INF421-a
Bases de la programmation et de l'algorithmique

(Bloc 4/ 9)

Philippe Baptiste

CNRS LIX, École Polytechnique

15 septembre 2006

Aujourd'hui

Révisions de Java

Les graphes

Les arbres

Parcours DFS

fonctions et méthodes

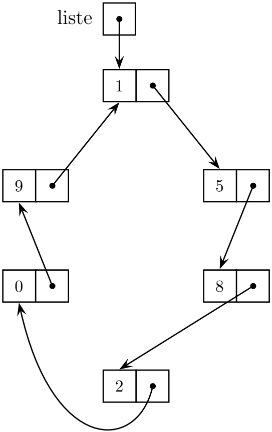
- Une méthode `static` (ou fonction) peut être utilisée sans référence à un objet particulier, *i.e.*, c'est une méthode de classe
- Une méth. statique n'a pas accès aux variables non statiques
- On peut appeler une méthode statique par `NomdeClasse.nomDeMethode()`, *e.g.*, `Math.abs()`
- Une méthode non `static` (ou dynamique) est toujours utilisée en référence à un objet . C'est une méthode d'objet .
- On ne peut jamais appliquer une méthode a "null"

this

- Dans une classe il est possible de faire référence à l'objet courant en utilisant le mot clef this .
- Ceci est particulièrement utile dans les méthodes dynamiques
- On ne peut jamais changer la valeur de this :
  NON !!! `this = this.suivant ;` NON !!!

this et les listes circulaires

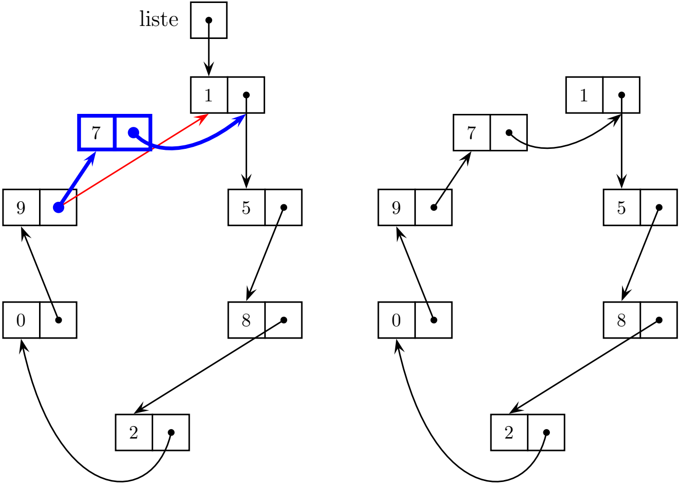
Dans une liste circulaire le champ `suivant` de la dernière cellule contient la référence de la première cellule



- Liste vide : `null`
- `Singleton` : ? ? ?

Listes circulaires

L'insertion d'un élément.



Listes circulaires

```
class ListeCirculaire {
    int val; ListeCirculaire suivant;
    ListeCirculaire (int v, ListeCirculaire l) {
        val = v;
        if (l == null)
            suivant = this;
        else {
            suivant = l;
            dernier(l).suivant = this; // CHER !!!
        }
    }
    static ListeCirculaire dernier(ListeCirculaire l) {
        if (l == null) return null;
        ListeCirculaire marque = l;
        while (l.suivant ≠ marque) l = l.suivant;
        return l;
    }
}
```

Listes circulaires

Un constructeur plus rusé

```
ListeCirculaire (int v, ListeCirculaire l) {
    val = v;
    if (l == null)
        suivant = this;
    else {
        ListeCirculaire precedent = l;
        l = l.suivant;
        suivant = l;
        precedent.suivant = this;
    }
}
```

Pourquoi ?

Affichage

```
static void affiche(ListeCirculaire l) {
    if (l == null) return;
    ListeCirculaire marque = l;
    while (l.suivant ≠ marque) {
        System.out.print(l.val + " ");
        l = l.suivant;
    }
    System.out.println(l.val);
}
```

Fusion de listes circulaires

Un constructeur plus rusé

```
ListeCirculaire (int v, ListeCirculaire l) {
    val = v;
    if (l == null)
        suivant = this;
    else {
        ListeCirculaire precedent = l;
        l = l.suivant;
        suivant = l;
        precedent.suivant = this;
    }
}
```

Pourquoi ?

