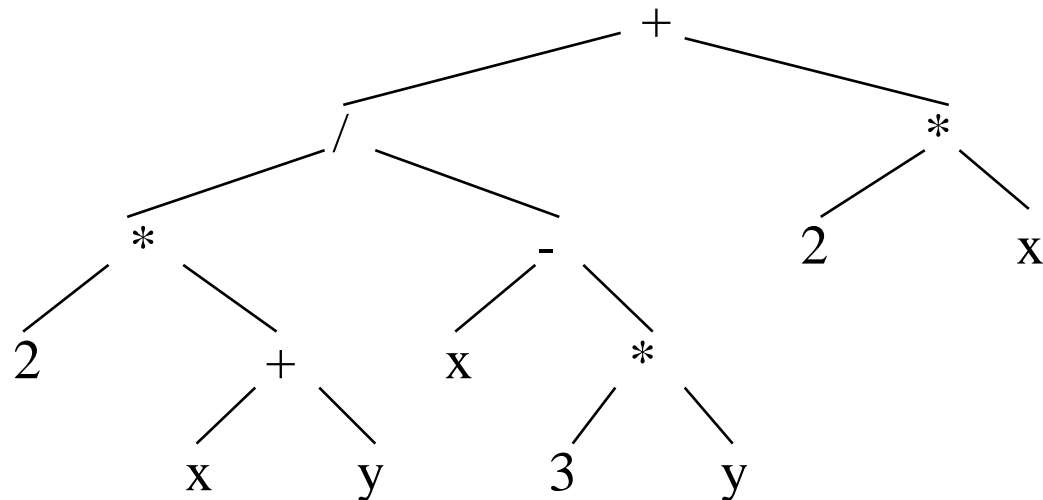
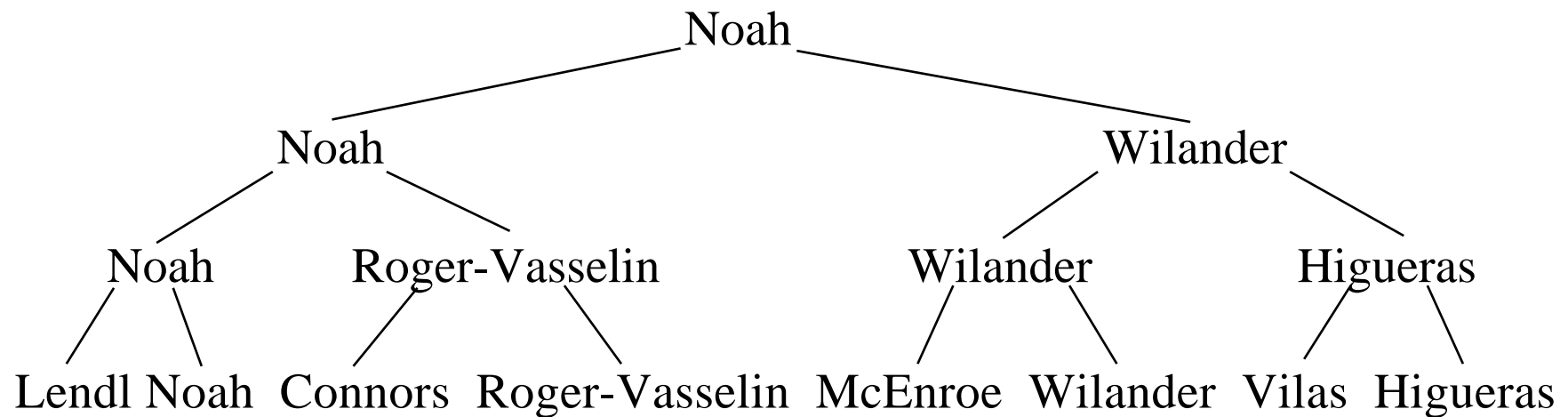


