Chapitre 2: Arbres

Christophe Morvan

Université de Marne-la-Vallée

15 septembre 2015

Plan

Progression

Observation

Structures non-linéaires

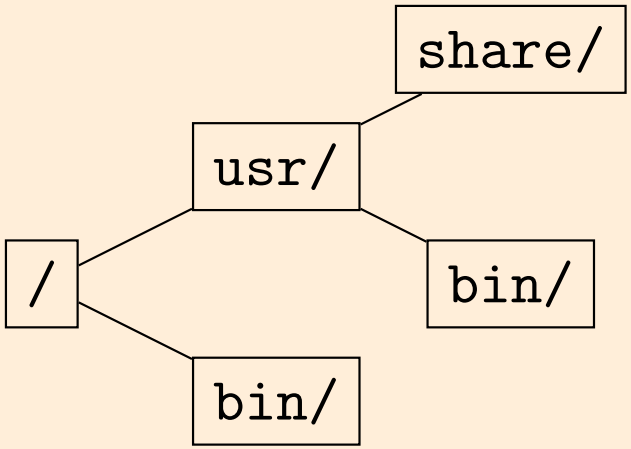
- Arbres généalogiques
- Systèmes de fichiers
- Document structuré
- Autres...

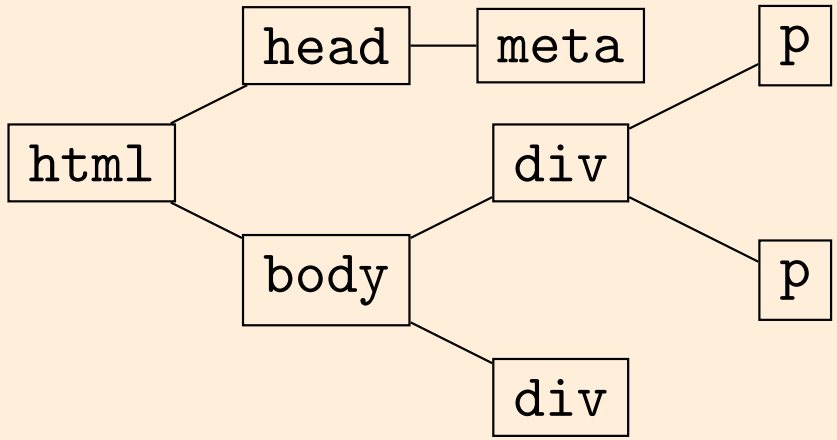
Observation

Structures non-linéaires

- Arbres généalogiques
- Systèmes de fichiers
- Document structuré
- Autres...

Exemple





Exemples

XML

Extensible Markup Language, langage de balisage extensible (en Français), est un langage de balisage *générique* héritier de SGML. La syntaxe est dite « extensible » car elle permet de définir des espaces de noms, c'est-à-dire des dialectes avec chacun leur vocabulaire et leur grammaire.
Exemple : XHTML, XSLT, RSS, SVG.

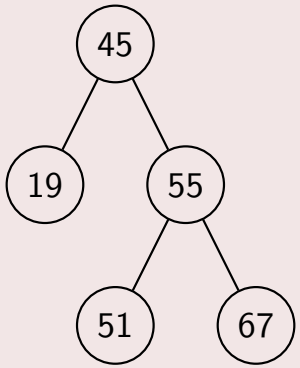
Exemples

XML

Extensible Markup Language, langage de balisage extensible (en Français), est un langage de balisage *générique* héritier de SGML. La syntaxe est dite « extensible » car elle permet de définir des espaces de noms, c'est-à-dire des dialectes avec chacun leur vocabulaire et leur grammaire.
Exemple : XHTML, XSLT, RSS, SVG.

Arbres binaires de recherche

Les arbres de recherches sont organisés de façon à ce que l'étiquette chaque nœud soit supérieure à celles contenues dans son sous-arbre gauche et inférieure à celles contenues dans le sous-arbre droit.



Progression

Définitions

Soit $\mathtt{T}$ un type quelconque.

