



## Plan

- I. Les arbres
- II. Les arbres binaires
- III. Les arbres binaires de recherche

2



## Les arbres - Définitions

### Définition :

Un arbre est un ensemble de noeuds organisés de façon hiérarchique, selon une relation de parenté, à partir d'une racine unique.

Un noeud est constitué d'un élément et de liens vers d'autres noeuds.

### Définition inductive : Un arbre est :

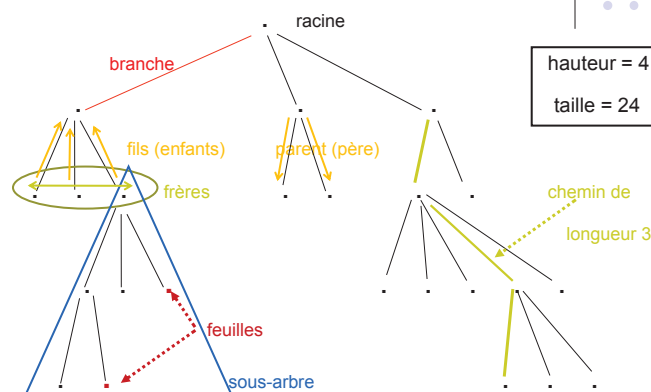
- soit un noeud unique (appelé aussi racine de l'arbre),
- soit construit à partir d'un noeud  $n$  et de  $k$  arbres  $A_1, \dots, A_k$  (de racines respectives  $n_1, \dots, n_k$ ) et tel que  $n$  est parent unique de  $n_1, \dots, n_k$ .

On définit un arbre vide comme un arbre sans noeud.

4



## Les arbres - Terminologie



6



# Structures arborescentes

1

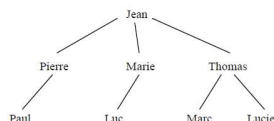
## I – Les arbres

La **structure arborescente** est très importante en informatique car elle apparaît dans de nombreux problèmes, notamment pour représenter un ensemble d'éléments avec *deux relations d'ordre*.

C'est un *cas particulier de graphe*.

Relation *verticale*  $V$  = Parent, père, mère

Relation *horizontale*  $H$  = Frère, soeur  
(selon âge décroissant p. ex.)



Ici on a :  $V(\text{Jean}, \text{Pierre})$ ,  $H(\text{Pierre}, \text{Marie})$ ,  $H(\text{Pierre}, \text{Thomas})$

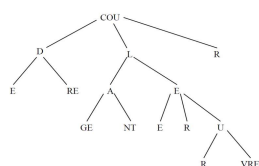
3



## Les arbres

Les arbres sont bien adaptés à la représentation d'une hiérarchie :

- une arborescence de répertoires ou fichiers Unix, Windows...
- la structure syntaxique d'un programme source (compilateur)
- des organigrammes de sociétés, questionnaires, tables des matières, index, dictionnaires...



5

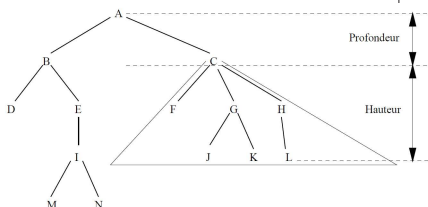
**Exemple :** représentation d'un index (lexique) de neuf mots :

coude, coudre, coulage, coulant, coulée,  
couler, couleur, couleuvre, cour.

Un mot correspond à une branche de l'arbre.

Nb mots = nb de feuilles

## Les arbres - Terminologie



- nœud étiqueté.
- ancêtre/ascendant : clôture de la relation "parent/père".
- descendant : clôture de la relation "fils/enfant".
- degré d'un nœud : nombre d'enfants de ce nœud.
- profondeur d'un nœud : longueur du chemin de la racine à ce nœud.
- hauteur d'un nœud : longueur du plus long chemin de ce nœud à une feuille de son sous-arbre.