Fusion de listes circulaires : exemple

Que fait le code suivant ?

```
ListeCirculaire l = null;
l = new ListeCirculaire(10, l);  l = new ListeCirculaire(11, l);
l = new ListeCirculaire(12, l);  l = new ListeCirculaire(13, l);
l = new ListeCirculaire(14, l);  l = new ListeCirculaire(15, l);
ListeCirculaire.affiche(l);
ListeCirculaire L = null;
L = new ListeCirculaire(1, L); L = new ListeCirculaire(2, L);
L = new ListeCirculaire(3, L); L = new ListeCirculaire(4, L);
L = new ListeCirculaire(5, L); L = new ListeCirculaire(6, L);
ListeCirculaire.affiche(L);
L = ListeCirculaire.fusion(l, L);
ListeCirculaire.affiche(L);
```

```
15 10 11 12 13 14
6 1 2 3 4 5
10 11 12 13 14 15 1 2 3 4 5 6
```

Aujourd'hui

- Révisions de Java
- Les graphes
- Les arbres
- Parcours DFS

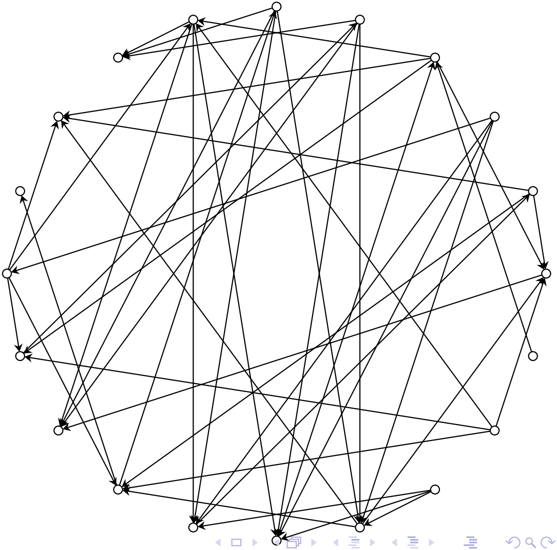
Les graphes

Un graphe orienté (digraph) $G = (S, A)$ est un couple formé

- d'un ensemble de nœuds (ou sommets) S
- et d'un ensemble $A \subseteq S \times S$ d'arcs

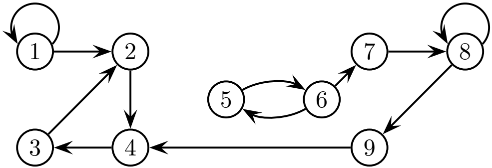
Représentation

- nœud $\leftrightarrow$ point
- arc (x, y) $\leftrightarrow$ ligne orientée de x à y



Chemins

- Un chemin de s à t est une suite $(s = s_0, \ldots, s_n = t)$ de nœuds reliés par des arcs, *i.e.*, $\forall i \in \{1, ..., n\}, (s_{i-1}, s_i) \in A$
- s_0 est l'origine du chemin et le nœud s_n son extrémité
- n est la longueur du chemin.
- Un circuit est un chemin de longueur non nulle dont l'origine coïncide avec l'extrémité
- Un chemin est simple si tous les nœuds sont distincts
- Un graphe est fortement connexe ssi $\exists$ un chemin entre toute paire de nœuds

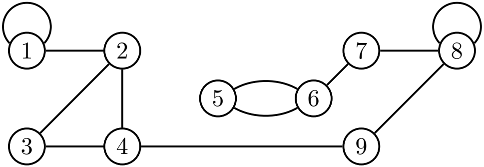


ch sp : (1, 2, 4, 3)
ch : (1, 1, 2, 4, 3)
circuit : (5, 6, 5)

Graphe non orienté

Un graphe non orienté $G = (S, A)$ est un couple formé

- d'un ensemble de nœuds S
- et d'un ensemble de paires A de sommets appelées arêtes



Est-il connexe ?

Le degré d'un nœud est le nombre d'arêtes dont ce sommet est un extrémité.