Définition récursive

Un arbre Γ de $\mathtt{T}$ est composé d'un ou plusieurs *nœuds*, de type $\mathtt{T}$, de sorte que :

1) Il y a un unique nœud appelé *racine*, noté **racine**(Γ).
2) Les nœuds distincts de **racine**(Γ) peuvent être groupés en $m \geqslant 0$ ensembles disjoints constituant des arbres. Chacun de ces ensembles est un *sous-arbre* de **racine**(Γ).

Définitions

Soit T un type quelconque.

Définition récursive

Un arbre Γ de T est composé d'un ou plusieurs *nœuds*, de type T, de sorte que :

1) Il y a un unique nœud appelé *racine*, noté **racine**(Γ).
2) Les nœuds distincts de **racine**(Γ) peuvent être groupés en $m \geqslant 0$ ensembles disjoints constituant des arbres. Chacun de ces ensembles est un *sous-arbre* de **racine**(Γ).

Remarque

Cette définition est récursive, mais non-circulaire, puisque les sous-arbres contiennent systématiquement au moins 1 nœud de moins que l'arbre dont ils sont issus.

Vocabulaire

Quelques définitions

- Nœud interne : c'est un nœud qui possède au moins un sous-arbre ;
- Feuille : il s'agit d'un nœud qui ne possède aucun sous-arbre ;
- Parent : à part la racine, tous les nœuds d'un arbre possèdent exactement un parent. Il s'agit de la racine de l'arbre dont ils sont la racine d'un des sous-arbres.
- Enfant : les nœuds internes d'un arbre possèdent des enfants qui sont les racines de leurs sous-arbres.
- Ascendant/Descendant : Les ascendants (*resp.* descendants) est l'ensemble constitué du parent (*resp.* des enfants) d'un nœud ainsi que leurs ascendants (*resp.* descendants).

Arbres binaires

Définition

Un *arbre binaire* de $\mathtt{T}$ est un ensemble de nœuds qui est soit vide soit composé d'une racine avec deux arbres binaires de $\mathtt{T}$ disjoints, appelés sous-arbres *droit* et *gauche*.

Arbres binaires

Définition

Un *arbre binaire* de $\mathtt{T}$ est un ensemble de nœuds qui est soit vide soit composé d'une racine avec deux arbres binaires de $\mathtt{T}$ disjoints, appelés sous-arbres *droit* et *gauche*.

Observation

Un arbre binaire, n'est pas un arbre au sens de la définition donnée sur le transparent 7

Arbres binaires

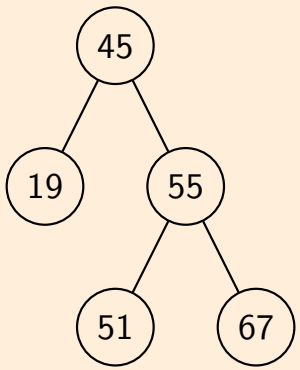
Définition

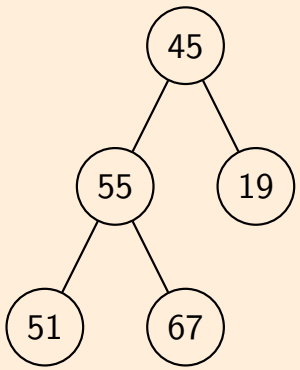
Un *arbre binaire* de T est un ensemble de nœuds qui est soit vide soit composé d'une racine avec deux arbres binaires de T disjoints, appelés sous-arbres *droit* et *gauche*.

Observation

Un arbre binaire, n'est pas un arbre au sens de la définition donnée sur le transparent 7

