1`(s) qui prend en argument un tableau de sites s ; et qui retourne, en temps linéaire par rapport à n, l'abcisse x_m d'un emplacement optimal pour l'ouverture d'un magasin.

Partie II. Allocation de clients à deux magasins de capacité limitée

Dans cette partie, les plus courtes distances entre les sites x_i et x_j sont données par une matrice symétrique (d_{ij}) représentée par un tableau d vérifiant $d[i][j] = d_{ij}$ pour tout i, j tels que $0 \leq i < n$ et $0 \leq j < n$. En outre, il n'y a que deux magasins distincts m et m' dont les emplacements x_m et $x_{m'}$ sont *donnés*. Ces deux magasins ont une capacité limitée à $n/2$, chaque magasin ne pouvant servir qu'un nombre limité de clients, $n/2$ au maximum. Pour simplifier, on supposera que n est pair.

On cherche à allouer les clients aux deux magasins en minimisant la somme des distances parcourues par les clients pour se rendre à un magasin, tout en respectant les contraintes de capacité.

Question 7 Montrer qu'une allocation est optimale si et seulement s'il n'existe pas deux clients qui pourraient améliorer la valeur de l'allocation en s'échangeant leurs magasins respectifs.

Dans une classe `Tableau`, on suppose donnée la méthode `trier`(a, c) qui trie, en temps $O(n \log n)$, un tableau a de n objets selon la méthode `plusPetitQue` du comparateur c ordonnant deux objets de a.

```
interface Comparateur {
  boolean plusPetitQue
    (Object x, Object y);
}
```

```
class Tableau {
  static void trier
    (Object[ ] a, Comparateur c) {...}
}
```

Question 8 Définir une classe `DeuxMagasins` et sa méthode statique `allocation`(d, m, m') qui prend en arguments le tableau d des distances, les deux magasins m et m' ; et qui retourne un tableau de n valeurs valant m ou m' selon que le client situé sur le site x_i se rend en x_m ou en $x_{m'}$, pour minimiser la somme des distances parcourues par les clients tout en respectant les contraintes de capacités des magasins. Donner sa complexité en temps, par rapport à n.

Partie III. Cas général : placement optimal des magasins

Les sites clients constituent à présent les n sommets d'un graphe G non-orienté dont les arcs sont valués par leurs longueurs. Nous appelons *matrice d'adjacence valuée* la matrice (m_{ij}) telle que $m_{ij} = \ell$ s'il existe, dans G, un arc de longueur ℓ entre le sommet i et le sommet j, et $m_{ij} = +\infty$ s'il n'existe pas d'arc entre i et j.

Question 9 Écrire la méthode statique `matriceAdjacence`(G) qui prend en argument le graphe G ; et qui retourne la matrice d'adjacence valuée du graphe G. Donner la complexité en temps de cette fonction.

Question 10 Écrire la méthode statique `plusCourtsChemins`(G) qui prend en argument le graphe G ; et qui retourne la matrice $n \times n$ des distances des plus courts chemins entre toute paire de sommets. Donner la complexité en temps de cette fonction.

Le problème général de trouver le placement optimal pour les k magasins sur le graphe G est NP-complet. Pour trouver le meilleur placement, on se contente donc de procéder par énumération sur tous les placements possibles des k magasins sur les sommets du graphe G. Cela revient à énumérer tous les sous-ensembles de taille k de l'ensemble $\{0, 1, \ldots n-1\}$. On codera un tel sous-ensemble par un k-uplet $a = \langle a_0, a_1, \ldots a_{k-1} \rangle$ tels qu'on ait $a_0 < a_1 < \cdots a_{k-1}$ et $0 \leq a_i < n$ pour $0 \leq i < n$.

On procède en deux étapes : 1/ définir une classe abstraite générale `Enumeration` paramétrée par une méthode abstraite `action`, et écrire, dans cette classe, une méthode `iterer` qui effectue l'action `action`(a) pour tout k-uplet a ; 2/ considérer une sous-classe de `Enumeration` qui effectuera les calculs nécessaires pour trouver le placement optimal.

```
abstract class Enumeration {
  int[ ] a;
  Enumeration (int k) { a = new int[k]; }
  abstract void action (int[ ] a);
  ...
}
```

Question 11 Écrire la méthode `iterer`(k, n) qui appelle la fonction `action`(a) pour tous les k-uplets a de combinaisons de k nombres parmi n. Quelle est sa complexité en temps ?