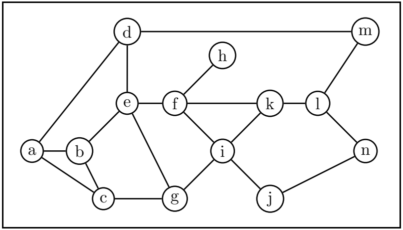
Pourquoi les graphes

- Un outil de modélisation très riche
  - Cheminement (*e.g.*mapquest)
  - Calcul de tournées de véhicules
  - Chip design (Very Large Scale Integration)
  - Ordonnancement de tâches
    - 1 tâche = 1 sommet
    - 1 précédence = 1 arc
- Un outil mathématique
  - Euler, Hamilton, Kirchhoff, Edmonds, Berge, Lovász, Seymour...
  - Les ponts de Königsberg, les 4 couleurs, ...
- Un outil fondamental de l'informatique

Quelques problèmes de théorie des graphes : Euler

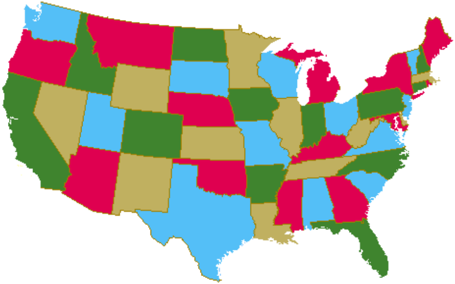
- G est planaire s'il est représentable sur un plan et
  - les sommets = des points distincts
  - les arêtes = des courbes simples qui ne s'intersectent pas
- Une face est une région du plan limitée par les arêtes (y compris la face infinie).

Formule d'Euler : Si n est le nombre de sommets, m le nombre d'arêtes et f le nombre de faces alors, si G est connexe, $n - m + f = 2$



$n = 14,\ m = 20,\ f = 8$

Quelques problèmes de théorie des graphes : 4 couleurs



©Robin Thomas

- Une carte = un graphe planaire (sommets = régions, arêtes = paires de régions voisines)
- Colorer les régions tq, toutes les régions voisines aient des couleurs différentes
- Toujours possible avec 4 couleurs Appel et Haken (76) ; Robertson, Seymour, Thomas et Sanders ; Gonthier et Werner.

Aujourd'hui

Révisions de Java

Les graphes

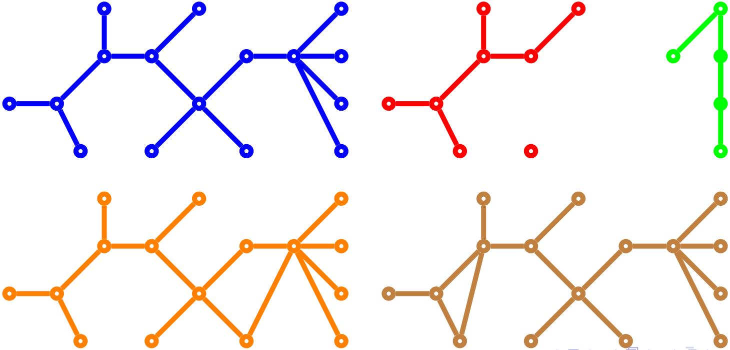
Les arbres

Parcours DFS

Définition des arbres

Un arbre (libre) est un graphe non vide connexe sans circuit.
Une forêt est un ensemble d'arbres.

Quels sont les arbres ?



Quelques propriétés élémentaires

$G = (S, A)$ un graphe non vide. Les conditions suivantes sont équivalentes :

1. G est un arbre libre,
2. Deux nœuds quelconques de S sont connectés par un chemin simple unique,
3. G est connexe, mais ne l'est plus si l'on retire une arête quelconque,
4. G est sans circuit, mais ne l'est plus si l'on ajoute une arête quelconque,
5. G est connexe, et $|A| = |S| - 1$,
6. G est sans circuit, et $|A| = |S| - 1$.

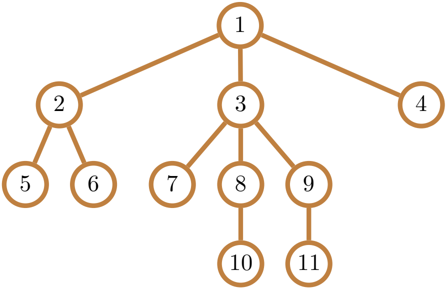
Arbre enraciné : un peu de généalogie

Un arbre enraciné arbre = arbre libre muni d'un nœud distingué, appelé sa racine

Soit T un arbre de racine r.