Exemple





Arbres binaires

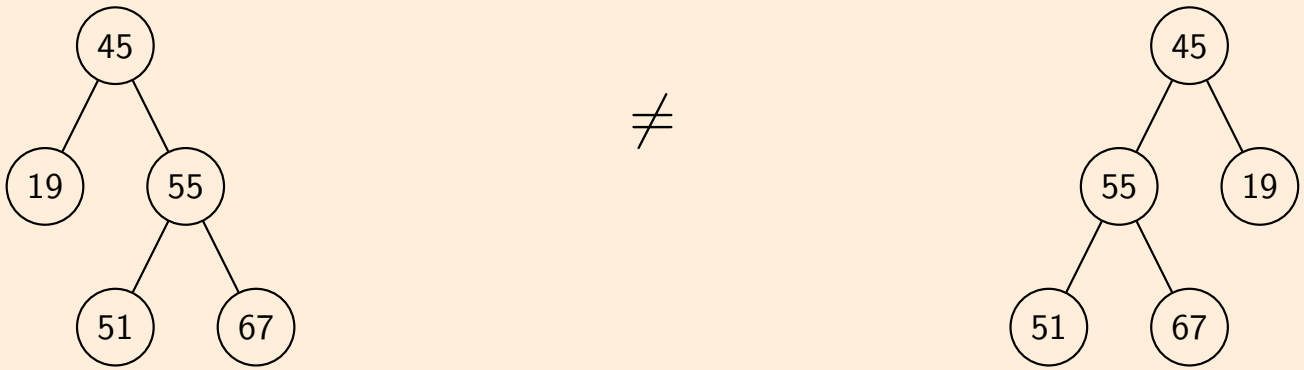
Définition

Un *arbre binaire* de T est un ensemble de nœuds qui est soit vide soit composé d'une racine avec deux arbres binaires de T disjoints, appelés sous-arbres *droit* et *gauche*.

Observation

Un arbre binaire, n'est pas un arbre au sens de la définition donnée sur le transparent 7

Exemple



Vocabulaire – 2

Quelques définitions supplémentaires

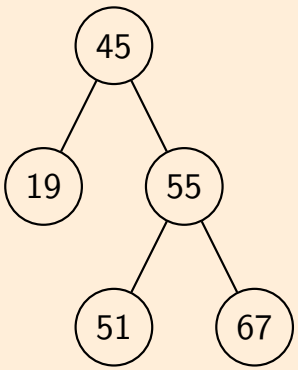
- Chemin : un chemin est une suite de nœuds d'un arbre (ordinaire ou binaire) qui part de la racine et qui abouti à une feuille, et passe de parent en enfant sans interruption.
- Profondeur : la profondeur d'un nœud est le nombre de ses ancêtres
- Hauteur : la hauteur d'un arbre est le nombre de nœuds du chemin le plus long moins 1. C'est aussi la plus grande profondeur.
- Parcours : énumération des nœuds d'un arbre.

Vocabulaire – 2

Quelques définitions supplémentaires

- Chemin : un chemin est une suite de nœuds d'un arbre (ordinaire ou binaire) qui part de la racine et qui abouti à une feuille, et passe de parent en enfant sans interruption.
- Profondeur : la profondeur d'un nœud est le nombre de ses ancêtres
- Hauteur : la hauteur d'un arbre est le nombre de nœuds du chemin le plus long moins 1. C'est aussi la plus grande profondeur.
- Parcours : énumération des nœuds d'un arbre.

Exemple



- Profondeur de 55 : 1
- Hauteur : 2
- Parcours préfixe : 45, 19, 55, 51, 67.

Arbres binaires : interface récursive

BinT<T>

- `static nil` : arbre vide
- `static BinT<T> tree (T elem , BinT<T> left, BinT<T> right)` : construit l'arbre binaire de racine `elem` avec comme sous-arbres gauche et droit les autres paramètres.
- `T root()` : fournit comme résultat la racine de l'arbre courant.
- `BinT<T> left ()` : fournit comme résultat le sous-arbre gauche de l'arbre binaire courant.
- `BinT<T> right ()` : fournit comme résultat le sous-arbre droit de l'arbre binaire courant.

```
BinT<Integer> l_1=BinT.nil(), l_2=BinT.tree(67, l_1, l_1);
l_2 = BinT.tree(55, l_2,l_1);
int val = 17 + l_2.root());
```

Exemple – Parcours préfixe

Programme exemple

```
void preOTrav(BinT<T> tree){
    /* Parcours prefixe d'un arbre binaire */
    if (tree!= BinT.nil()){ /* Est-il non-vide */
        System.out.println(tree.root());
        preOTrav(tree.left());
        preOTrav(tree.right());
    }
    /* Si l'arbre est vide on n'affiche rien */
}
```