Structures arborescentes

Exemples d'arbres



spécification des arbres binaires

type arbre

paramètre nœud

opérations

\emptyset : arbre

$\langle _, _, _ \rangle$: nœud \times arbre \times arbre arbre

racine : arbre / nœud

g : arbre / arbre

d : arbre / arbre

préconditions

racine(a): a $\neq \emptyset$

g(a): a $\neq \emptyset$

d(a): a $\neq \emptyset$

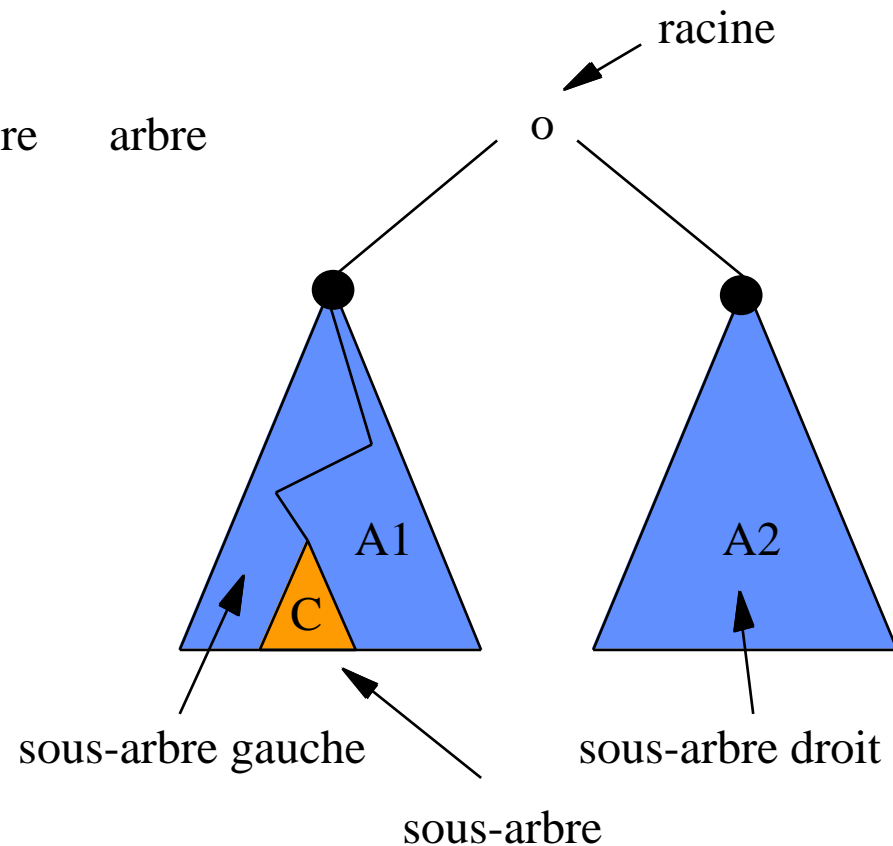
axiomes

o: nœud, A1, A2: arbre

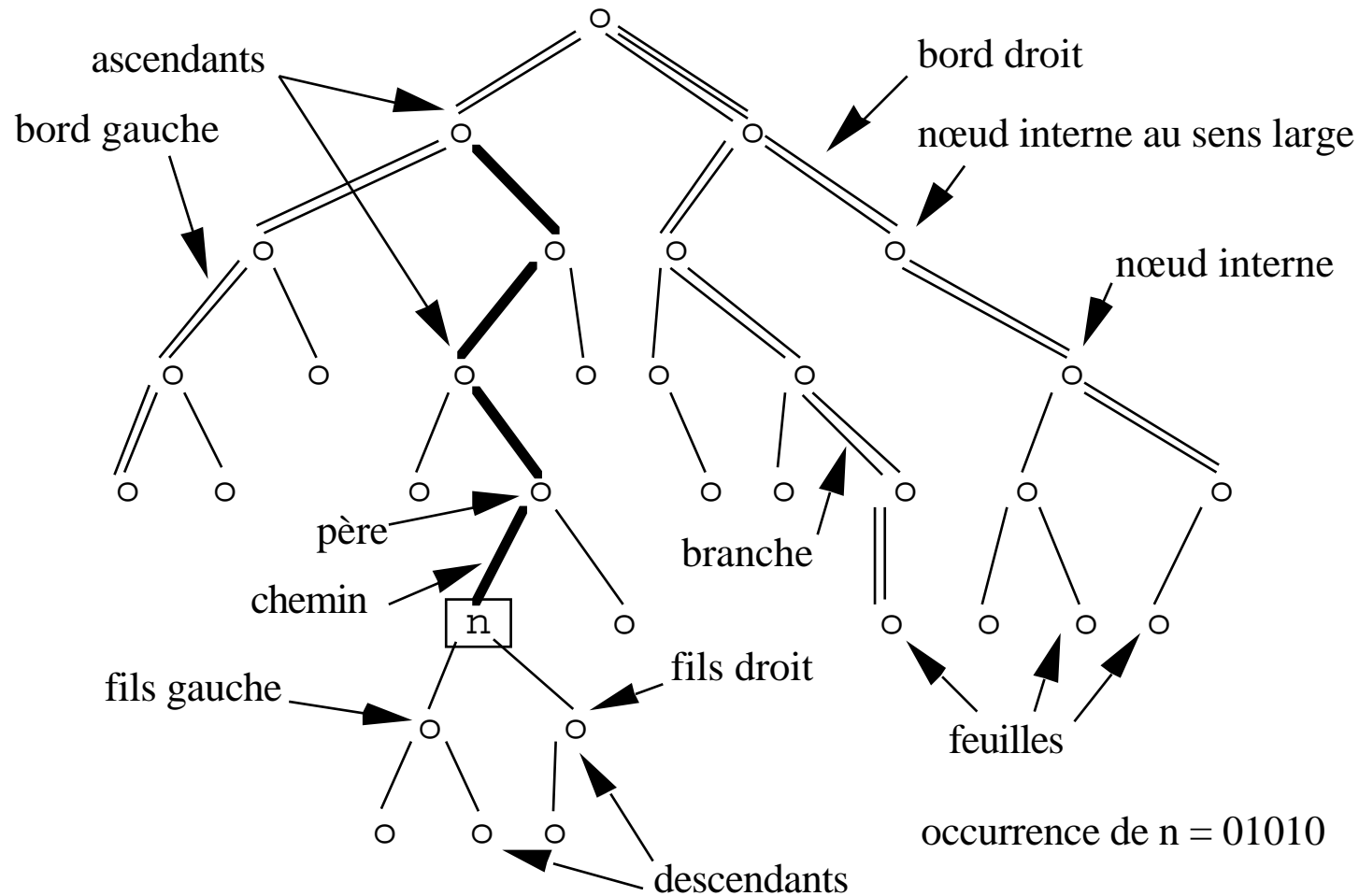
racine($\langle o, A1, A2 \rangle$) = o

g($\langle o, A1, A2 \rangle$) = A1

d($\langle o, A1, A2 \rangle$) = A2



terminologie

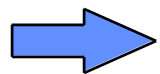


recherche d'un nœud

```
décision : nœud    (à_gauche, trouvé, à_droite)
recherche (<o, A1, A2>) = si décision(o) = trouvé alors o
                        sinsi décision(o) = à_gauche alors recherche(A1)
                        sinon recherche(A2) fsi

fonction recherche (a: arbre) retourne noeud;
début
    tant que décision (racine(a))    trouvé faire
        si décision (racine(a)) = à_gauche alors a := g(a);
                                sinon a := d(a);

        finsi;
    fait;
    retourner racine(a);
fin;
```



complexité de la recherche au pire en $h(a)$

mesures (1)