- Pour tout nœud x, il existe un chemin simple unique de r à x.
- Tout nœud y sur ce chemin est un ancêtre de x, et x est un descendant de y.
- Le sous-arbre de racine x est l'arbre contenant tous les descendants de x.
- L'avant-dernier nœud y sur l'unique chemin reliant r à x est le père de x, et x est un fils de y.
- L'arité d'un nœud est le nombre de ses fils.
- Un nœud sans enfant est une feuille
- La hauteur = long. max d'un chemin de r à une feuille + 1.

Arbre enraciné : un peu de généalogie

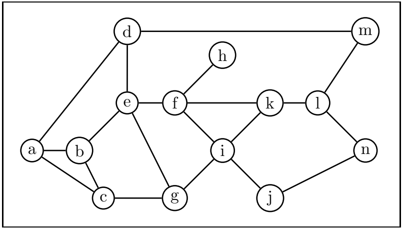


- Racine =
- Descendants de 3 = { }
- Le ss-arbre de racine 3 =
- Le père de 8 =
- Le fils de 9 =
- Les feuilles = { }
- Hauteur de l'arbre =
- Arité de 2 = Degré =

Retour sur Euler

- G est planaire s'il est représentable sur un plan et
  - les sommets = des points distincts
  - les arêtes = des courbes simples qui ne s'intersectent pas
- Une face est une région du plan limitée par les arêtes (y compris la face infinie).

Formule d'Euler : Si n est le nombre de sommets, m le nombre d'arêtes et f le nombre de faces alors, si G est connexe, $n - m + f = 2$



$n = 14,\ m = 20,\ f = 8$

Retour sur Euler

- Partons d'une représentation planaire de $G = (V, E)$.
- Enlevons suffisement d'arêtes de G pour qu'on obtienne un arbre T.
- Pour un arbre : $f = 1$ et $m = n - 1$. Soit donc : $n - m + f = 2$
- On repart de T et on ajoute les arêtes de G - T (une par une)
- En ajoutant une arête, on augmente m et f de ? ? ?

Arbre enraciné $\longrightarrow$ arbres binaires

Def récursive des arbres : Arbre = couple formé de sa racine et d'un ensemble d'arbres. C'est encore un peu compliqué. Regardons pour le moment les arbres binaires.

> Un arbre binaire sur un ensemble fini est soit vide, soit l'union disjointe d'un nœud appelé sa racine, d'un arbre binaire appelé sous-arbre gauche, et d'un arbre binaire appelé sous-arbre droit

Représentation naturelle des arbres binaires : $A = (A_g, r, A_d)$.

```
class Arbre {
    int val; Arbre gauche, droite;
    Arbre (Arbre gauche, int val, Arbre droite) {
        this.gauche = gauche;
        this.val = val;
        this.droite = droite; }}
```

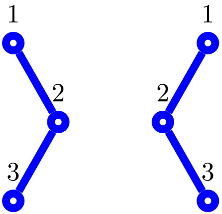
Construire un arbre

```
Arbre a = new Arbre(new Arbre(null, 6, null),
                    1,
                    new Arbre(new Arbre(null,
                                        2,
                                        new Arbre (null, 1, null)),
                              7,
                              new Arbre(null, 9, null)));
```

Dessiner l'arbre correspondant.

Arbres binaires et arbres d'arité 2 au plus

- Un arbre *ordonné* est un arbre dans lequel l'ensemble des fils de chaque nœud est totalement ordonné.
- Un arbre binaire n'est pas simplement un arbre ordonné dont tous les nœuds sont d'arité au plus 2.



Ecrire un arbre

```
public String toString() {
    String sg = "Vide"; String sd = "Vide";
    if (gauche ≠ null) sg = gauche.toString();
    if (droite ≠ null) sd = droite.toString();
    return "(" + sg + ", " + val + ", " + sd + ")"; }
```

Attention : méthode dynamique récursive.

```
Arbre a = new Arbre(new Arbre(null, 6, null),
                    1,
                    new Arbre(new Arbre(null,
                                        2,
                                        new Arbre (null, 1, null)),
                              7,
                              new Arbre(null, 9, null)));
System.out.println(a);
/* ((Vide, 6, Vide), 1, ((Vide, 2, (Vide, 1, Vide)), 7, (Vide, 9, Vide)
```