7

## II – Les arbres binaires

### Définition :

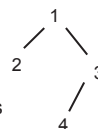
Un arbre binaire est un arbre tel que tout nœud a au-plus deux fils, appelés fils gauche et fils droit.

### Primitives des arbres binaires :

Création d'arbre, test arbre vide, accès au nœud racine, accès aux fils (sous-arbres) gauche et droit

### Primitives des nœuds :

Test feuille, consultation et modification de la valeur d'un nœud



8

## Parcours d'arbres binaires

Le parcours d'un arbre consiste à examiner tous les nœuds de l'arbre.

Parmi les différents ordres de parcours possibles pour un arbre, citons :

- **les parcours en profondeur**, avec des variantes qui diffèrent selon le moment où est traité le nœud courant
  - Parcours préfixe (ou préordre ou DGD)
  - Parcours postfixe (ou postordre ou AGD)
  - Parcours infixé (ou symétrique)
- **le parcours en largeur**

9

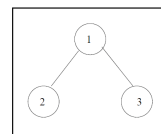
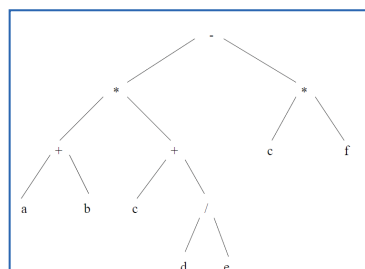
## Parcours préfixe (ou préordre ou DGD)

Étude de la racine

puis Étude du sous-arbre gauche

puis Étude du sous-arbre droit

DGD = Descendant Gauche Droite



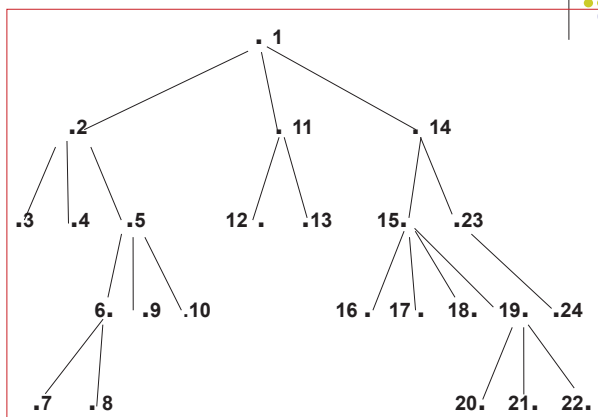
Énumération préfixe :

- \* + a b + c / d e \* c f

L'énumération préfixe d'un arbre d'expression arithmétique s'appelle notation polonaise préfixée de l'expression arithmétique.

10

## Ordre préfixe sur arbre non binaire (exemple p.6)



11

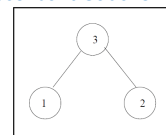
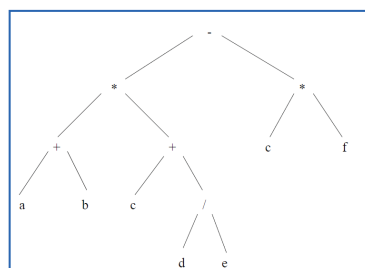
## Parcours postfixe (ou postordre ou AGD)

Étude du sous-arbre gauche

puis Étude du sous-arbre droit

puis Étude de la racine

AGD = Ascendant Gauche Droite



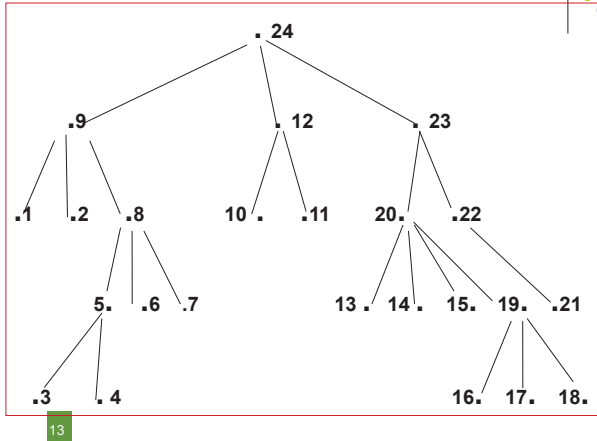
Énumération postfixe :

a b + c d e / + \* c f \*

L'énumération postfixe d'un arbre d'expression arithmétique s'appelle notation polonaise postfixée de l'expression arithmétique.