- nombre de nœuds:

$\text{taille}(A) = \text{si } A = \emptyset \text{ alors } 0$

$\text{sinon soit } A = \langle o, A_1, A_2 \rangle; 1 + \text{taille}(A_1) + \text{taille}(A_2) \text{ fsi}$

- nombre de feuilles:

$\text{nf}(A) = \text{si } A = \emptyset \text{ alors } 0$

$\text{sinon soit } A = \langle o, A_1, A_2 \rangle; \max(1, \text{nf}(A_1) + \text{nf}(A_2)) \text{ fsi}$

- hauteur d'un nœud dans un arbre de racine r:

$h(x) = \text{si } x = r \text{ alors } 0 \text{ sinon } 1 + h(\text{père}(x)) \text{ fsi}$

- hauteur de l'arbre (\emptyset): $h(a) = \max\{ h(x) \mid x \text{ nœud de } a \}$

$h(A) = \text{si } A = \emptyset \text{ alors } -1$

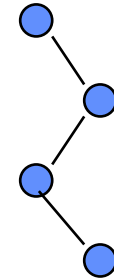
$\text{sinon soit } A = \langle o, A_1, A_2 \rangle; 1 + \max(h(A_1), h(A_2)) \text{ fsi}$

mesures (2)

- longueur de cheminement: $LC(A) = \{h(x) \mid x \text{ nœud de } A\}$.
 $LC(A) = \text{si } A = \emptyset \text{ alors } 0$
 $\text{sinon soit } A = \langle o, A_1, A_2 \rangle; LC(A_1) + LC(A_2) + \text{taille}(A) - 1 \text{ fsi}$
- profondeur moyenne (d'un nœud) de l'arbre:
 $PC(A) = LC(A) / \text{taille}(A)$ ← *complexité moyenne recherche d'un nœud*
- longueur de cheminement externe:
 $LCE(A) = \{h(x) \mid x \text{ feuille de } A\}$.
 $LCE(A) = \text{si } A = \emptyset \text{ alors } 0$
 $\text{sinon soit } A = \langle o, A_1, A_2 \rangle; LCE(A_1) + LCE(A_2) + \text{nf}(A_1) + \text{nf}(A_2) \text{ fsi}$
- profondeur moyenne externe (d'une feuille) de l'arbre:
 $PCE(A) = LCE(A) / \text{nf}(A)$. ← *complexité moyenne recherche d'une feuille*

propriétés sur les mesures

$$\begin{array}{lcl} \text{taille}(A) & 2 \text{ nf}(A) - 1 & \text{égalité si arbre complet} \\ \log_2 \text{taille}(A) & h(A) & \text{taille}(A) - 1 \\ \log_2 \text{nf}(A) & h(A) & \text{égalité si arbre filiforme} \end{array}$$



égalité atteinte pour les arbres complets qui ont toutes les feuilles à la profondeur $h(A)$

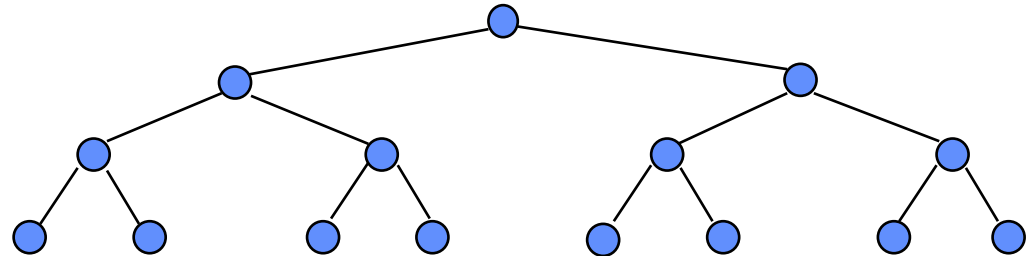
$$\text{PCE}(A) = \log_2 \text{nf}(A)$$

Si A de taille n ,

$$\text{PC}(A) = \frac{n+1}{n} \log_2(n+1) - 2$$

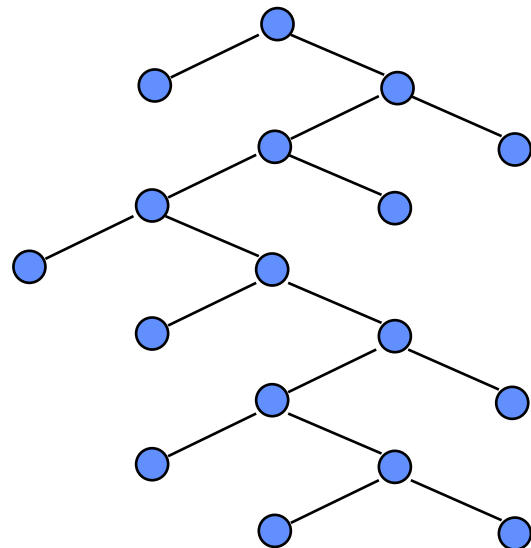
Si A complet de taille n ,

$$\text{PC}(A) = \frac{n+1}{n} \text{PCE}(A) - \frac{n-1}{n} \quad \Rightarrow \quad \text{PC}(A) = \text{PCE}(A) - 1$$



exemple de mesures

	profondeur	# nœuds	# feuilles
	0	1	0
	1	2	1
	2	2	1
	3	2	1
	4	2	1
PC	4	2	1
PCE	5	2	1
	6	2	1
	7	2	1
	8	2	2



