

- 1/ Rappels et compléments Java.
- 2/ Tableaux, boucles et invariants.
- 3/ Notions élémentaires de complexité.
- 4/ Récursion.
- 5/ Structures de données et introduction aux types abstraits de données. **ICI**
- 6/ Quelques compléments Java.

- Notion d'arbres, définitions et propriétés
- Type abstrait Arbre
- Algorithmes de parcours d'arbres
- Arbres binaires de recherche

## Les arbres

De manière informelle, un arbre est un ensemble d'éléments appelés **nœuds** liés par une **relation de parenté** établissant une hiérarchie entre les nœuds.

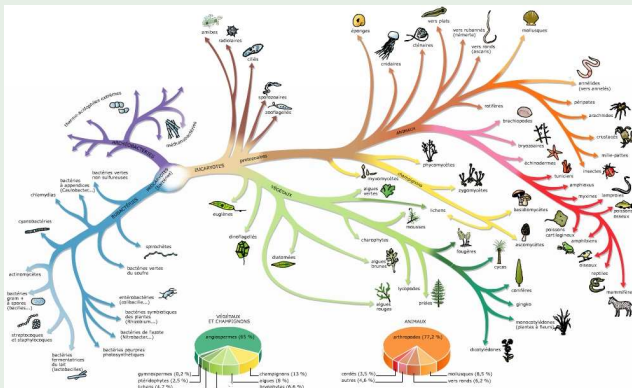
## Notion d'arbres

## Les arbres

De manière informelle, un arbre est un ensemble d'éléments appelés **nœuds** liés par une **relation de parenté** établissant une hiérarchie entre les nœuds.

## Examples

Un arbre généalogique ou un arbre représentant des espèces.



## Définition récursive

Un **arbre** est un ensemble fini de **nœuds** tel que:

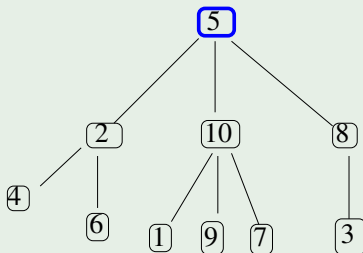
- Il existe un nœud particulier appelé le **nœud racine** (ou la **racine**).
- Les nœuds restants sont partitionnés en ensemble qui sont eux mêmes des arbres (parfois appelés aussi **sous-arbres**).

## Définition récursive

Un **arbre** est un ensemble fini de **nœuds** tel que:

- Il existe un nœud particulier appelé le **nœud racine** (ou la **racine**).
- Les nœuds restants sont partitionnés en ensemble qui sont eux mêmes des arbres (parfois appelés aussi **sous-arbres**).

## Un exemple



## Définition informelle récursive

Un **arbre** est un ensemble fini de **nœuds** tel que:

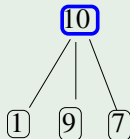
- Il existe un nœud particulier appelé le **nœud racine** (ou la **racine**).
- Les nœuds restants sont partitionnés en ensemble qui sont eux mêmes des arbres (parfois appelés aussi **sous-arbres**).

## Définition informelle récursive

Un **arbre** est un ensemble fini de **nœuds** tel que:

- Il existe un nœud particulier appelé le **nœud racine** (ou la **racine**).
- Les nœuds restants sont partitionnés en ensemble qui sont eux mêmes des arbres (parfois appelés aussi **sous-arbres**).

## Un exemple



- Un **noeud**  $N$  unique est un arbre dont la **racine** est  $N$  lui-même.
- Si  $N$  est un noeud et  $A_1, A_2, \dots, A_k$  sont des **arbres** dont les **racines** sont  $N_1, N_2, \dots, N_k$  alors la structure telle que  $N$  est le **noeud père** de  $N_1, N_2, \dots, N_k$  est aussi un **arbre**.
- Un arbre sans noeud est un arbre **vide**.
- Un forêt est un ensemble d'arbres.

- Liens de **parenté** : père, fils, frère, ... entre les nœuds (aussi désignés par **sommets** ).
- **Branche**.
- **Nœuds** : nœud racine, nœud interne, nœud externe ou **feuille** .
- **Etiquette** d'un nœud.
- **Sous-arbre**.

- **Chemin d'un nœud  $X$  à la racine** : c'est la suite de nœuds  $N_1, N_2, \dots, X$  de l'arbre tels que  $N_i$  est le père de  $N_{i+1}$  ( $N_1$  est la racine de l'arbre).