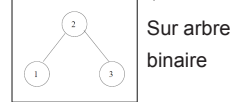
12

## Ordre postfixe sur arbre non binaire (exemple p.6)

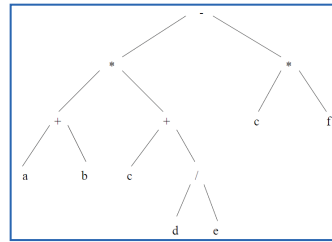


## Parcours infixe (ou symétrique)

Étude du sous-arbre gauche  
 puis Étude de la racine  
 puis Étude du sous-arbre droit



Énumération infixe :  
 $a + b * c + d / e - c * f$



L'énumération infixe d'un arbre d'expression arithmétique, avec parenthésage, correspond à l'évaluation habituelle d'une expression arithmétique.

**Parenthésage :**  
 - descente sur fils gauche = ouvrir parenthèse  
 - remontée du fils droit = fermer parenthèse

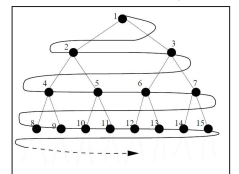
## Algorithme général pour les parcours en profondeur

```
void parcours(Arbre_binaire A)
{
    if (!est_vide?(A))
    {
        traitement_racine_préfixe;
        parcours(fils_gauche(A));
        traitement_racine_infixe;
        parcours(fils_droit(A));
        traitement_racine_postfixe;
    }
}
```

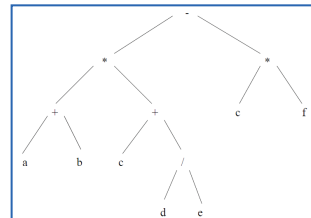
Complexité :  $O(n)$

## Parcours en largeur (ou "par niveaux décroissants")

Étude de la racine  
 puis Étude du premier niveau de l'arbre (de gauche à droite)  
 puis Étude du second niveau de l'arbre (de gauche à droite)  
 etc.



Énumération en largeur :  
 $- * + + c f a b c / d e$



Aucun intérêt dans le cas d'arbres d'expressions arithmétiques !

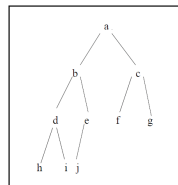
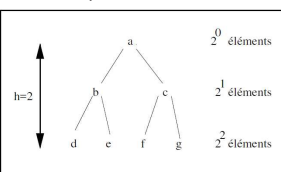
## Arbres binaires particuliers

### Arbre binaire complet

si tout noeud de profondeur inférieure à la hauteur de l'arbre a exactement deux fils.

Un arbre complet de hauteur  $h$  contient :  $\sum_{i=0}^h 2^i = 2^{h+1} - 1$  éléments.

$2^{h+1} - 1$   
 $= 2^{2+1} - 1$   
 $= 2^3 - 1$   
 $= 7$  noeuds



### Arbre binaire complet à gauche (ou parfait)

s'il est complet jusqu'à la profondeur  $h-1$  et si à la profondeur  $h$ , les fils existants sont cadrés à gauche dans la représentation graphique de l'arbre (de hauteur  $h$ ).

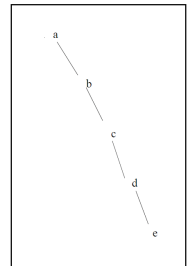
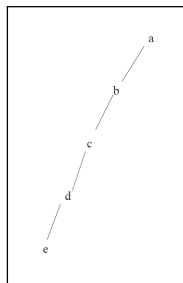
$2^h \leq \text{Nombre d'éléments} \leq 2^{h+1} - 1$

## Arbres binaires particuliers

### Arbre dégénéré (ou peigne)

si la structure se rapproche de celle d'une liste.  
 Les cas extrêmes sont appelés *peigne à gauche* et *peigne à droite*.