taille = 17

LC = 72

PC = 4,2 2.42

h = 8

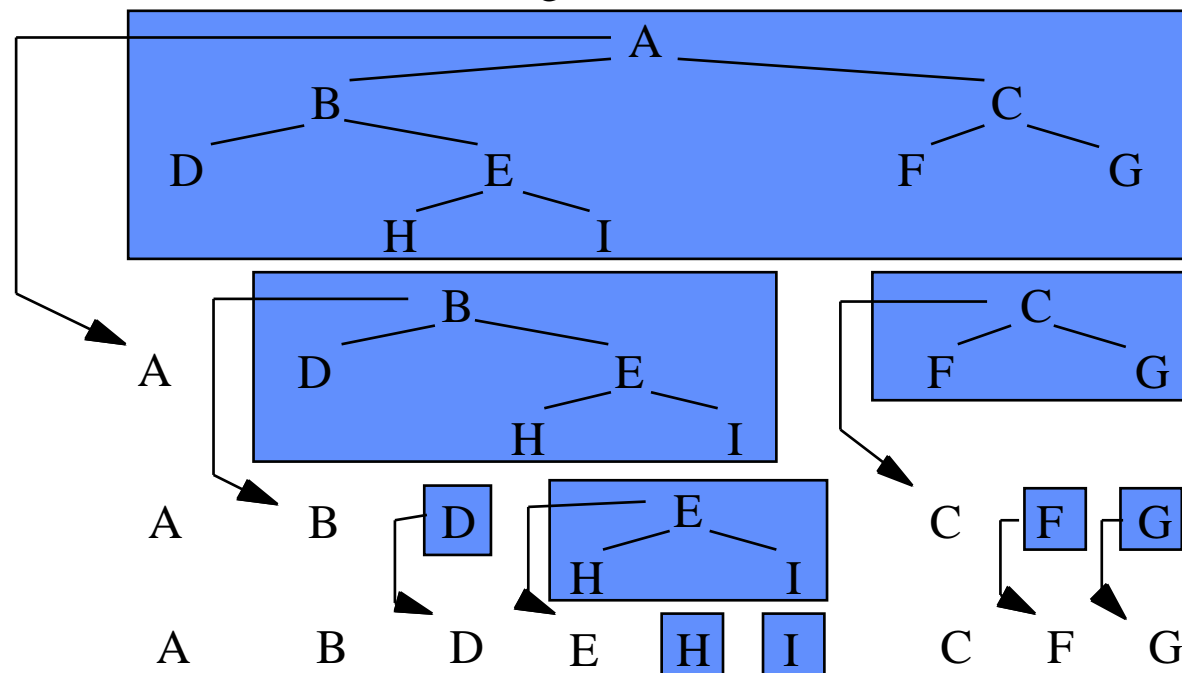
nf = 9

LCE = 44

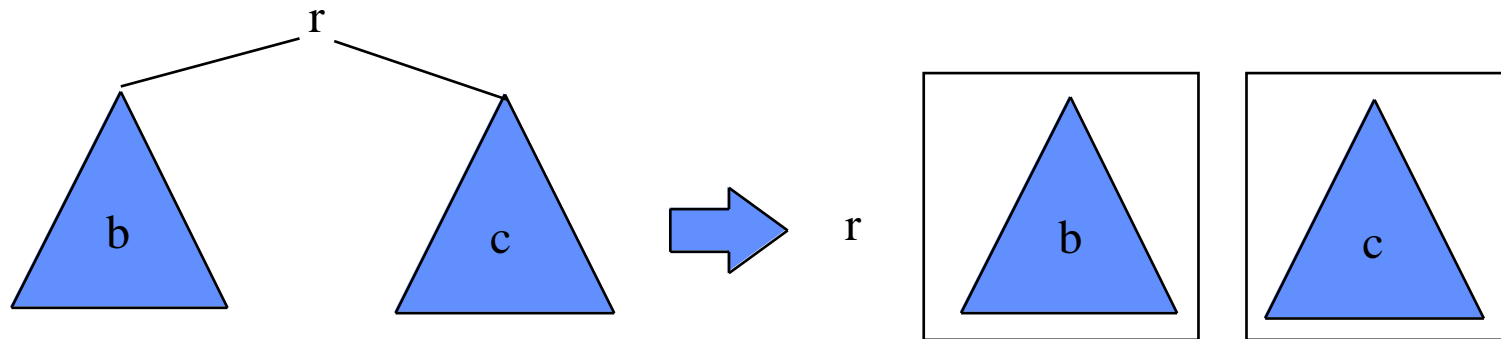
PCE = 4,9 3,2

ordre de parcours (1)

- problème : construire la liste préfixe d'un arbre binaire
tout nœud précède ses descendants,
les nœuds d'un sous arbre gauche sont avant ceux du sous arbre droit



ordre de parcours (2)



- Généralisation de la solution

- Spécification axiomatique

préfixe (A) = **si** A = \emptyset **alors** listevide

sinon soit A = <r, b, c>; cons (r, préfixe(b)&préfixe(c)) **fsi**

- Forme itérative

iter-pref(ln, la) = **si** la = listevide **alors** ln

sinsi la = \emptyset :: la' **alors** iter-pref(ln, la')

sinon soit la = <r,b,c> :: la'; iter-pref(ln&[r], [b;c]&la' **fsi**

préfixe (A) = iter-pref (listevide, [a])

ordre de parcours (3)

extension type arbre

utilise liste [nœud]