Progression

Implémentation – 1

Il existe de multiples implémentations des arbres.
Java en possède une dans Swing

Programme exemple

```
import java.util.Queue;  /* Ces imports sont utilisés dans la suite */
import java.util.List;
import java.util.LinkedList;
import java.util.ArrayList;
public class Tree<T> {
    private class Node<F>{ /* Classe interne */
        F value;
        List<Tree<F>> children; /* Il s'agit d'arbre "ordinaires" */
        Node(F value){
            this.value=value;
            children=new ArrayList<Tree<F>>(); }
        Node(F value, List<Tree<F>> children){
            this(value);
            this.children=children; }}}
```

Implémentation – 2

Programme exemple

```
public class Tree<T> {
    Node<T> root;
    public Tree(T value){
        /* Arbre avec seulement une racine */
        root=new Node(value);  }
    public Tree(T value, List<Tree<T>> children){
        this(value);
        root.children=children;
        /* Attention : il ne s'agit pas d'une recopie de la liste */}
    public T getValue(){
        return root.value; }
    public List<Tree<T>> getChildren(){
        return root.children; }
    public Tree<T> getIChildren(int index){
        return root.children.get(index); }
    public String toString (){
        return root.value.toString()+" -> "+root.children.toString();
    }}
```

Notes sur l'implémentation

Remarques

- C'est une implémentation ultra simple
- Il n'y a pas de méthode pour ajouter un nœud unique *dans* l'arbre
- Il est très simple d'ajouter un sous-arbre à un nœud
- Il faut être très vigilant avec les listes (modifier une des listes modifie l'arbre)

Notes sur l'implémentation

Remarques

- C'est une implémentation ultra simple
- Il n'y a pas de méthode pour ajouter un nœud unique *dans* l'arbre
- Il est très simple d'ajouter un sous-arbre à un nœud
- Il faut être très vigilant avec les listes (modifier une des listes modifie l'arbre)

Autres implémentations

Dans la suite nous examinerons d'autres implémentations

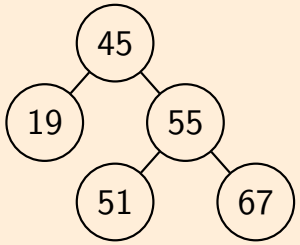
- Arbres binaire
- Tas
- Arbres binaire de recherche
- Arbres rouge-noir (TP)

La plupart permettent une insertion *efficace*.

Progression

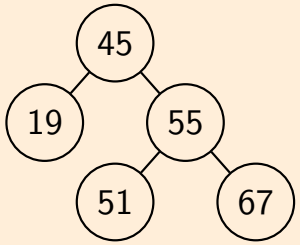
Parcours d'arbres

Exemple



Parcours d'arbres

Exemple

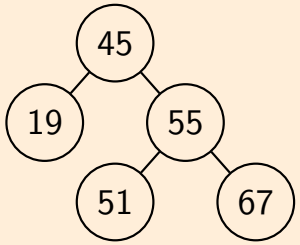


Parcours en largeur

- Parcours tous les fils, puis itérer sur leurs fils
- Sur l'exemple : $45, 19, 55, 51, 67$
- S'appuie sur une file

Parcours d'arbres

Exemple



Parcours en largeur

- Parcours tous les fils, puis itérer sur leurs fils
- Sur l'exemple : $45, 19, 55, 51, 67$
- S'appuie sur une file

Parcours en profondeur

La description est pour les arbres binaire elle peut s'adapter pour un arbre quelconque (sauf pour l'ordre *infixe*). Repose sur une pile.