Nombre d'éléments =  $h + 1$



**Conclusion sur le rapport taille/hauteur des arbres binaires :**

$h+1 \leq t \leq 2^{h+1} - 1$

avec  $t$  = taille arbre (nb noeuds)

i.e.

$h$  = hauteur arbre

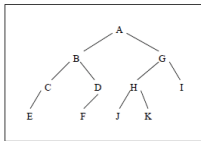
$\log_2(t) \leq h \leq t - 1$

## Implantation des arbres binaires



### Les principaux choix de représentation :

- représentation en pur chaînage,
- représentations mixtes tableau/chaînage,
- représentation avec stockage optimal de l'arbre (en tableau).



19

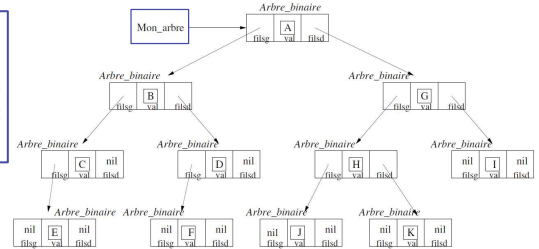
## Représentation en pur chaînage



C'est la technique la plus utilisée pour représenter les arbres en raison de son aspect dynamique.

Un arbre est représenté par sa racine, à partir de laquelle on peut retrouver tous les autres noeuds de l'arbre.

```
typedef struct Abin
{
    elt_arbre val;
    struct Abin * filsg;
    struct Abin * filsd;
} Arbre_binaire;
```



20

## Représentations mixtes tableau/chaînage (1)



**Avantage :** simplifier la gestion mémoire en utilisant des indices (entiers) dans un tableau pour faire le chaînage, à la place des pointeurs du chaînage classique.

**Inconvénient :** l'aspect dynamique de cette représentation est limité par la taille fixée du tableau (même dynamiquement).

```
typedef struct Abin
{
    entier racine;
    Noeud tab[N];
    Liste ind_libres;
} Arbre_binaire;

typedef struct Nbin
{
    elt_arbre val;
    entier filsg;
    entier filsd;
} Noeud;
```

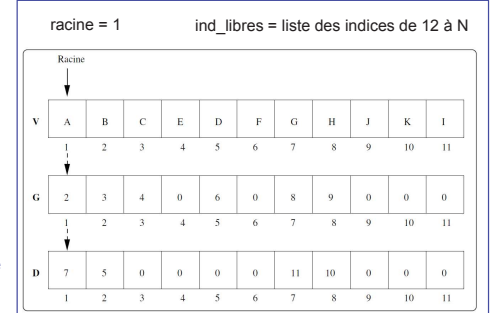


21

## Représentations mixtes tableau/chaînage (2)



```
typedef struct Abin
{
    entier racine;
    elt_arbre V[N];
    entier G[N];
    entier D[N];
    Liste ind_libres;
} Arbre_binaire;
```



Le lien entre les trois tableaux est fait via le  $i^{\text{ème}}$  élément de chacun d'entre eux, qui concerne le même ( $i^{\text{ème}}$ ) noeud de l'arbre.

22

## Représentation avec stockage optimal de l'arbre (en tableau)



**Objectif :** stocker les éléments, sans les liens, dans un tableau.

```
typedef elt_arbre Arbre_bin[N];
```

ou  

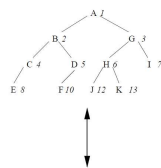
```
typedef elt_arbre* Arbre_bin;
```

Pour retrouver l'organisation interne de l'arbre, il est nécessaire de numéroté ses noeuds.

arbre de hauteur h  
 $N = 2^{h+1}-1$

**Principe (numérotation en largeur) :**  
Soit un noeud de numéro i,  
- son fils gauche a le numéro  $2*i$ ,  
- son fils droit a le numéro  $(2*i)+1$ .

