

# I202B, Les arbres + la récursivité

J. Vander Meulen   C. Damas

Mars 2018

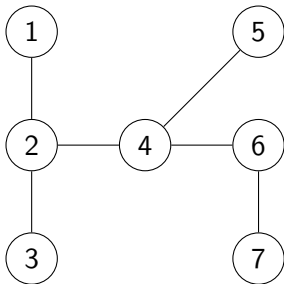
Qu'est ce qu'un arbre ?

Représentation informatique des arbres

Algorithmes

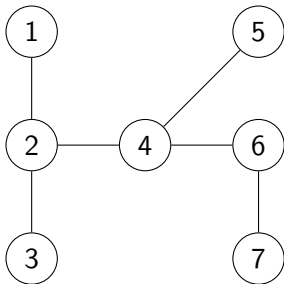
## Un arbre (non orienté) est un graphe particulier

- Un graphe non vide, non orienté, acyclique et connexe.



## Un arbre (non orienté) est un graphe particulier

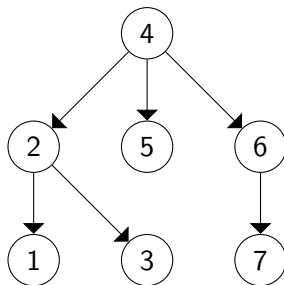
- Un graphe non vide, non orienté, acyclique et connexe.



- Un unique chemin d'un noeud à un autre

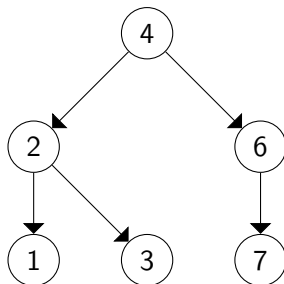
# Un arbre (orienté) est aussi un graphe particulier

- Un graphe non vide, acyclique, ayant une unique racine et tel que tous les nœuds sauf la racine ont un unique parent

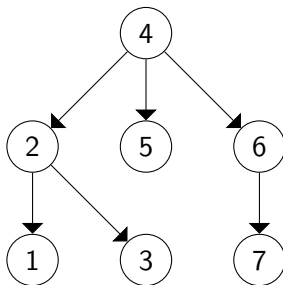


# Un arbre binaire

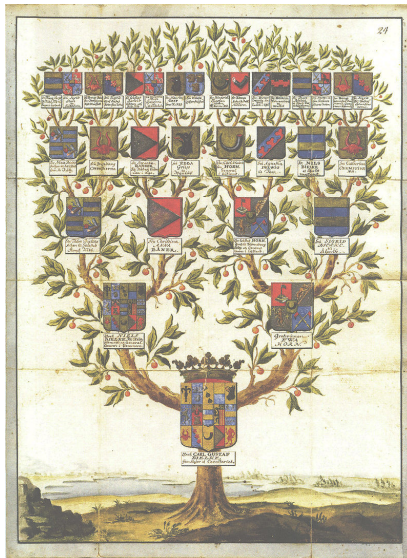
- Un arbre dont chaque noeud a au plus deux noeuds adjacents (souvent appelés fils gauche et fils droit)



