

## STR2 : arbres

loig.jezequel@univ-nantes.fr

# Retour sur les structures de données déjà étudiées

## Structures non ordonnées

- ▶ L'ordre des éléments n'est pas spécifié, ou
- ▶ l'ordre des éléments n'a pas d'importance.

Exemple : ensembles

## Structures linéaires

- ▶ Ordre absolu sur les éléments,
- ▶ on accède aux éléments un à un dans cet ordre.

Exemples : tableaux, listes, piles, files

## Structures non-linéaires ?

Pour quoi faire ? Comment faire ?

# Besoin de structures non linéaires

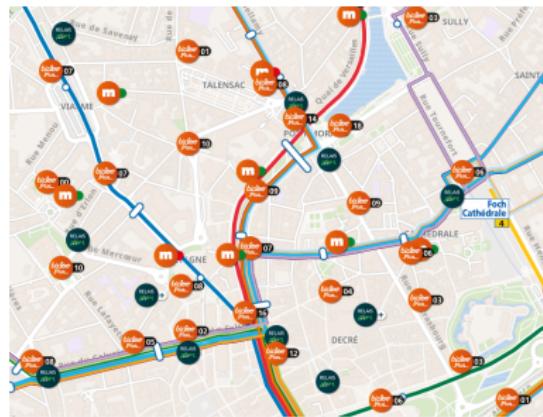


Figure: Recherche d'itinéraire en transports en commun

Et plein d'autres exemples...

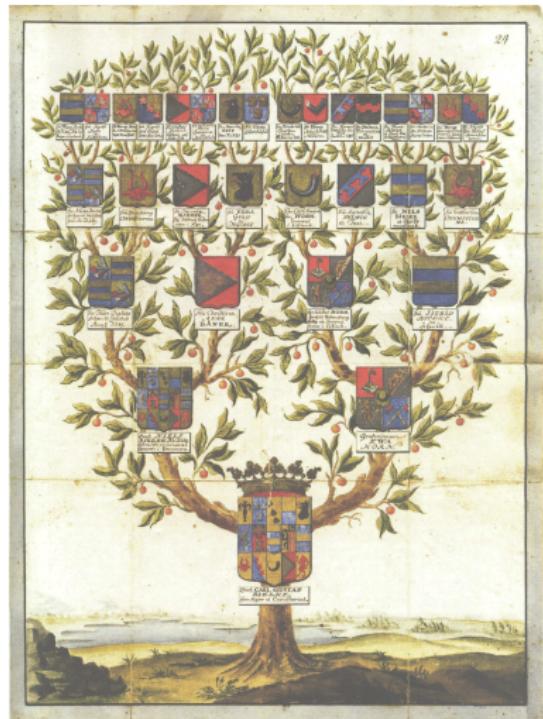
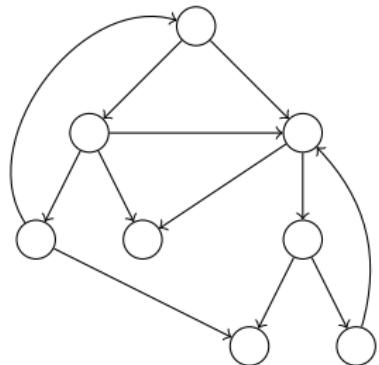


Figure: Représentation de liens familiaux

# Quelques types de structures non-linéaires

## Graphes

Intuition : n'importe quels éléments peuvent être reliés.



# Quelques types de structures non-linéaires

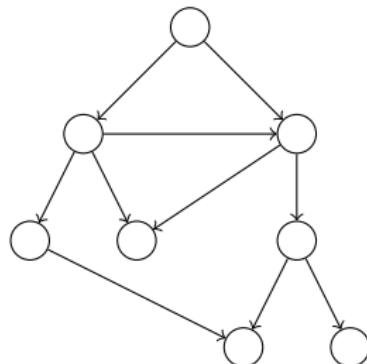
## Graphes

Intuition : n'importe quels éléments peuvent être reliés.

## DAG

(directed acyclic graphs)

Intuition : représentation des ordres partiels.



# Quelques types de structures non-linéaires

## Graphes

Intuition : n'importe quels éléments peuvent être reliés.

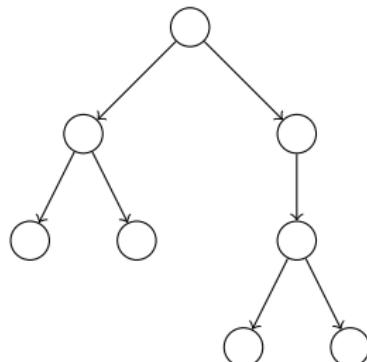
## DAG

(directed acyclic graphs)

Intuition : représentation des ordres partiels.

## Arbres

Intuition : représentation des liens de descendance.