Question 12 Écrire une méthode statique `imprimerCombinaisons`(n, k) qui imprime (sans répétitions) toutes les combinaisons de k nombres parmi $\{0, 1, \ldots n-1\}$.

On passe maintenant à la deuxième étape. On suppose qu'un tableau c donne le coût d'ouverture $c[i]$ du magasin situé au sommet i $(0 \leq i < n)$.

Question 13 Écrire une fonction `valeurDe`(d, c, a) qui prend en arguments le tableau des distances d, le tableau c de coûts d'ouverture, et un k-uplet a ; et qui retourne la somme $S(Y)$ qui correspond au placement a.

Question 14 En déduire une méthode statique `magasins`(d, c, k) qui prend en argument le tableau des distances d, le tableau c de coûts d'ouverture, et le nombre k de magasins ; et qui retourne le placement optimal $\langle a_0, a_1, \ldots a_{k-1} \rangle$. Quelle est sa complexité en temps ?

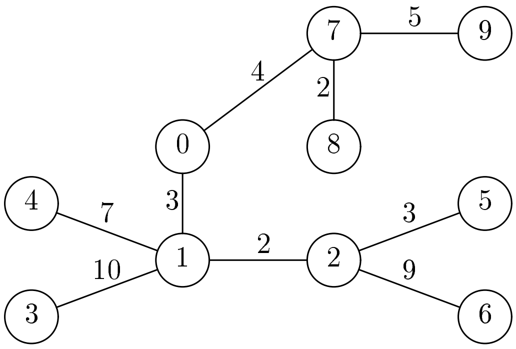
Partie IV. Placement d'un magasin dans un arbre non-enraciné

Dans cette partie, on suppose qu'il y a un seul magasin à placer ($k = 1$). Au lieu d'une entreprise de distribution, on envisage une application de type « construction d'une caserne de pompiers » : le but est de pouvoir rapidement se rendre partout. Ainsi, le placement de la caserne doit maintenant être choisi de façon à minimiser la quantité

$$M(Y) = \max\{d(x_i, Y) \mid x_i \in X\}$$

On appelle *centre* d'un graphe G un sommet y qui minimise $M(\{y\})$.

Pour simplifier, on suppose que le graphe non-orienté G a une structure d'*arbre non-enraciné* : un graphe G est un arbre non-enraciné si, entre toute paire de sommets distincts, il existe un et un seul chemin simple (c'est-à-dire ne passant pas deux fois par le même sommet), comme sur la figure suivante :



On remarque que, dans un arbre non-enraciné, le nombre d'arcs (non-orientés) est égal au nombre de sommets moins un.

Sur la figure, le centre est le sommet 1 dont la distance à tous les autres sommets n'excède pas 12.

Question 15 Donner un algorithme pour trouver, en temps linéaire sur le nombre de sommets n de G, le centre d'un arbre non-enraciné G.
Indications : 1/ on pourra faire deux passes sur le graphe G pour calculer la distance maximale de tout sommet à une feuille ; 2/ on suppose qu'on sait calculer b à partir de a de longueur ℓ, en temps $O(\ell)$, où b est défini par :

$$b[i] = \max\{a[j] \mid j \neq i, 0 \leq j < \ell\} \qquad (0 \leq i < \ell)$$

Question 16 Écrire la méthode statique `centre`(G) qui prend en argument un arbre non-enraciné G de n sommets ; et qui retourne, en temps linéaire par rapport à n, un centre de G.
Indication : on peut utiliser la fonction suivante correspondant à la 2ème indication de la question précédente.

```
static int[ ] maxExterne (int[ ] a) {
  int ell = a.length; int[ ] b = new int[ell];
  int max = 0, max2 = 0;
  for (int i = 0; i < ell; ++i)
    if (max <= a[i]) {
      max2 = max; max = a[i];
    } else if (max2 < a[i])
      max2 = a[i];
  for (int i = 0; i < ell; ++i)
    if (a[i] == max) b[i] = max2;
    else b[i] = max;
  return b;
}
```