- Parcours préfixe : racine, parcours sous-arbre gauche puis parcours sous-arbre droit : $45, 19, 55, 51, 67$
- Parcours infixe : parcours sous-arbre gauche, racine puis parcours sous-arbre droit : $19, 45, 51, 55, 67$
- Parcours postfixe : parcours sous-arbre gauche, parcours sous-arbre droit puis racine : $19, 51, 67, 55, 45$

Parcours en largeur

Programme exemple

```
public class Tree<T> {
    public static <F> void breadthTraversal(Tree<F> tr){
        Queue<Tree<F>> visit=new LinkedList<Tree<F>>();
        /* Pourquoi LinkedList ? */
        visit.add(tr);
        while (!visit.isEmpty()){
            /* poll : retrait de l'element en tete de file */
            /* Pour simplifier on utilise a nouveau tr */
            tr = visit.poll();
            /* addAll : ajout des elemnts en fin de file */
            visit.addAll(t.getChildren());
            System.out.print(t.getValue()+" ");
        }}}
```

Les parcours en profondeur sont similaires, mais utilisent une pile... ou le font implicitement en s'appuyant sur la récursivité.

Progression

Utilisation des arbres

Tri et recherche

Les arbres peuvent être utilisés pour trier une structure linéaire (comme une liste ou un tableau). Le tas est un cas particulier très employé.
Les arbres permettent également de stocker, et de maintenir organisées des données pour la recherche.

Utilisation des arbres

Tri et recherche

Les arbres peuvent être utilisés pour trier une structure linéaire (comme une liste ou un tableau). Le tas est un cas particulier très employé.
Les arbres permettent également de stocker, et de maintenir organisées des données pour la recherche.

Calculs

Les arbres permettent également de représenter des expressions (arithmétiques ou autres) de façon à réaliser leur évaluation.

Utilisation des arbres

Tri et recherche

Les arbres peuvent être utilisés pour trier une structure linéaire (comme une liste ou un tableau). Le tas est un cas particulier très employé.
Les arbres permettent également de stocker, et de maintenir organisées des données pour la recherche.

Calculs

Les arbres permettent également de représenter des expressions (arithmétiques ou autres) de façon à réaliser leur évaluation.

Traduction

Les arbres permettent également de représenter des données de façon à faciliter leur analyse par ordinateur, ainsi que leur conversion d'un format vers l'autre.

Tas - 1

Définition

Le tas est un arbre binaire qui possède les propriétés suivantes :

- Chacun des nœud possède une valeur supérieure à chacun de ses enfants ;
- Il est complet à gauche : les feuilles sont sur au plus deux niveaux et les feuilles les plus profondes sont à gauche.

Utilisation

L'utilisation principale des tas est le tri : la valeur la plus grande est toujours à la racine du tas, et la profondeur est toujours logarithmique vis-à-vis du nombre d'éléments qu'il contient.
Il permet également d'implémenter des files de priorités puisque l'élément à la racine de l'arbre est toujours l'élément maximal, et que l'insertion est logarithmique.

Tas - 2

Implémentation

Lorsqu'on veut implémenter un tas, il faut pouvoir effectuer les deux opérations suivantes efficacement :

- Ajout d'une valeur
- Suppression de l'élément maximal (et assurer l'organisation du tas)

Tas - 2

Implémentation

Lorsqu'on veut implémenter un tas, il faut pouvoir effectuer les deux opérations suivantes efficacement :

- Ajout d'une valeur
- Suppression de l'élément maximal (et assurer l'organisation du tas)

Schéma classique

Dans la pratique un tas est, le plus souvent implémenté à l'aide d'un tableau. Tableau où les éléments sont placés de façon à ce que le fils gauche du nœud i a pour indice $2(i+1)-1$ et le fils droit $2(i+1)$.

Tas - 2

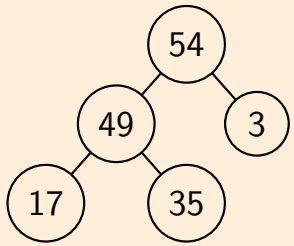
Implémentation

Lorsqu'on veut implémenter un tas, il faut pouvoir effectuer les deux opérations suivantes efficacement :

- Ajout d'une valeur
- Suppression de l'élément maximal (et assurer l'organisation du tas)

Schéma classique

Dans la pratique un tas est, le plus souvent implémenté à l'aide d'un tableau. Tableau où les éléments sont placés de façon à ce que le fils gauche du nœud i a pour indice $2(i+1)-1$ et le fils droit $2(i+1)$.