# Quelques types de structures non-linéaires

## Graphes

Intuition : n'importe quels éléments peuvent être reliés.

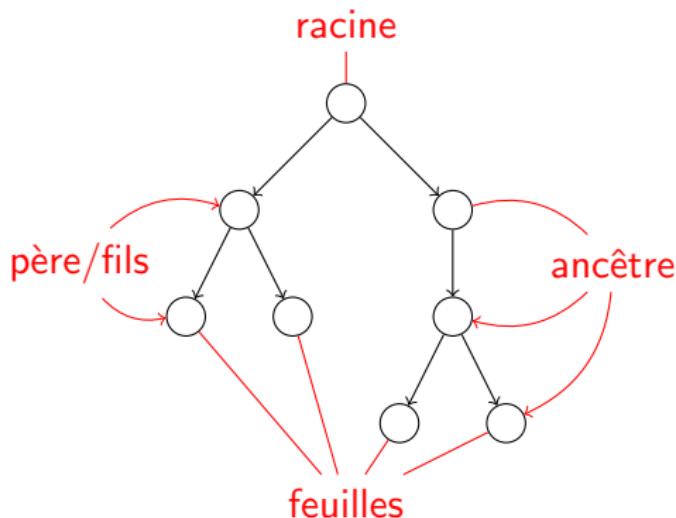
## DAG

(directed acyclic graphs)

Intuition : représentation des ordres partiels.

## Arbres

Intuition : représentation des liens de descendance.



# Arbres binaires

## Principe

Arbre où chaque élément (nœud) a exactement deux fils.

## Interface pour un arbre A

- ▶  $\text{root}(A)$  : valeur à la racine
- ▶  $\text{left}(A)$  : fils gauche de (la racine de) A
- ▶  $\text{right}(A)$  : fils droit de (la racine de) A
- ▶  $\text{make}(v, A, A')$  : arbre dont la racine contient la valeur v et a pour fils gauche A et pour fils droit A'
- ▶  $\text{nil}$  : arbre vide
- ▶  $\text{isEmpty}(A)$  : vrai si A est vide, faux sinon

# Mise en œuvre sur le principe des listes chaînées

## Élément d'arbre (nœud)

Contenu :

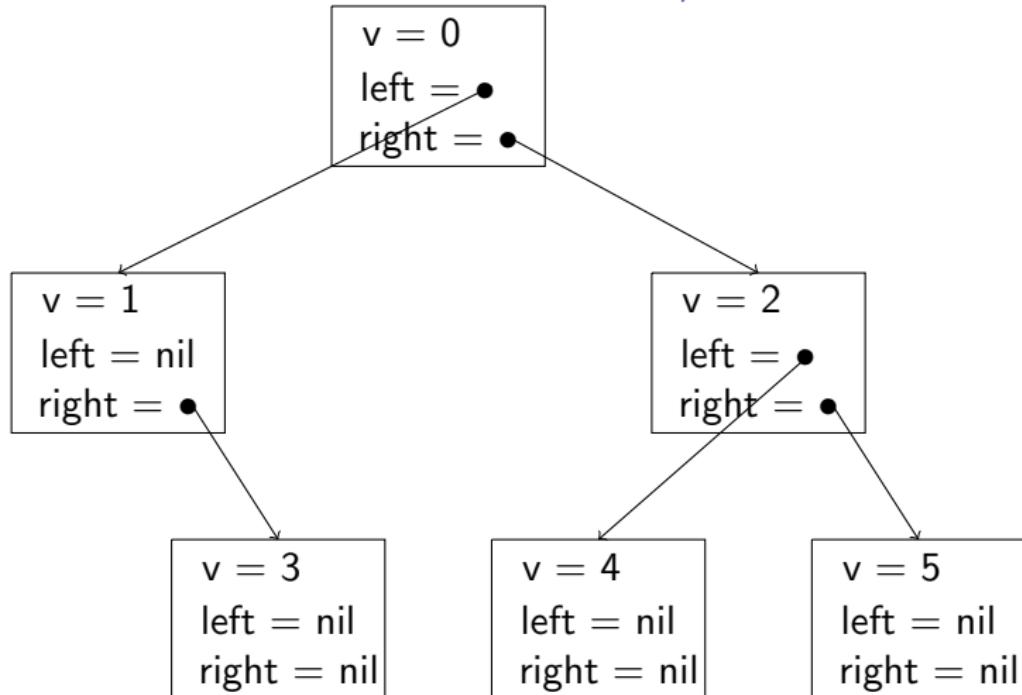
- ▶ une valeur val,
- ▶ (un pointeur vers) un arbre left, et
- ▶ (un pointeur vers) un arbre right.

Arbre représenté par sa racine.