Corrigé du contrôle classant – INF 431

Question 1 2 et 4, car $S(0) = 1+2+4+7+9 = 23$, $S(1) = 1+1+3+6+9 = 20$, $S(2) = 2+1+2+5+7 = 17$, $S(3) = 4+3+2+3+5 = 17$, $S(5) = 7+6+5+3+2 = 23$, $S(6) = 9+8+7+5+2 = 31$.

Question 2

$$
\begin{aligned}
& S(i) - S(i-1) \\
&= \textstyle\sum_{j=0}^{i-1}(x_i - x_j) + \sum_{j=i+1}^{n-1}(x_j - x_i) - \sum_{j=0}^{i-2}(x_{i-1} - x_j) - \sum_{j=i}^{n-1}(x_j - x_{i-1}) \\
&= (x_i - x_{i-1}) + (i-1)(x_i - x_{i-1}) - (n-i-1)(x_{i-1} - x_i) - (x_i - x_{i-1}) \\
&= (2i-n)(x_i - x_{i-1})
\end{aligned}
$$

Donc $S(i-1) > S(i)$ si $2i < n$; et $S(i-1) = S(i)$ si $2i = n$; et $S(i-1) < S(i)$ si $2i > n$.

Question 3 `ccmrcrrcrc`.

Question 4 $S \to CSD \mid \mathtt{m} \mid C\mathtt{m} \mid \mathtt{m}D, \quad C \to \mathtt{c}R, \quad D \to R\mathtt{c}, \quad R \to \mathtt{r}R \mid \epsilon.$

Question 5 On considère le morphisme ϕ tel que $\phi(r) = \epsilon$. Alors $\phi(L) = \{\mathtt{c}^p\mathtt{mc}^q \mid -1 \leq p - q \leq 1\}$. Soit R l'ensemble des mots sur $\{\mathtt{c}, \mathtt{m}\}$ de longueur impaire. Si L est rationnel, alors $\phi(L)$ l'est aussi car image par morphisme. Et comme R est rationnel, on a $L \cap R$ rationnel, puisque les langages rationnels sont clos par intersection. Mais $L \cap R = \{\mathtt{c}^i\mathtt{mc}^i \mid i \geq 0\}$. On a vu dans le cours que ce langage ne pouvait être rationnel (lemme de l'étoile). Donc L ne peut être rationnel.

Question 6 Solution en $O(n)$ puisqu'on ne fait qu'un seul parcours de s.

```
static int magasin1 (Paire[ ] s) {
  int n = s.length; int v = s[0].deux;
  for (int i=1; i < n; ++i)
    v = v + s[i].un - s[0].un;
  int iMin = 0; int min = v;
  for (int i=1; i < n; ++i) {
    v = v + (2*i - n) * (s[i].deux - s[i-1].deux) +
      (s[i].un - s[i-1].un);
    if (v < min) { min = v; iMin = i; }
  }
  return s[iMin].un;
}
```

Question 7 Le seul sens difficile consiste à montrer que si une allocation n'est pas optimale, alors il existe un échange de deux clients qui améliore $S(Y)$. Si une allocation n'est pas optimale, cela signifie qu'on peut changer l'allocation d'un sous-ensemble de X entre x_m et x'_m. Autrement dit, pour $Z = \{x_{i_1}, \ldots x_{i_\ell}\}$ et $Z' = \{x'_{i_1}, \ldots x'_{i_\ell}\}$, on a :

$$\sum_{x_i \in Z} d(x_i, x_m) + \sum_{x'_i \in Z'} d(x'_i, x_{m'}) > \sum_{x_i \in Z} d(x_i, x_{m'}) + \sum_{x'_i \in Z'} d(x'_i, x_m)$$

En posant $\delta(x) = d(x, x_m) - d(x, x_{m'})$, cela revient à dire

$$\mathcal{E} = \sum_{x_i \in Z} \delta(x_i) - \sum_{x'_i \in Z'} \delta(x'_i) > 0$$

Or un échange entre x et x' baisse la valeur de $S(Y)$ ssi $\delta(x) > \delta(x')$. Donc s'il n'existe aucun échange possible entre X et X' pour diminuer $S(Y)$, on a $\mathcal{E} \leq 0$ et une contradiction.