- **Chemin d'un nœud  $X$  à la racine** : c'est la suite de nœuds  $N_1, N_2, \dots, X$  de l'arbre tels que  $N_i$  est le père de  $N_{i+1}$  ( $N_1$  est la racine de l'arbre).
- **Longueur d'un chemin** : nombre de liens de parenté dans le chemin considéré.

- **Chemin d'un nœud  $X$  à la racine** : c'est la suite de nœuds  $N_1, N_2, \dots, X$  de l'arbre tels que  $N_i$  est le père de  $N_{i+1}$  ( $N_1$  est la racine de l'arbre).
- **Longueur d'un chemin** : nombre de liens de parenté dans le chemin considéré.
- **Profondeur d'un nœud** : longueur du chemin du nœud considéré. La profondeur peut se calculer (ou se définir) récursivement
  - 1 profondeur(racine) = 0.
  - 2 profondeur( $X$ ) = 1 + profondeur (pere( $X$ )).

- **Chemin d'un nœud  $X$  à la racine** : c'est la suite de nœuds  $N_1, N_2, \dots, X$  de l'arbre tels que  $N_i$  est le père de  $N_{i+1}$  ( $N_1$  est la racine de l'arbre).
- **Longueur d'un chemin** : nombre de liens de parenté dans le chemin considéré.
- **Profondeur d'un nœud** : longueur du chemin du nœud considéré. La profondeur peut se calculer (ou se définir) récursivement
  - 1 profondeur(racine) = 0.
  - 2 profondeur( $X$ ) = 1 + profondeur(père( $X$ )).
- **Degré d'un nœud  $X$**  : c'est le nombre de fils de  $X$ .

- **Taille d'un arbre** : c'est le nombre de nœuds de l'arbre. La taille d'un arbre peut se calculer récursivement :

- 1 Si l'arbre est vide alors

$$\text{taille}(\text{vide}) = 0$$

- 2 sinon

$$\text{taille}(\text{arbre}) = 1 + \sum_{s \in \{\text{sous arbres}\}} \text{taille}(s) .$$

- **Taille d'un arbre** : c'est le nombre de nœuds de l'arbre. La taille d'un arbre peut se calculer récursivement :

- 1 Si l'arbre est vide alors

$$\text{taille}(\text{vide}) = 0$$

- 2 sinon

$$\text{taille}(\text{arbre}) = 1 + \sum_{s \in \{\text{sous arbres}\}} \text{taille}(s).$$

- **Hauteur d'un arbre** : La plus grande profondeur de l'ensemble des nœuds. La hauteur peut aussi se calculer récursivement (**Exercice!**).

- **Arbre de degré  $n$**  : arbre dont tous les nœuds sont au maximum de degré  $N$ . Il y a des cas particuliers les **arbres binaires** (degré 2) et les arbres de **degré 1** (listes). Dans le cas général, le degré **n'est pas connu** (n'est pas fixé d'avance!).
- **Niveau d'un arbre** : ensemble des nœuds de même profondeur.

On peut représenter une forêt par un tableau tel que

$$\text{père}[s] = \text{père de } s$$

(si  $s$  est une **racine**,  $\text{père}[s] = s$ .)

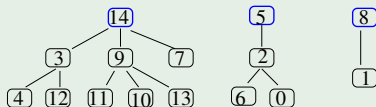
# Représentation par tableau

On peut représenter une forêt par un tableau tel que

père[s] = père de s

(si s est une **racine**, père[s] = s.)

## Schéma



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	INDICES
2	8	2	14	3	5	2	14	8	14	9	9	3	9	14	CONTENUS

## Exemples abstraits

La liste n'est pas exhaustive

- Etudier/analyser la structure d'une expression (compilation)
- Cas particulier : les expressions arithmétiques.
- Arbre lexicographique pour classer/stocker des mots par ordre alphabétique (cf. **exemple**).

## Exemples abstraits

La liste n'est pas exhaustive

- Etudier/analyser la structure d'une expression (compilation)
- Cas particulier : les expressions arithmétiques.
- Arbre lexicographique pour classer/stocker des mots par ordre alphabétique (cf. **exemple**).

## Un exemple qui nous concerne

Hiérarchie d'héritage en JAVA.

## Une opération classique

- On désire appliquer le même traitement à **tous les nœuds** d'un arbre.
- Un cas particulier concerne l' **affichage** .