## Lien avec l'interface

- ▶  $\text{root}(A) : A.\text{v}$
- ▶  $\text{left}(A) : A.\text{left}$
- ▶  $\text{right}(A) : A.\text{right}$
- ▶  $\text{make}(v, A, A') : \text{val} = v, \text{left} = A, \text{left}' = A'$
- ▶  $\text{nil} : \text{nil}$
- ▶  $\text{isEmpty}(A) : A == \text{nil}$

## Mise en œuvre des arbres binaires, suite



### Remarque

Comme les listes chaînées qui ne garantissent pas l'absence de cycles, cette représentation des arbres n'assure pas que ce sont vraiment des arbres.

# Parcours d'arbre

## Objectif

Afficher les valeurs de tous les nœuds d'un arbre.

## Récursivité

La structure de données est construite de manière récursive, le parcours sera donc naturellement écrit de manière récursive.

### Algorithme : parcours(A)

- ▶ si isEmpty(A), terminer
- ▶ sinon afficher(A.v), parcours(A.left), parcours(A.right)

Dans quel ordre sont affichées les valeurs des nœuds de A ?

# Parcours d'arbre

## Objectif

Afficher les valeurs de tous les nœuds d'un arbre.

## Récursivité

La structure de données est construite de manière récursive, le parcours sera donc naturellement écrit de manière récursive.

### Algorithme : parcours(A)

- ▶ si isEmpty(A), terminer
- ▶ sinon afficher(A.v), parcours(A.left), parcours(A.right)

Dans quel ordre sont affichées les valeurs des nœuds de A ?

### Autres versions

- ▶ parcours(A.left), afficher(A.v), parcours(A.right)
- ▶ parcours(A.left), parcours(A.right), afficher(A.v)

# Arbres n-aires

## Principe

Arbre où chaque élément (nœud) a autant de fils qu'on veut.

## Mises en œuvre

1. fils1, fils2, etc, sur le modèle de left et right
  - ▶ limité, il faut connaître le nombre de fils maximum possible dans un arbre
2. tableau de fils
3. liste de fils

# Exemple d'utilisation des arbres

## Rappel sur les tableaux triés (n éléments)

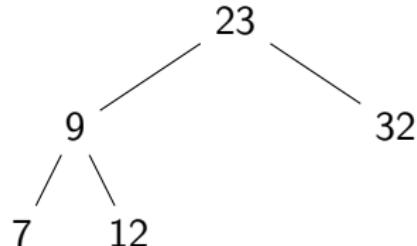
- ▶ trouver un élément : de l'ordre de  $\log(n)$  opérations
- ▶ insérer un élément : de l'ordre de  $n$  opérations
- ▶ supprimer un élément : de l'ordre de  $n$  opérations

Avec les arbres on peut mieux faire

- ▶ trouver un élément : de l'ordre de  $\log(n)$  opérations
- ▶ insérer un élément : de l'ordre de  $\log(n)$  opérations
- ▶ supprimer un élément : de l'ordre de  $\log(n)$  opérations

En utilisant des **arbres binaires de recherche**

# Arbre binaire de recherche



## Principe

Un nœud  $(\text{val}, \text{left}, \text{right})$  est tel que :

- ▶ tout  $v_\ell$  de left vérifie  $v_\ell \leq \text{val}$
- ▶ tout  $v_r$  de right vérifie  $v_r \geq \text{val}$

## Algorithme : recherche( $v, A$ )

si  $\text{isEmpty}(A)$  ou  $A.\text{val} == v$ ,  
alors

    retourner  $A$

sinon

    si  $v < A.\text{val}$  alors retourner  $\text{recherche}(v, A.\text{left})$   
    sinon retourner  $\text{recherche}(v, A.\text{right})$

# Insertion, nombre d'opérations

Algorithme : insérer( $v$ ,  $A$ )

si isEmpty( $A$ ), alors retourner make( $v$ , nil, nil)

si  $v < A.val$ , alors

    si isEmpty( $A.left$ ), alors  $A.left = \text{make}(v, \text{nil}, \text{nil})$

    sinon insérer( $v$ ,  $A.left$ )

sinon ( $v \geq A.val$ )

    si isEmpty( $A.right$ ), alors  $A.right = \text{make}(v, \text{nil}, \text{nil})$

    sinon insérer( $v$ ,  $A.right$ )

Nombre d'opérations dans le pire cas, pour un arbre à  $n$  nœuds

Hauteur de l'arbre (aussi bien pour la recherche que pour l'insertion), c'est-à-dire  $\log(n)$  opérations si l'arbre est équilibré.

# Ce qu'on peut faire de plus

## Suppression d'élément

Nombre d'opérations de l'ordre de la hauteur de l'arbre, basé sur la recherche du successeur du nœud à retirer (c'est-à-dire le nœud qui a la valeur suivante dans tout l'arbre).

## Arbres équilibrés

Nécessaire pour toujours améliorer les performances par rapport à un tableau trié.

- ▶ Insertion en gardant l'équilibre
- ▶ Suppression en gardant l'équilibre