Question 8

```
class DeuxMagasins implements Comparateur {
  Paire[ ] delta;

  DeuxMagasins (int[ ][ ] d, int m1, int m2) {
    int n = d[0].length;
    Paire[ ] delta = new Paire[n];
    for (int i=0; i < n; ++i) {
      delta[i].un = d[i][m1] - d[i][m2];
      delta[i].deux = i;
    }
  }

  public boolean plusPetitQue (Object x, Object y) {
    return ((Paire)x).un <= ((Paire)y).un;
  }

  static int[ ] allocation (int[ ][ ] d, int m1, int m2) {
    DeuxMagasins mm = new DeuxMagasins (d, m1, m2);
    Tableau.trier (mm.delta, mm);
    int n = d[0].length;
    int[ ] r = new int[n];
    for (int i=0; i < n/2; ++i) r[mm.delta[i].deux] = m1;
    for (int i=n/2; i < n; ++i) r[mm.delta[i].deux] = m2;
    return r;
  }
}
```

Question 9 La construction de la matrice parcourt les sommets et tous les arcs en $O(n + E)$, donc $O(n^2)$ avec l'initialisation de d.

```
  static int[ ][ ] matriceAdjacence (Graphe g) {
    int n = g.succ.length;
    int[ ][ ] d = new int[n][n];
    for (int x = 0; x < n; ++x)
      for (int y = 0; y < n; ++y)
        d[x][y] = Integer.MAX_VALUE;
    for (int x = 0; x < n; ++x)
      for (Liste ls = g.succ[x]; ls != null; ls = ls.suivant) {
          int y = ls.val.un; int dxy = ls.val.deux;
          d[x][y] = dxy;
      }
    return d;
  }
```

Question 10 Algorithme de Floyd-Warshall en $O(n^3)$ vu dans le cours.

```
  static int[ ][ ] plusCourtsChemins (Graphe g) {
    int n = g.succ.length;
    int[ ][ ] d = matriceAdjacence (g);
    for (int k = 0; k < n; ++k)
      for (int x = 0; x < n; ++x)
        for (int y = 0; y < n; ++y)
          d[x][y] = Math.min (d[x][y], d[x][k] + d[k][y]);
    return d;
  }
```

Question 11

```
  void iterer (int k, int n) { iterer1 (k, 0, n); }
```

```
  void iterer1 (int k, int i, int n) {
    if (k == 0) action (a);
    else {
      int k0 = a.length;
      for (int j = i; j < n; ++j) {
        a[k0-k] = j;
        iterer1 (k-1, j+1, n);
  } } }
```

Question 12

```
class Impression extends Enumeration {
  Impression (int k) { super(k); }

  void action (int[ ] a) {
    for (int i = 0; i < a.length; ++i)
      System.out.print (a[i] + " ");
    System.out.println();
} }

  static void imprimer (int n, int k) {
    Impression imp = new Impression(k);
    imp.iterer(k, n);
  }
```

Question 13

```
  static int valeurDe (int[ ][ ] d, int[ ] c, int[ ] a) {
    int n = d[0].length, k = a.length;
    int v = 0;
    for (int j = 0; j < k; ++j) v = v + c[j];
    for (int i = 0; i < n; ++i)
      v = v + distMagasinLePlusProche (d, i, a);
    return v;
  }

  static int distMagasinLePlusProche (int[ ][ ] d, int i, int[ ] a) {
    int k = a.length;
    int v = Integer.MAX_VALUE;
    for (int j = 0; j < k; ++j)
      v = Math.min (v, d[i][a[j]]);
    return v;
  }
```