opérations

préfixe : arbre liste

infixe : arbre liste

postfixe : arbre liste

sémantique

préfixe (A) = **si** A = \emptyset **alors** listevide

sinon soit A = $\langle o, G, D \rangle$; [o] & préfixe(G) & préfixe(D) **fsi**

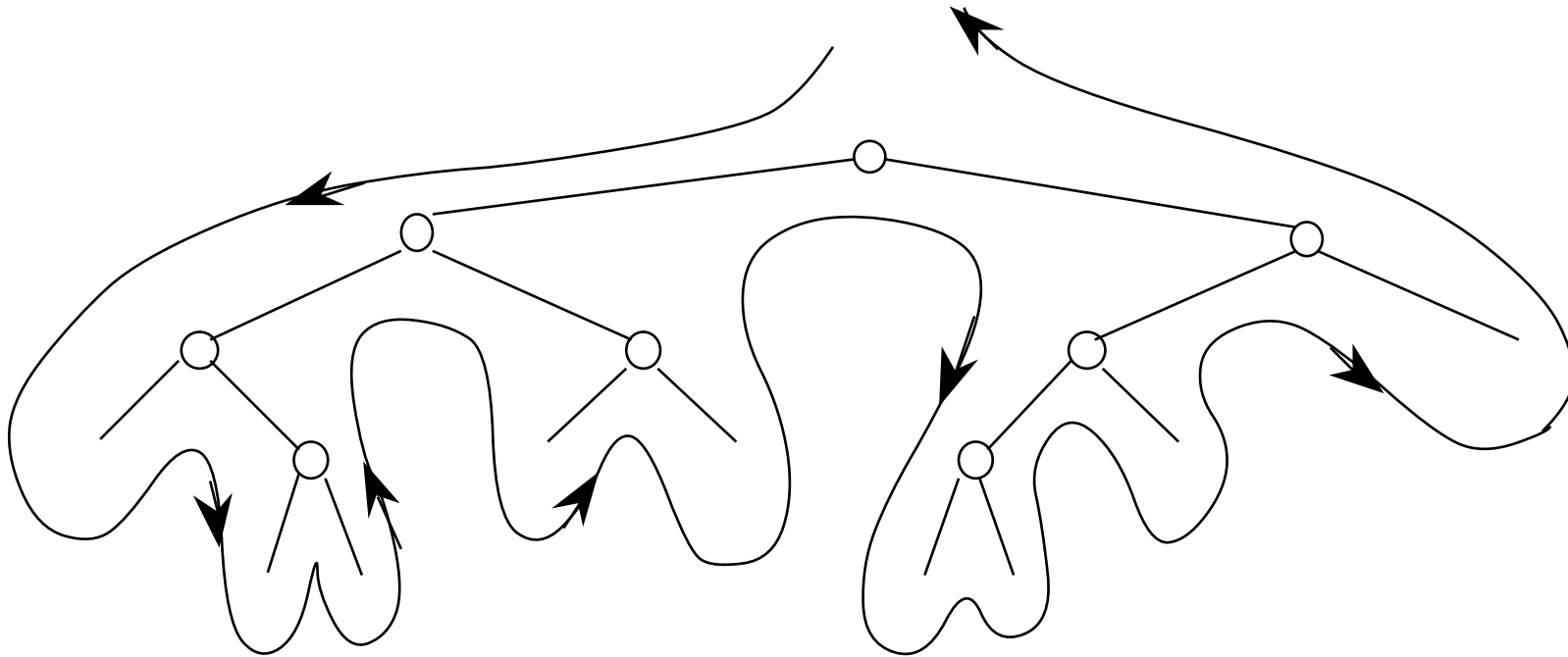
infixe (A) = **si** A = \emptyset **alors** listevide

sinon soit A = $\langle o, G, D \rangle$; infixe(G) & [o] & infixe(D) **fsi**

postfixe (A) = **si** A = \emptyset **alors** listevide

sinon soit A = $\langle o, G, D \rangle$; postfixe(G) & postfixe(D) & [o] **fsi**

exploration d'un arbre (1)



```
procédure explorer ( A: arbre );  
début si A =  $\emptyset$  alors trait_arbre_vide;  
      sinon trait_préfixe ( racine (A) );  
            explorer ( g (A) ); trait_infixe ( racine (A) ); explorer ( d (A) );  
            trait_postfixe ( racine (A) ); finsi;  
fin;
```

exploration d'un arbre (2)

```

procédure explor_iter ( A: arbre );
type t_sens = (à_gauche, à_droite); var P: pile [⟨arbre, t_sens⟩]; sens: t_sens;
début sens := à_gauche;

```

```

    répéter si sens = à_gauche alors

```

```

        tantque A ≠ ∅ faire trait_prefixe(racine(A));

```

```

explorer(g(A)) → empiler(P, ⟨A, à_gauche⟩); A := g(A); fait;

```

```

A=fils gauche(sommet) trait_arbre_vide;

```

```

finsi; ← fin exploration de A

```

```

si ¬ estvide (P) alors

```

```

    ⟨A, sens⟩ := sommet (P); dépiler (P);

```

```

    si sens = à_gauche alors trait_infixe( racine(A) );

```

```

explorer(d(A)) → empiler (P, ⟨A, à_droite⟩); A := d (A);

```

```

A=fils droit(sommet) sinon trait_post_fixe ( racine (A) ); finsi;

```

```

finsi;

```

```

jusqu'à estvide (P);

```

```

fin;

```

pile = chemin de la racine de l'arbre complet jusque racine(A)

implantation contiguë par linéarité

- par linéarisation: liste préfixe ou liste postfixe

$\text{rang}(\text{fils_gauche}(o)) = \text{rang}(o) + 1$

$\text{rang}(\text{fils_droit}(o)) = \text{rang}(o) + \text{taille}(\text{sous-arbre-gauche}(o)) + 1$

fonction rang_fils_droit(l: liste_préfixe, k: entier) **retourne** entier;

var i, j: entier;

début j := k + 1;

si fils_g_existe(ième(l, k)) **alors**

 i := 1; {i nombre de noeuds du sous arbre gauche restant à parcourir}

tantque i < 0 **faire** i := i + nb_fils(ième(l, j)) - 1;

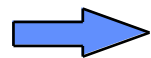
 j := j + 1;

fait;

finsi;

retourner (j);

fin;



complexité de $d(a)$ dépend de la taille des données

implantation contiguë par niveau (1)