Hauteur d'un arbre

Rappel : hauteur = long. max d'un chemin de la racine à une feuille + 1

```
/* La méthode statique */
static int hauteur(Arbre a) {
    if (a == null)
        return 0;
    return 1 + Math.max(hauteur(a.gauche), hauteur(a.droite)); }
/* Et la méthode dynamique (condensée)
   Rappel : int x = (condition) ? 4 : 9; est équivalent à
   int x = 9; if (condition) x = 4; */
int hauteur() {
    return 1 + Math.max((gauche == null) ? 0 : gauche.hauteur(),
                        (droite == null) ? 0 : droite.hauteur()); }
```

Les feuilles d'un arbre

```
/* La méthode statique */
static String feuilles(Arbre a) {
    if (a == null) return "";
    if (a.gauche == null && a.droite == null) return a.val + " ";
    return feuilles(a.gauche) + feuilles(a.droite);
}
/* Et la méthode dynamique */
String feuilles() {
    if (gauche == null && droite == null) return val + " ";
    if (gauche == null)
        return droite.feuilles();
    if (droite == null)
        return gauche.feuilles();
    return gauche.feuilles() + droite.feuilles();
}
```

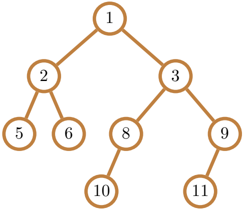
Égalité de deux arbres binaires

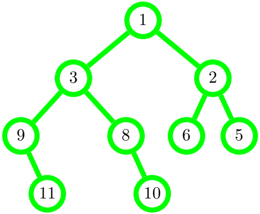
```
// Une version statique
static boolean egal(Arbre x, Arbre y) {
    return ((x == null && y == null) ||
            (x ≠ null && y ≠ null && x.val == y. val &&
             egal(x.gauche, y.gauche) && egal(x.droite, y.droite)));
}
```

Egalité de deux arbres binaires

```
// Une version dynamique
boolean egal(Arbre x) {
    if (x == null || x.val ≠ val ||
        (gauche == null && x.gauche ≠ null) ||
        (droite == null && x.droite ≠ null))
        return false;
    if (gauche == null && droite == null)
        return true;
    if (gauche == null)
        return droite.egal(x.droite);
    if (droite == null)
        return gauche.egal(x.gauche);
    return droite.egal(x.droite) && gauche.egal(x.gauche);
}
```

Inverser un arbre binaire



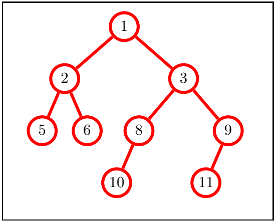



```
static Arbre inverser(Arbre x) {
   if (x == null)
      return null;
   return new Arbre(inverser(x.droite), x.val, inverser(x.gauche));
}
// Arbre = ((Vide, 6, Vide), 1, ((Vide, 2, (Vide, 1, Vide)), 7, (Vide,
// Inv = (((Vide, 9, Vide), 7, ((Vide, 1, Vide), 2, Vide)), 1, (Vide, 6
```

Codage binaire d'un arbre binaire

Etant donné un arbre binaire

- Le chemin de la racine à un sommet s
- est une succession d'arêtes orientées a gauche (0) ou à droite (1)
- Chaque sommet est identifié par code binaire unique
- Exemple : nœud 11 : "1 1 0" ; nœud 2 : "0" ; nœud 1 : "" ou encore "ε"



Aujourd'hui

Comment parcourir un arbre

Objectif : parcourir un arbre (pour imprimer les sommets ou pour les numéroter). Une première réponse, le parcours en profondeur d'abord (DFS) . Plusieurs variantes mais 3 étapes essentielles :

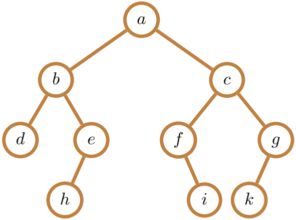
- Récursion sur le sous-arbre gauche
- Récursion sur le sous-arbre droit
- Impression (ou numérotation) du sommet courant

L'ordre dans lequel on effectue ces opérations est déterminant :

- Préfixe : sommet courant puis sous-arbre gauche puis sous-arbre droit
- Infixe : sous-arbre gauche puis sommet courant puis sous-arbre droit
- Postfixe : sous-arbre gauche puis sous-arbre droit puis sommet courant