L'arbre se code ainsi :

0	1	2	3	4
54	49	3	17	35

Tas - 3

Ajout d'une valeur

1) On place la nouvelle valeur en dernière position dans le tableau
2) Si elle est plus grande que son parent $((i-1)/2)$, on échange les deux
3) Retour à l'étape 1).

Tas - 3

Ajout d'une valeur

1) On place la nouvelle valeur en dernière position dans le tableau
2) Si elle est plus grande que son parent $((i-1)/2)$, on échange les deux
3) Retour à l'étape 1).

Supression du maximum

1) Échange de la dernière feuille avec la racine
2) Supprimer la dernière feuille
3) Échanger avec le plus grand de ces deux fils jusqu'à ce que la valeur soit plus grande que ses deux fils

(Une fois la dernière feuille supprimée, pas de modification de la structure du tas.)

Exemples

Exemple

Ajout de la valeur 56 dans ce tableau :

0	1	2	3	4	5
54	49	3	17	35	

Exemples

Exemple

Ajout de la valeur 56 dans ce tableau :

0	1	2	3	4	5
54	49	3	17	35	
54	49	3	17	35	56

Exemples

Exemple

Ajout de la valeur 56 dans ce tableau :

0	1	2	3	4	5
54	49	3	17	35	
54	49	3	17	35	56
54	49	56	17	35	3

Exemples

Exemple

Ajout de la valeur 56 dans ce tableau :

0	1	2	3	4	5
54	49	3	17	35	
54	49	3	17	35	56
54	49	56	17	35	3
56	49	54	17	35	3

Exemples

Exemple

Ajout de la valeur 56 dans ce tableau :

0	1	2	3	4	5
54	49	3	17	35	
54	49	3	17	35	56
54	49	56	17	35	3
56	49	54	17	35	3

Exemple

Suppression du maximum dans ce tableau :

0	1	2	3	4	5
56	49	54	17	35	3

Exemples

Exemple

Ajout de la valeur 56 dans ce tableau :

0	1	2	3	4	5
54	49	3	17	35	
54	49	3	17	35	56
54	49	56	17	35	3
56	49	54	17	35	3

Exemple

Suppression du maximum dans ce tableau :

0	1	2	3	4	5
56	49	54	17	35	3
3	49	54	17	35	56

Exemples

Exemple

Ajout de la valeur 56 dans ce tableau :

0	1	2	3	4	5
54	49	3	17	35	
54	49	3	17	35	56
54	49	56	17	35	3
56	49	54	17	35	3

Exemple

Suppression du maximum dans ce tableau :

0	1	2	3	4	5
56	49	54	17	35	3
3	49	54	17	35	56
3	49	54	17	35	

Exemples

Exemple

Ajout de la valeur 56 dans ce tableau :

0	1	2	3	4	5
54	49	3	17	35	
54	49	3	17	35	56
54	49	56	17	35	3
56	49	54	17	35	3

Exemple

Suppression du maximum dans ce tableau :

0	1	2	3	4	5
56	49	54	17	35	3
3	49	54	17	35	56
3	49	54	17	35	
54	49	3	17	35	

Arbres binaires de recherche

Définition

Un arbre binaire de recherche est un arbre binaire qui possède les propriétés suivantes :

- La valeur de chaque nœud est supérieure à la racine de son sous-arbre gauche ;
- La valeur de chaque nœud est inférieure à la racine de son sous-arbre droit ;

Arbres binaires de recherche

Définition

Un arbre binaire de recherche est un arbre binaire qui possède les propriétés suivantes :

- La valeur de chaque nœud est supérieure à la racine de son sous-arbre gauche ;
- La valeur de chaque nœud est inférieure à la racine de son sous-arbre droit ;

Utilisation

Lorsque l'arbre binaire de recherche est *équilibré* la recherche y est logarithmique par rapport au nombre de nœuds.
Il existe des techniques pour maintenir un arbre binaire de recherche équilibré.
Un exemple performant en pratique : les arbres rouge-noir (TP).