- rappel occurrence

occurrence (x) \Rightarrow mot de $\{0, 1\}^*$ \longleftarrow *injectif*

code : $\{0, 1\}^* \rightarrow \mathbb{N}^*$ \longleftarrow *bijectif*

code () = 1

code ($\mu 0$) = 2 * code (μ)

code ($\mu 1$) = 2 * code (μ) + 1

code : nœud $\rightarrow \mathbb{N}^*$, code (x) = code(occurrence(x)) \longleftarrow *injectif*

code (père (x)) = code (x) div 2

code (filsgauche (x)) = 2 * code (x)

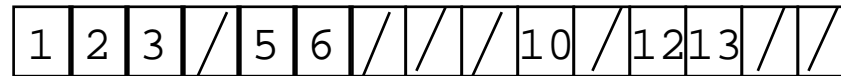
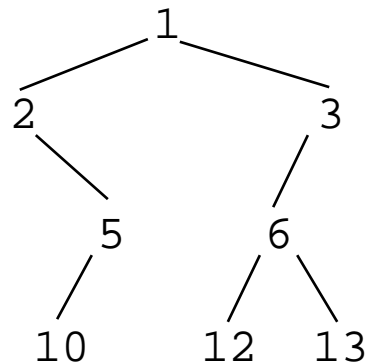
code (filsdroit (x)) = 2 * code (x) + 1

h (x) = \log_2 code (x) \Rightarrow *organisation par niveau*

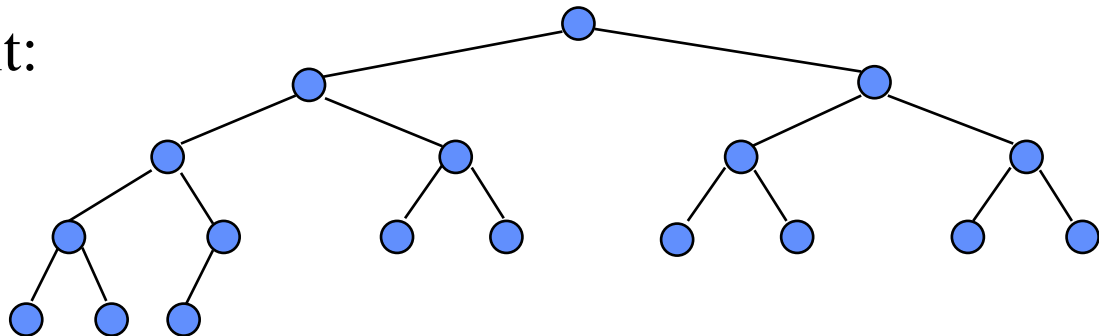
- prendre un espace contiguë $2^{h(A+1)}-1$ emplacements

nœud o placé en code(o)

implantation contiguë par niveau (2)

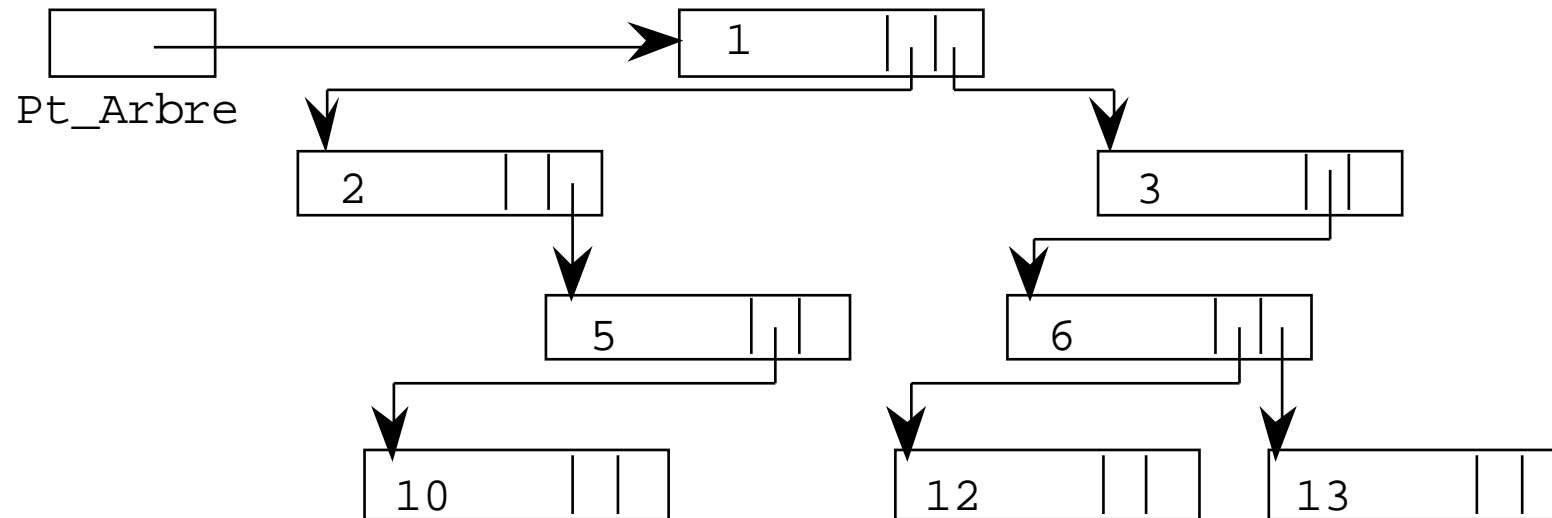
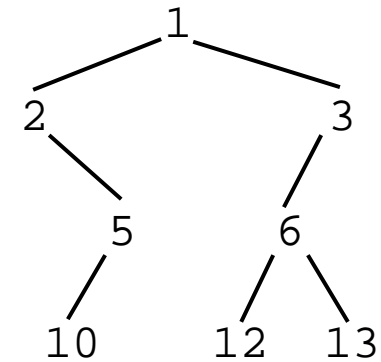