La racine de l'arbre a le numéro 1.



23

## Représentation avec stockage optimal de l'arbre (en tableau)



### Avantages :

- Moins coûteux que les représentations précédentes (qui stockent un élément et deux liens pour chaque noeud).
- Le parcours séquentiel du tableau est un parcours en largeur de l'arbre.
- Cette représentation est optimale pour les arbres complets (pas de "trous" dans le tableau), mais aussi pour les arbres parfaits (complets à gauche) car les "trous" sont à la fin du tableau et on peut donc choisir  $N$  = nombre d'éléments de l'arbre.

### Inconvénients :

- Moins souple que les représentations avec chaînage,
- Très mal adapté aux ajouts/suppressions de noeuds internes (problème de la réorganisation des arbres complets).

24

### III – Les arbres binaires de recherche (ABR)

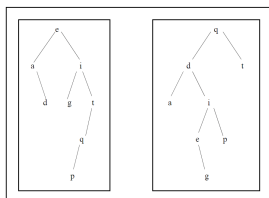
#### Définition :

Un ABR  $A = \langle r, G, D \rangle$  est un arbre binaire ordonné tel que :

- la valeur associée au noeud  $r$  est :
  - $\geq$  aux valeurs de chaque noeud du sous-arbre  $G$ ,
  - $<$  aux valeurs de chaque noeud du sous-arbre  $D$ ,
- $G$  et  $D$  sont des ABR.

Le parcours symétrique d'un ABR produit la suite de ses éléments triée dans l'ordre croissant.

À une suite d'éléments en ordre croissant peuvent correspondre plusieurs ABR.



a d e g i p q t  
(ordre alphabétique)

25

### Recherche dans un ABR

#### Méthode :

Si l'ABR est vide alors élément non trouvé

Sinon, on compare l'élément recherché  $x$  avec la valeur  $v$  de la racine de l'ABR :

- si  $x = v$  alors élément trouvé,
- si  $x < v$  alors exploration du sous-arbre gauche,
- si  $x > v$  alors exploration du sous-arbre droit.

Complexité :  $O(\log_2(n))$

26

### Ajout dans un ABR

Deux possibilités :

- ajout aux feuilles (le plus courant et le moins coûteux)
- ajout à la racine (maintient un meilleur équilibre de l'arbre, préférable si recherche des éléments les plus récemment ajoutés)

Dans tous les cas, la propriété ABR doit être conservée.

27

### Ajout aux feuilles dans un ABR

#### Méthode :

Si l'ABR est vide alors on crée un arbre avec un seul noeud contenant le nouvel élément.

Sinon, on compare le nouvel élément  $x$  avec la valeur  $v$  de la racine de l'ABR :

- si  $x > v$  alors ajout dans le sous-arbre droit,
- sinon ajout dans le sous-arbre gauche.

Complexité :  $O(\log_2(n))$

28

### Ajout à la racine dans un ABR

#### Méthode pour ajouter un élément $x$ à la racine d'un ABR :

Si l'ABR est vide alors on crée un arbre avec un seul noeud contenant le nouvel élément.

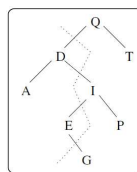
Sinon :

- couper l'ABR en deux sous-arbres (ABR)  $G$  et  $D$  tels que :
  - $G$  contient tous les éléments de l'arbre inférieurs ou égaux à  $x$ ,
  - $D$  contient tous les éléments de l'arbre strictement supérieurs à  $x$ .
- reconstruire ensuite un ABR avec  $x$  en racine,  $G$  en fils gauche et  $D$  en fils droit.