Question 14

```
class Affectation extends Iteration {
  int vMin = Integer.MAX_VALUE; int[ ] r; int[ ][ ] d;
  Affectation (int k, int[ ][ ] d0) {
    super(k); r = new int[k]; d = d0;
  }

  void action (int[ ] a) {
    int v = valeurDe (d, c, a);
    if (v < vMin)
      for (int i = 0; i < a.length; ++i)
        r[i] = a[i];
  }
}

  static int[ ] magasins (int[ ][ ] d, int[ ] c, int k) {
```

```
    int n = d[0].length;
    Affectation f = new Affectation(k, d, c);
    f.iterer(k, n);
    return f.r;
  }
```

Question 15 Dans une première passe, on fait un parcours *dfs* produisant un arbre de recouvrement A en calculant les hauteurs internes (valuées) $h(x)$ des noeuds x de A, c'est-à-dire les plus longues distances des chemins à des feuilles en ne passant que par des noeuds internes au sous-arbre de racine x.

$$h(x) = \max\{d(x, y) + h(y) \mid y \text{ sous-arbre de } x \text{ dans } A\}$$

Dans une deuxième passe, en faisant le même parcours *dfs*, on calcule les hauteurs externes des noeuds y, c'est-à-dire les plus longues distances des chemins à des feuilles en ne passant que par des sommets extérieurs au sous-arbre de racine y.

$$he(y) = d(y, x) + \max(\{he(x)\} \cup \{d(x, z) + h(z) \mid z \text{ frère de } y \text{ dans } A\})$$

où x est le père de y. Le plus long chemin issu de x est donc $\max\{h(x), he(x)\}$. Le centre de G est obtenu en cherchant le minimum de cette valeur sur tous les sommets de G.

Question 16

```
  final static int BLANC = 0, GRIS = 1, NOIR = 2;

  static int[ ] hauteur (Graphe g) {
    int n = g.succ.length; int[ ] couleur = new int[n];
    int[ ] r = new int[n];
    for (int x = 0; x < n; ++x) couleur[x] = BLANC;
    hauteur1(g, r, 0, couleur);
    return r;
  }

  static void hauteur1 (Graphe g, int[ ] r, int x, int[ ] couleur) {
    couleur[x] = GRIS; int h = 0;
    for (Liste ls = g.succ[x]; ls != null; ls = ls.suivant) {
      int y = ls.val.un; int dxy = ls.val.deux;
      if (couleur[y] == BLANC) {
        hauteur1(g, r, y, couleur);
        h = Math.max(h, dxy + r[y]);
      }
    }
    r[x] = h;
  }

  static int[ ] exterieur (Graphe g, int[ ] h) {
    int n = g.succ.length; int[ ] couleur = new int[n];
    int[ ] r = new int[n];
    for (int x = 0; x < n; ++x) couleur[x] = BLANC;
    exterieur1(g, h, r, 0, couleur);
    return r;
  }

  static void exterieur1 (Graphe g, int[ ] h, int[ ] r, int x, int[ ] couleur) {
    couleur[x] = GRIS;
    int ell = Liste.longueur(g.succ[x]);
    int[ ] a = new int[ell+1];
    a[0] = r[x]; int i = 1;
    for (Liste ls = g.succ[x]; ls != null; ls = ls.suivant) {
```

```
    int y = ls.val.un; int dxy = ls.val.deux;
    a[i++] = dxy + h[y];
  }
  int[ ] b = maxExterne(a); int j = 1;
  for (Liste ls = g.succ[x]; ls != null; ls = ls.suivant) {
    int y = ls.val.un; int dxy = ls.val.deux;
    if (couleur[y] == BLANC) {
      r[y] = dxy + b[j++];
      exterieur1 (g, h, r, y, couleur);
    }
  }
}

static int centre (Graphe g) {
  int n = g.succ.length;
  int[ ] h = hauteur(g), he = exterieur(g, h);
  int m = Integer.MAX_VALUE; int iMin = -1;
  for (int x = 0; x < n; ++x)
    if (Math.max (h[x], he[x]) < m) {
      iMin = x; m = Math.max(h[x], he[x]);
    }
  return iMin;
}
```