Tous les arbres ne sont pas binaires

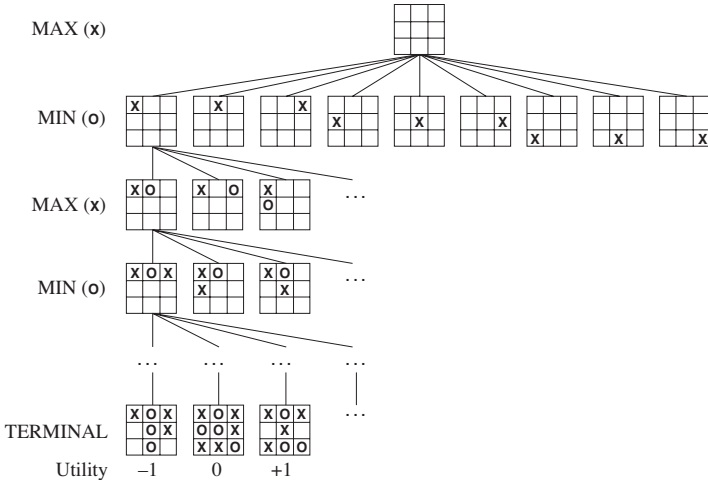


# Exemples 1 : les arbres généalogiques



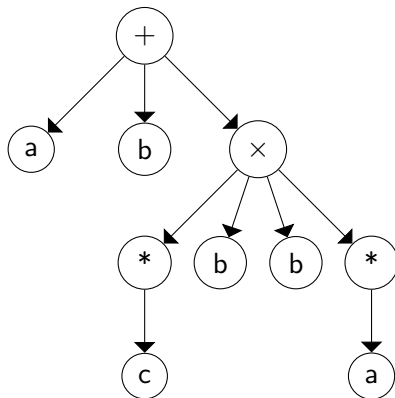


## Exemples 2 : les arbres en IA

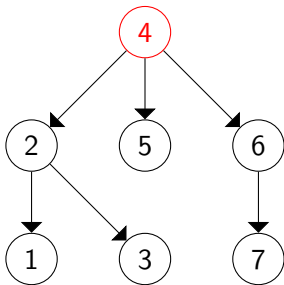


## Exemples 3 : les arbres dans les compilateurs

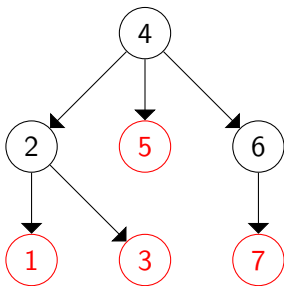
- Une expression :  $a|b|c^*bba^*$



## Terminologie : racine (anglais : root)

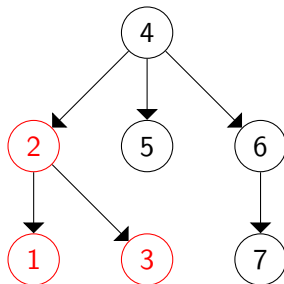


Terminologie : feuilles (anglais : leaves) (🍃)



## Terminologie : parent-enfant (anglais : parent-children)

- 2 est le parent de 1 et 3
- 1 et 3 sont les enfants de 2



# Une définition récursive des arbres

- Pour pouvoir écrire facilement des algorithmes récursifs sur les arbres, on va considérer une deuxième définition des arbres orientés
- Les deux définitions sont équivalentes
- L'ensemble des arbres qu'on peut définir avec les deux définitions sont les mêmes

# Une définition récursive des arbres

- Dans un premier temps, on va définir les plus petits arbres possibles (les arbres composés d'un unique noeud)
- Dans un second temps, on va définir des arbres de plus en plus grands (les arbres composés de plus d'un noeud)

Premier temps : soit un ensemble  $E$ , définition d'un arbre de hauteur 0

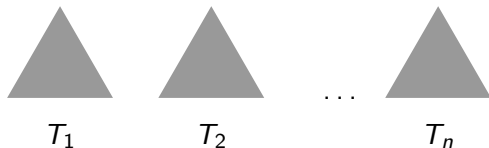
Une racine avec un label  $e \in E$  est un arbre





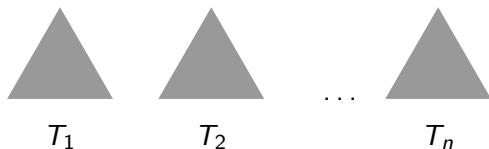
Second temps : soit un ensemble  $E$ , définition d'un arbre de hauteur  $> 0$

- Supposons  $n$  arbres  $T_1, T_2, \dots, T_n$  :

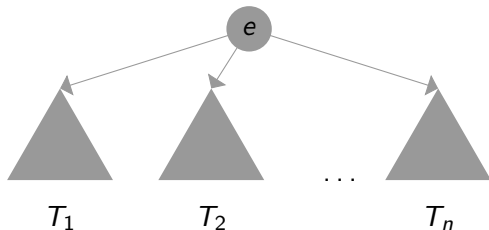


Second temps : soit un ensemble  $E$ , définition d'un arbre de hauteur  $> 0$

- Supposons  $n$  arbres  $T_1, T_2, \dots, T_n$  :

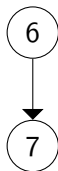
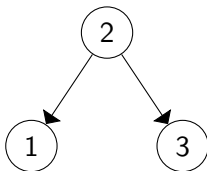


- Le graphe suivant, où  $e \in E$ , est aussi un arbre valide :



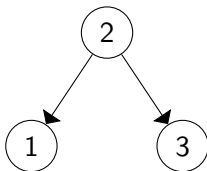
## Second temps : exemple

- Supposons les 3 arbres :

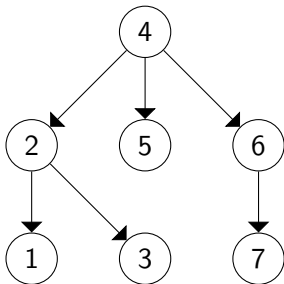


## Second temps : exemple

- Supposons les 3 arbres :



- On peut construire l'arbre suivant :



Qu'est ce qu'un arbre ?