Complexité :  $O(\log_2(n))$

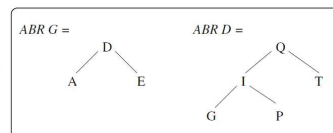
29

### Ajout à la racine dans un ABR

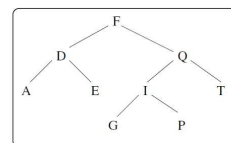


Mon\_abr

Coupeure selon F



Insertion de F à la racine



Ajout à la racine de l'élément  $F$  dans l'ABR Mon\_abr

30

## Principe de l'algorithme de coupure

Pour couper un ABR A en deux ABRs G et D selon une valeur x :

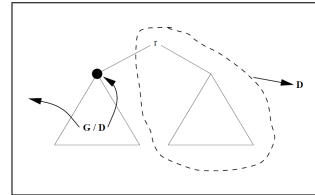
- si x est strictement inférieure à la valeur de la racine de A :
  - la racine de A et son sous-arbre droit appartiendront à D,
  - le sous-arbre gauche de A contient G et le futur sous-arbre gauche de (la racine de) D ;
- si x est supérieure ou égale à la valeur de la racine de A :
  - la racine de A et son sous-arbre gauche appartiendront à G,
  - le sous-arbre droit de A contient D et le futur sous-arbre droit de (la racine de) G.

Complexité :  $O(\log_2(n))$

31

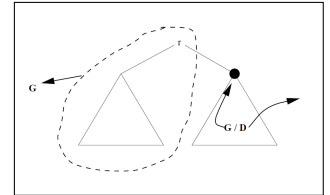
## Principe de l'algorithme de coupure

$x < \text{racine de A}$



32

$x \geq \text{racine de A}$



## Suppression dans un ABR

Méthode :

- 1) recherche de l'élément à supprimer dans l'ABR,
- 2) suppression de l'élément,
- 3) éventuellement réorganisation pour conserver le caractère ABR
  - cas suppression d'un noeud feuille : pas de réorganisation,
  - cas suppression d'un noeud ayant un fils unique,
  - cas suppression d'un noeud ayant deux fils.

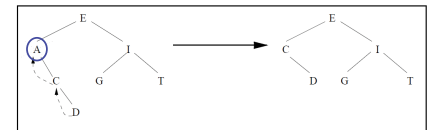
Complexité :  $O(\log_2(n))$

33

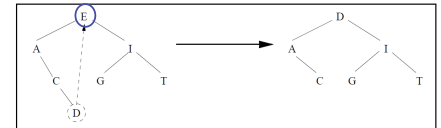


## Réorganisation suite à suppression dans un ABR

- cas suppression d'un noeud ayant un fils unique : remplacer ce noeud par son fils.



- cas suppression d'un noeud ayant deux fils : remplacer ce noeud par le plus grand élément (maximum) de son sous-arbre gauche.

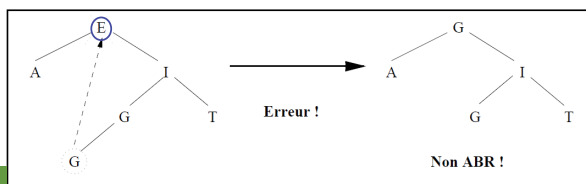


Complexité :  $O(\log_2(n))$

34

## Réorganisation suite à suppression dans un ABR

**Attention !** Dans le cas de la suppression d'un noeud ayant deux fils, la méthode "symétrique" ne fonctionne pas : remplacer le noeud supprimé par le plus petit élément de son sous-arbre droit ne conserve pas le caractère ABR (problème si le sous-arbre droit contient des doublons).



35

## Les arbres binaires de recherche (ABR)

La complexité de la recherche, l'ajout ou la suppression d'un élément :

- dans un ABR équilibré de taille n :  $O(\log_2(n))$
- dans un ABR dégénéré :  $O(n)$

Si ABR complet ou parfait : recherche toujours en  $O(\log_2(n))$  mais réorganisations supplémentaires lors des ajouts et suppressions.

Donc, complexité de  $O(\log_2(n))$  en moyenne et de  $O(n)$  dans le pire des cas.

Exemple : représentation d'un index via un ABR.

36