Parcours DFS "préfixe"

- Visiter la racine
- Visiter le sous-arbre gauche
- Visiter le sous-arbre droit
- Implémentation récursive (ou dérécursivée avec une pile)

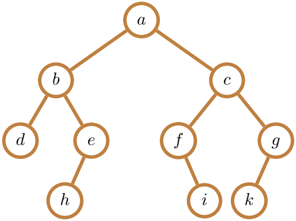


Ordre de visite
$a, b, d, e, h, c, f, i, g, k$

Parcours DFS “préfixe”

Coder l’arbre (de char)

```
Arbre e = new Arbre(new Arbre('h'), 'e', null);
Arbre b = new Arbre(new Arbre('d'), 'b', e);
Arbre f = new Arbre(null, 'f', new Arbre('i'));
Arbre g = new Arbre(new Arbre('k'), 'g', null);
Arbre c = new Arbre(f, 'c', g);
Arbre a = new Arbre(b, 'a', c);
```

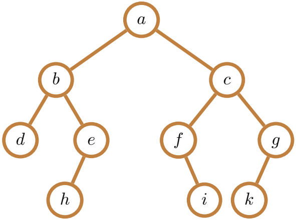


Ordre de visite

$a, b, d, e, h, c, f, i, g, k$

Parcours DFS "préfixe" : Codage

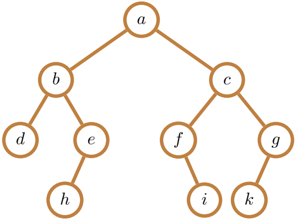
```
static void prefixe(Arbre x) {
    if (x == null) return;
    System.out.println(x.val);
    prefixe(x.gauche);
    prefixe(x.droite);
}
```



Ordre de visite
$a, b, d, e, h, c, f, i, g, k$

Parcours DFS "infixe"

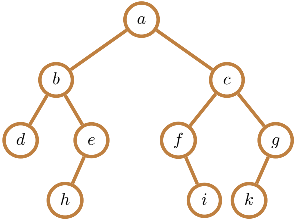
- Visiter le sous-arbre gauche
- Visiter la racine
- Visiter le sous-arbre droit
- Implémentation récursive (ou dérécursivée avec une pile)



Ordre de visite
$d, b, h, e, a, f, i, c, k, g$

Parcours DFS "infixe"

```
static void infixe(Arbre x) {
    if (x == null) return;
    infixe(x.gauche);
    System.out.println(x.val);
    infixe(x.droite);
}
```

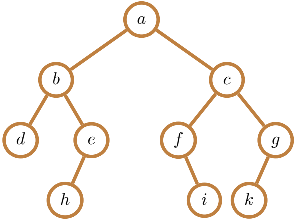


Ordre de visite

$d, b, h, e, a, f, i, c, k, g$

Parcours DFS "suffixe"

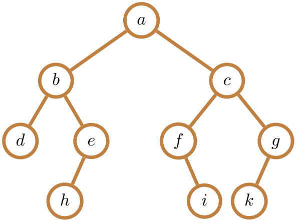
- Visiter le sous-arbre gauche
- Visiter le sous-arbre droit
- Visiter la racine
- Implémentation récursive (ou dérécursivée avec une pile)



Ordre de visite
$d, h, e, b, i, f, k, g, c, a$

Parcours DFS “suffixe”

```
static void suffixe(Arbre x) {
    if (x == null) return;
    suffixe(x.gauche);
    suffixe(x.droite);
    System.out.println(x.val);
}
```



Ordre de visite

$d, h, e, b, i, f, k, g, c, a$

Calculer les pères

Etant donné un sommet, comment remonter à la racine ?

- Ajouter une référence vers le père de chaque sommet
- Calculer les pères pendant la DFS (ou à la création de l'arbre)

```
static void prefixe(Arbre x) {
    if (x == null) return;
    System.out.println(x.val);
    if (x.gauche ≠ null) x.gauche.pere = x;
    if (x.droite ≠ null) x.droite.pere = x;
    prefixe(x.gauche); prefixe(x.droite); }
static void cheminRacine(Arbre x) {
    if (x == null) return;
    cheminRacine(x.pere);
    System.out.print(x.val + " -> "); }

// vers h : a -> b -> e -> h ->
```