Représentation informatique des arbres

Algorithmes

# Une représentation orientée POO (application pratique de la définition récursive)

- Une classe Tree
- 3 attributs :
  - Une valeur
  - Une référence vers son parent
  - Un conteneur contenant des références vers ses fils
- 2 constructeurs :
  - Un destiné aux arbres de hauteur 0
  - Un autre destiné aux arbres de hauteur  $> 0$

## Classe Tree

```
public class Tree {  
    private int value;  
    private Tree parent;  
    private Tree[] children;  
  
    public Tree(int v, Tree[] chd) {  
        value = v;  
        children = chd;  
  
        int i = 0;  
        while(i != chd.length) {  
            chd[i].parent = this;  
            i++;  
        }  
  
        ...  
    }  
}
```

## Classe Tree

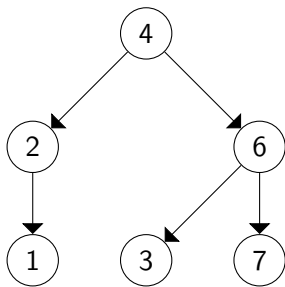
```
public class Tree {  
    ...  
  
    public Tree(int v) {  
        this(v, new Tree[0]);  
    }  
  
    ...  
}
```



## Créer un arbre

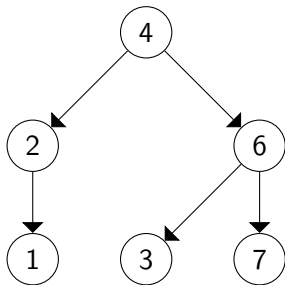
```
public class Main{  
    public static void main(String[] args){  
        Tree l1 = new Tree(1);  
        Tree l3 = new Tree(3);  
        Tree l5 = new Tree(5);  
        Tree l7 = new Tree(7);  
  
        Tree t2 = new Tree(2, new Tree[]{l1, l3});  
        Tree t6 = new Tree(6, new Tree[]{l7});  
  
        Tree t4 = new Tree(4, new Tree[]{t2, l5, t6});  
    }  
}
```

## Une représentation d'arbres binaires avec un tableau



valeur	4	2	1	6	3	7	0
fils gauche	1	2	-1	4	-1	-1	1
fils droit	3	1	-1	5	-1	-1	2
	0	1	2	3	4	5	

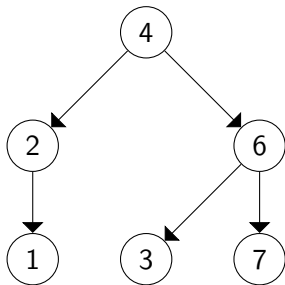
## Une représentation d'arbres binaires avec un tableau



valeur	4	2	1	6	3	7	0
fils gauche	1	2	-1	4	-1	-1	1
fils droit	3	1	-1	5	-1	-1	2

0 1 2 3 4 5

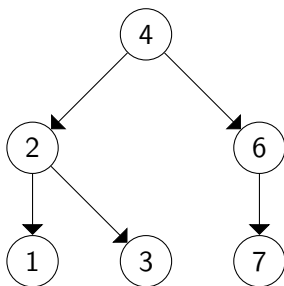
## Une représentation d'arbres binaires avec un tableau



valeur	4	2	1	6	3	7	0
fils gauche	1	2	-1	4	-1	-1	1
fils droit	3	1	-1	5	-1	-1	2

0 1 2 3 4 5

## Une représentation d'arbres binaires complets avec un tableau



valeur

?	4	2	6	1	3	7
0	1	2	3	4	5	6

Qu'est ce qu'un arbre ?