- occupation: $\text{taille}(A)/(2^{h(A+1)}-1)$, peut être très mauvais
- arbre binaire parfait:
 code(o) $\text{taille}(A)$
 occupation à 100%
- voir les tas



implantation chaînée

```
type Noeud; type Pt_Noeud is access Noeud;  
type Noeud is  
  record  
    Contenu : Element;  
    Gauche, Droite : Pt_Noeud;  
  end record;
```

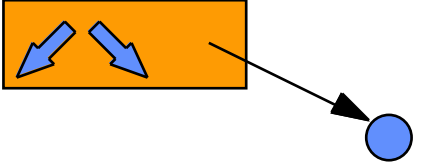


paquetage des arbres (1, Ada)

```
with Ada.Finalization;
package Arbres_Binaires is
  type Arbre (Hauteur_Max : Positive) is
    abstract tagged limited private ;
private
  type Noeud; type Pt_Noeud is access Noeud'Class;
  type Noeud is abstract tagged record
    Gauche, Droite : Pt_Noeud;
  end record;
  type Position is record
    Le_Noeud : Pt_Noeud := null; A_Gauche : Boolean;
  end record;
  type Tab_Pt_Noeud is array (Positive range <>) of Position;
  type Arbre (Hauteur_Max : Positive) is abstract
    new Ada.Finalization.Limited_Controlled with record
      Chemin : Tab_Pt_Noeud (1.. Hauteur_Max) ; Sommet : T_Sommet := 0;
    end record;
end Arbres_Binaires;
```

structure de base d'un nœud

constituant d'un chemin



chemin dans l'arbre

paquetage des arbres (2, Java)

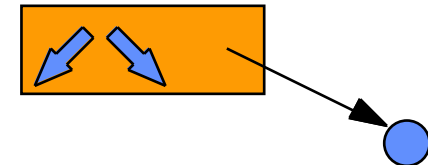
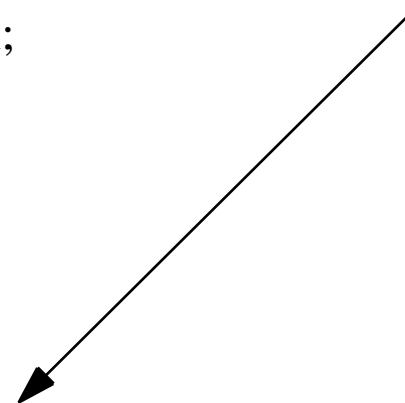
```
package bibSDjava.Les_Arbres_Binaires;
```

```
public abstract class Noeud {  
    public Noeud Gauche = null;  
    public Noeud Droite = null;  
}
```

```
public class Position {  
    public Noeud Le_Noeud;  
    public boolean A_Gauche;  
}
```

```
public abstract class Arbre_Binaire {  
    protected Position Chemin [];  
    protected int Sommet = -1;  
    public Arbre_Binaire () { Chemin = new Position [500];  
                                Chemin[0] = new Position (); }  
}
```

← *pas de création d'objet*



paquetage des arbres (3)

- brique de base, incomplet
- opérations fournies:
 - parcours en liste préfixe, infixé, postfixé
 - hauteur d'un nœud
 - destruction de l'arbre
 - aucune opération de modification de la structure, car les critères sont inconnus
 - aucune visibilité à l'utilisateur, mais visibilité aux paquetages enfants (Ada) ou aux classes dérivées (Java et C++)
- construction des opérations de parcours
 - exploration itérative -> simplification pour le parcours considéré
 - isolation des séquences <début trait 1> et <trait n trait n+1>

paquetage des arbres (4)

```
sens := à_gauche;
répéter
  si sens = à_gauche alors
    tantque A ≠ ∅ faire
      empiler(P, <A, à_gauche>);
      A := g(A);
    fait;
  finsi;
  si ¬ estvide (P) alors
    <A, sens> := sommet (P);
    dépiler (P);
    si sens = à_gauche alors
      trait_infixe( racine(A) );
      empiler (P, <A, à_droite>);
      A := d (A);
    finsi;
  finsi;
jusqu'à estvide (P);
```

```
● trait n    trait n+1
  si d(A) = ∅ alors
    empiler(P, <A, à_droite>);
    A := d(A);
    tantque g(A) = ∅ faire
      empiler (P, <A, à_gauche>);
      A := g(A);
    fait;
  sinon
    tantque ¬ estvide(P) faire
      <A,sens> := sommet(P);
      dépiler(P);
      si sens = à_gauche alors
        sortie;
      finsi;
    fait;
  finsi;
```

arbres généraux et forêts (1)