## Une opération classique

- On désire appliquer le même traitement à **tous les nœuds** d'un arbre.
- Un cas particulier concerne l' **affichage** .

## Type de parcours

L' **ordre** de **visite** des nœuds de l'arbre est important.

- 1 parcours en profondeur **préfixe**, **postfixe** et **infixe**.
- 2 parcours en largeur.

- **En largeur**: parcours par niveau de l'arbre et de gauche à droite à l'intérieur de chaque niveau.

- **En largeur**: parcours par niveau de l'arbre et de gauche à droite à l'intérieur de chaque niveau.
- **En profondeur**:
  - Exporation **réursive par sous-arbres** des nœuds de l'arbre.
  - Selon l'ordre des sous-arbres, on a deux cas:  
**parcours en profondeur de gauche à droite** ou  
**parcours en profondeur de droite à gauche**

- **En largeur**: parcours par niveau de l'arbre et de gauche à droite à l'intérieur de chaque niveau.
- **En profondeur**:
  - Exorption **réursive par sous-arbres** des nœuds de l'arbre.
  - Selon l'ordre des sous-arbres, on a deux cas:  
**parcours en profondeur de gauche à droite** ou  
**parcours en profondeur de droite à gauche**
- Selon l'ordre de **traitement d'un nœud père** par rapport à ses fils, on parle de parcours
  - **préfixe** : le père avant ses fils
  - **postfixe** : le père après ses fils
  - **infixe** : le père entre deux fils (notamment pour les arbres binaires)

## Arbre particulier

**Définition** : arbres de degré 2.

**Spécifique**. On parle de sous-arbre gauche et de sous-arbre droit ou encore de fils gauche, fils droit.

## Arbre particulier

**Définition** : arbres de degré 2.

**Spécifique**. On parle de sous-arbre gauche et de sous-arbre droit ou encore de fils gauche, fils droit.

## Correspondance entre arbre binaire et arbre quelconque

Un arbre non-binaire peut toujours être représenté par un arbre binaire (et **vice-versa**). En effet, il existe une **bijection** entre un arbre binaire avec  $n$  feuilles et un arbre quelconque avec  $n$  nœuds.

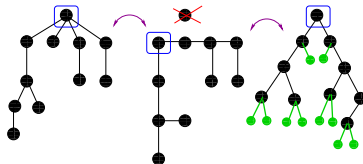
## Arbre particulier

**Définition** : arbres de degré 2.

**Spécifique**. On parle de sous-arbre gauche et de sous-arbre droit ou encore de fils gauche, fils droit.

## Correspondance entre arbre binaire et arbre quelconque

Un arbre non-binaire peut toujours être représenté par un arbre binaire (et **vice-versa**). En effet, il existe une **bijection** entre un arbre binaire avec  $n$  feuilles et un arbre quelconque avec  $n$  nœuds.





## Définition du type Arbres Binaires

- **Nom du type** : ArbreBin
- **Opérateurs** (signature des méthodes)
  - Création:  $\rightarrow$  ArbreBin
  - racine: ArbreBin  $\rightarrow$  Noeud
  - sag: ArbreBin  $\rightarrow$  ArbreBin
  - sad: ArbreBin  $\rightarrow$  ArbreBin
  - cons: Noeud x ArbreBin x ArbreBin  $\rightarrow$  ArbreBin
  - estVide: ArbreBin  $\rightarrow$  Booléen
  - info: Noeud  $\rightarrow$  Contenu
- **Propriétés des opérateurs**:
  - préconditions** :  
(a) racine(A) valide si  $A \neq \text{vide}$ , (b) sag(A) valide si  $A \neq \text{vide}$  et (c) sad(A) valide si  $A \neq \text{vide}$ .
  - axiomes** :  
 $\forall \text{ rac (noeud)}, \forall \text{ sg (ArbreBin)} \text{ et } \forall \text{ sd (ArbreBin)}$  alors (i)  $\text{racine(cons(rac,sg,sd))}=\text{rac}$ , (ii)  $\text{sag(cons(rac,sg,sd))}=\text{sg}$  et (iii)  $\text{sd(cons(rac,sg,sd))}=\text{sd}$ .

## Algorithme du parcours en profondeur (gauche-droite) d'un arbre binaire

Dans ce qui suit, **arb** est du type `ArbreBin` et **traiter1** , **traiter2** et **traiter3** sont effectuent une opération sur les contenus des nœuds.