Représentation informatique des arbres

Algorithmes

# Les algorithmes récursifs : intuitions

Pour résoudre un problème, on considère généralement différents cas :

- cas simples :
  - par exemple, les arbres de hauteur 0
  - il existe généralement un algo trivial pour ces instances

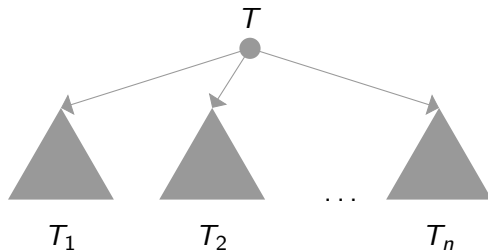
# Les algorithmes récursifs : intuitions

Pour résoudre un problème, on considère généralement différents cas :

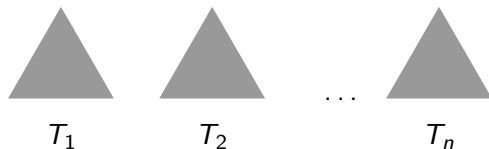
- cas simples :
  - par exemple, les arbres de hauteur 0
  - il existe généralement un algo trivial pour ces instances
- cas plus complexes :
  - par exemple, les arbres de hauteur  $> 0$
  - avant de traiter ces instances, on va considérer des instances plus petites.



Avant de traiter tout cet arbre




On va traiter uniquement ses sous-arbres  
(de manière récursive)




Remarque : les sous-arbres  $T_1, T_2, \dots, T_n$  sont plus petits que l'arbre initial

## Un 1<sup>er</sup> exemple : # d'un arbre



- Écrire un algorithme récursif qui renvoie le # d'un arbre

```
public static int nbrLeaves(Tree t)
```

## Un 1<sup>er</sup> exemple : # d'un arbre

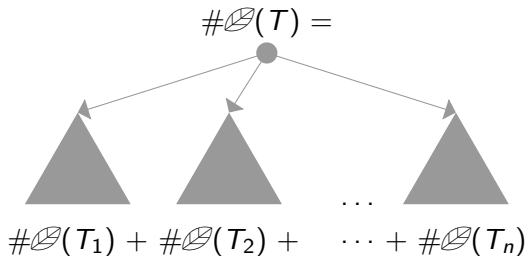
- Écrire un algorithme récursif qui renvoie le # d'un arbre

```
public static int nbrLeaves(Tree t)
```

- Cas simple : le # d'une  = 1




$\# \text{ (leaf icon) }$  d'un arbre de hauteur  $> 0$



## #🍃 : code Java


```
public static int nbrLeaves(Tree t) {  
    int r;  
    if (t.children.length == 0) {  
        r = 1;  
    } else {  
        r = 0;  
        int i = 0;  
        while (i != t.children.length) {  
            r += nbrLeaves(t.children[i]);  
            i++;  
        }  
    }  
    return r;  
}
```

## Un 2<sup>e</sup> exemple : aplanir les d'un arbre


- Écrire un algorithme récursif qui renvoie un tableau contenant les  d'un arbre

```
public static Tree[] flattenLeaves(Tree t)
```

## Un 2<sup>e</sup> exemple : aplanir les d'un arbre

- Écrire un algorithme récursif qui renvoie un tableau contenant les  d'un arbre

```
public static Tree[] flattenLeaves(Tree t)
```

- Pour résoudre ce problème on va construire un algorithme plus général
  - On place les  d'un arbre dans **une partie** de tableau

```
static int flattenLeaves(Tree t, Tree[] a, int idx)
```





## Un 2<sup>e</sup> exemple : code Java

```
static int flattenLeaves(Tree t, Tree[] a, int idx) {
    int r;
    if (t.children.length == 0) {
        a[idx] = t;
        r = 1;
    } else {
        r = 0;
        int i = 0;
        while (i != t.children.length) {
            r += flattenLeaves(t.children[i], a, idx + r);
            i++;
        }
    }
    return r;
}
```

## Un 2<sup>e</sup> exemple : code Java

```
static int flattenLeaves(Tree t) {  
    int nl = nbrLeaves(t);  
    Tree[] r = new Tree[nl];  
    flattenLeaves(t, r, 0);  
    return r;  
}
```

## Un 3<sup>e</sup> exemple : imprimer le chemin vers la racine

- Tous les algorithmes traitant des arbres ne sont pas obligatoirement récurifs

```
public static void pathV1(Tree t) {  
    System.out.println(t.value);  
    if (t.parent != null) {  
        pathV1(t.parent);  
    }  
}
```

```
public static void pathV2(Tree t) {  
    while(t != null) {  
        System.out.println(t.value);  
        t = t.parent;  
    }  
}
```