type arbregen, forêt

paramètre nœud; **utilise** entier

opérations

$\langle _, _ \rangle$: nœud \times forêt arbregen

racine : arbregen nœud

sousarbres : arbregen forêt

\emptyset : forêt

longueur : forêt entier

ième : forêt \times entier / arbregen

insérer : forêt \times entier \times arbregen / forêt

— *forêt = liste[arbregen]*

préconditions

ième(f, k): $1 \leq k \leq \text{longueur}(f)$

insérer(f, k, a): $1 \leq k \leq \text{longueur}(f) + 1$

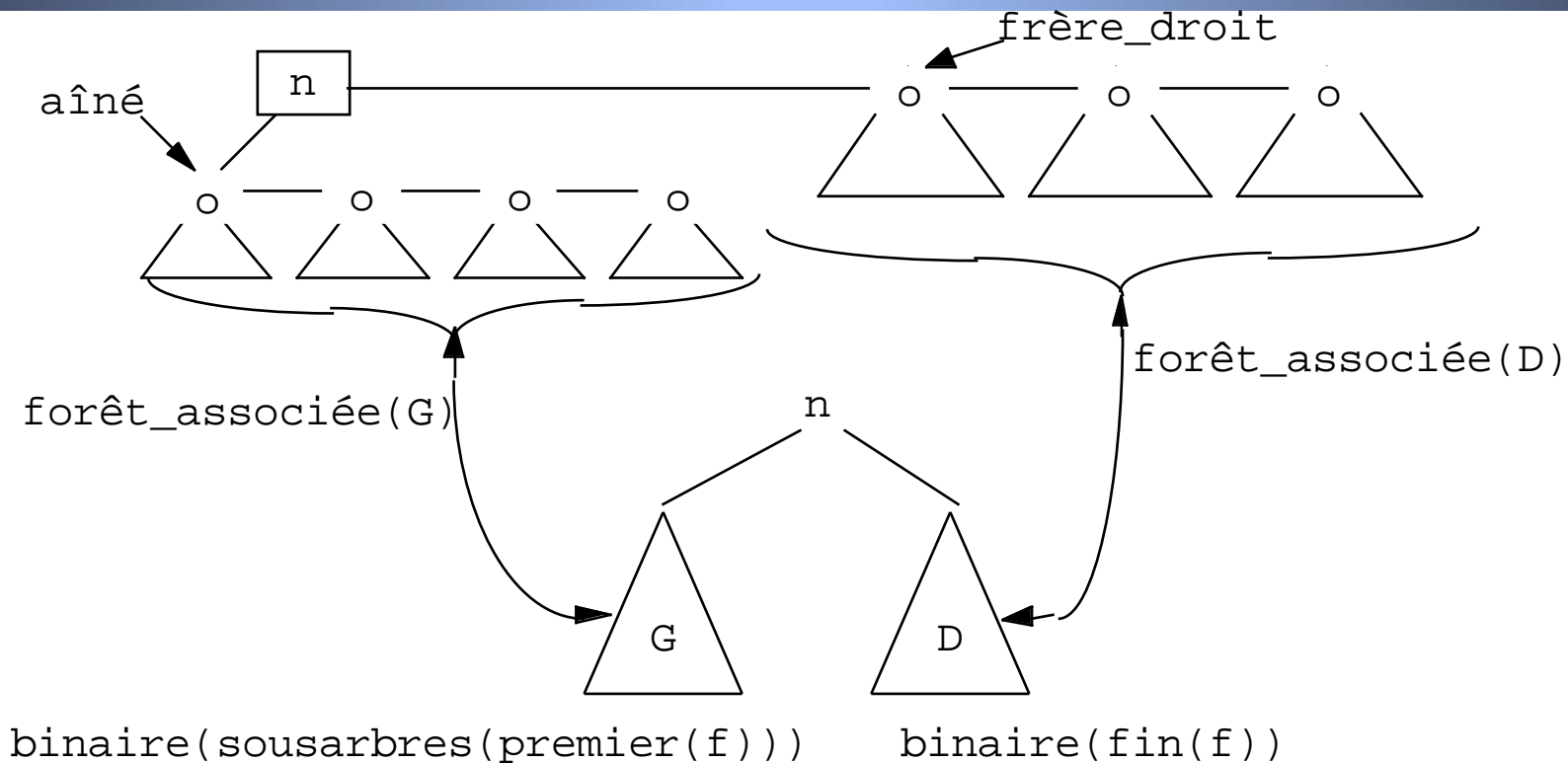
axiomes

racine($\langle o, f \rangle$) = o

sousarbres($\langle o, f \rangle$) = f

+ *axiomes des listes*

arbres généraux et forêts (2)



$\text{binaire}(f) = \text{si } f = \emptyset \text{ alors } \emptyset \text{ sinon soit } f = \langle o, f' \rangle :: f''; \langle o, \text{binaire}(f'), \text{binaire}(f'') \rangle \text{ fsi}$

$\text{f_a}(A) = \text{si } A = \emptyset \text{ alors } \emptyset \text{ sinon soit } A = \langle o, G, D \rangle; \langle o, \text{f_a}(G) \rangle :: \text{f_a}(D) \text{ fsi}$

implantation des arbres généraux (1)

- implantation contiguë par linéarisation préfixe ou postfixe
connaissance du nombre de fils, mêmes problèmes que arbres binaires
- implantation contiguë par niveau => nombre de fils borné
aggravation de la mauvaise occupation
- implantation chaînée avec 1 pointeur par fils
borner le nombre de fils => taille fixe, mal occupée (feuilles)
emplacements de taille variable => gestion difficile
bien adapté aux arbres n-aires
voir B-arbres

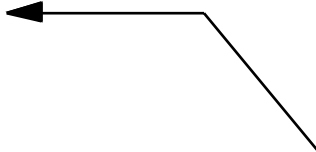
implantation des arbres généraux (2)

```
with Listes_Contigues;  
generic  
  Taille : in Positive := 100;                -- nombre de fils maximum  
  type Valeur_Noeud is private;  
package Arbres_Generaux is  
  type Arbre_General;  
  type Pt_Arbre is access Arbre_General;  
  package Forets is new Listes_Contigues(Element => Pt_Arbre);  
  subtype Foret is Forets.Liste (Taille_Max => Taille);  
  type Arbre_General is                        -- type définissant un nœud  
    record  
      Contenu : Valeur_Noeud;  
      Sous_Arbres : Foret;  
    end record;  
end Arbres_Generaux;
```

implantation des arbres généraux (3)

- implantation chaînée de l'arbre binaire associé à la forêt

```
type Arbregen;  
type Pt_Arbregen is access Arbregen;  
type Arbregen is  
  record  
    Contenu : Element;  
    Aine, Suivant : Pt_Arbregen;  
    Frere_Null : Boolean;  
end record;
```



variante: si pas de frère, le suivant est le père

implantation des arbres généraux (4)