# Algorithme du parcours en profondeur (gauche-droite) d'un arbre binaire

Dans ce qui suit, **arb** est du type **ArbreBin** et **traiter1**, **traiter2** et **traiter3** sont effectués une opération sur les contenus des nœuds.

## Parcours

```
parcourir (arb)
    traiter1(info(racine(arb)))
    Si sag  $\neq$  vide alors
        parcourir (sag(arb));
    traiter2(info(racine(arb)))
    Si sad  $\neq$  vide alors
        parcourir (sad(arb));
    traiter3(info(racine(arb)))
```



## Parcours

```
parcourir (arb)
    traiter1(info(racine(arb)))
    Si sag  $\neq$  vide alors
        parcourir (sag(arb));
    Si sad  $\neq$  vide alors
        parcourir (sad(arb));
```



## Parcours

```
parcourir (arb)
  Si sag  $\neq$  vide alors
    parcourir (sag(arb));
  Si sad  $\neq$  vide alors
    parcourir (sad(arb));
  traiter1(info(racine(arb)))
```

# Algorithme du parcours en profondeur (gauche-droite) infixe

## Parcours

```
parcourir (arb)
    Si sag  $\neq$  vide alors
        parcourir (sag(arb));
    traiter1(info(racine(arb)))
    Si sad  $\neq$  vide alors
        parcourir (sad(arb));
```

## Algorithme du parcours en largeur d'un arbre binaire

On utilise une **file** qu'on appelle ci-dessous **fifo** .

# Algorithme du parcours en largeur d'un arbre binaire

On utilise une **file** qu'on appelle ci-dessous **fifo** .

## ParcoursLargeur

```
parcourirlargeur (arb)
    ajouter arb dans fifo ;
    Tant que fifo  $\neq$  vide faire
        prem := prochain élément de fifo;
        traiter1((racine(prem)));
        Si sag  $\neq$  vide alors
            ajouter sag(prem) dans fifo
        Si sad  $\neq$  vide alors
            ajouter sad(prem) dans fifo
    FinTantQue
Finparcourirlargeur.
```

**L'ordre est souvent crucial en algorithmique**

Ce sont des arbres binaires **ordonnés**

## L'ordre est souvent crucial en algorithmique

Ce sont des arbres binaires **ordonnés**. Rappelons-nous que les tableaux déjà triés offrent certains avantages. Idem pour les listes.

## Définition

Un **arbre binaire de recherche** est un arbre binaire tel que

- les valeurs des nœuds du **sous-arbre gauche** de **tout nœud X** de l'arbre sont **inférieures** à X.
- les valeurs des nœuds du **sous-arbre droit** de **tout nœud X** de l'arbre sont **supérieurs** à X.

Dans l'algorithme ci-dessous, **arb** est un ArbreBin (de recherche) et le résultat est le sous-arbre dont l'élément recherché est la racine.

Dans l'algorithme ci-dessous, **arb** est un ArbreBin (de recherche) et le résultat est le sous-arbre dont l'élément recherché est la racine.

### Recherche

**recherche** (val, arb)

/★ conditions d'arrêt ★/

**si** arb = vide **alors** retourner(vide);

**si** val = info(racine(arb)) **alors**  
retourner(arb);

/★ corps de la récursion ★/

**si** val < info(racine(arb)) **alors**  
retourner( **recherche** (val, sag(arb)));

**sinon**  
retourner( **recherche** (val, sad(arb)));

**Finrecherche.**

On désire rajouter un élément **nouveau** qui est un **nœud**.

## Ajout d'un élément

On désire rajouter un élément **nouveau** qui est un **nœud**.

### Ajout

**ajout** (val, arb)

/\* on crée le nœud \*/

**nouveau** = new Noeud(val);

/\* conditions d'arrêt \*/

**si** arb = vide **alors**

retourner(cons(nouveau, vide, vide));

/\* corps de la récursion \*/

**si** val < info(racine(arb)) **alors**

**si**(sag(arb) = vide) **alors**

insérer **nouveau** comme nouvel sous arbre gauche

**sinon** retourner( **ajout** (val, sag(arb)));

**sinon si** (sad(arb)=vide) **alors**

insérer **nouveau** comme nouvel sous arbre gauche

**sinon** retourner( **ajout** (val, sad(arb)));

**Finajout.**



## Trois étapes

- 1 Trouver l'élément à supprimer.
- 2 Supprimer le nœud correspondant.
- 3 Réorganiser (respecter l'ordre).