Arbres et arbres binaires

Irena.Rusu@univ-nantes.fr
LINA, bureau 123, 02.51.12.58.16

Sommaire

- Arbres (Rappels)
  - Généralités
  - Parcours
  - Représentations en machine
- Arbres binaires
  - Définition, implémentation
  - Hauteur

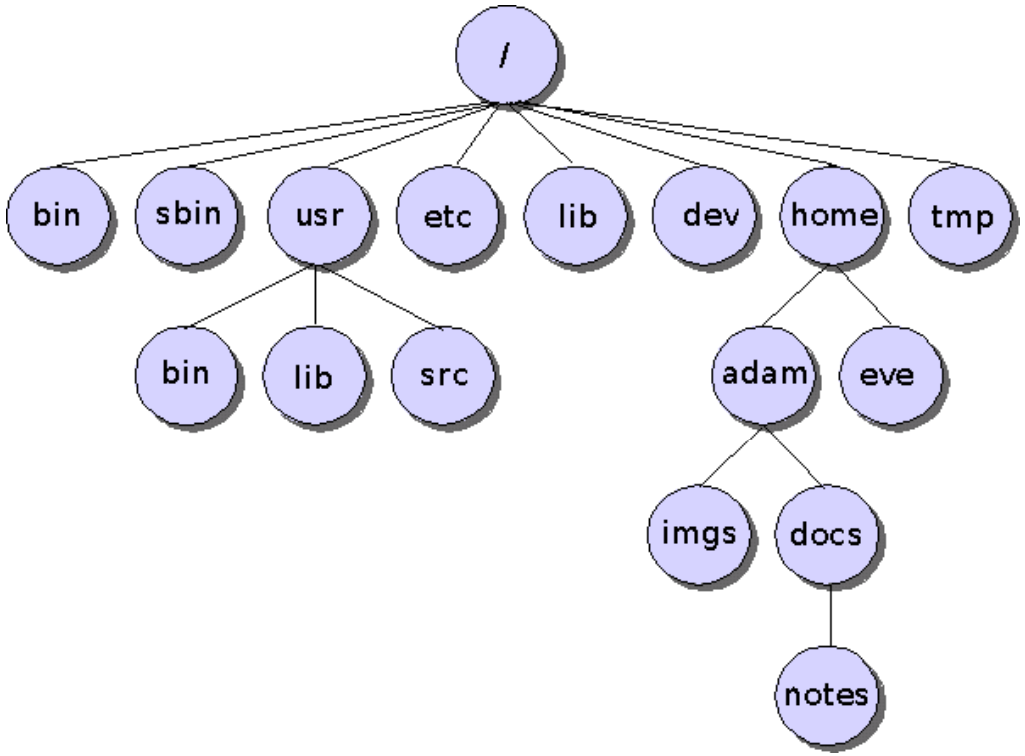
Sommaire

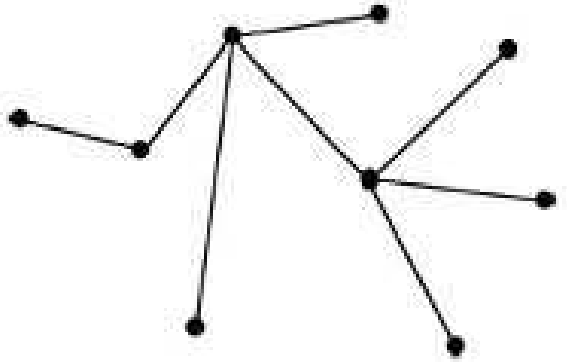
- Arbres (rappels)
  - Généralités
  - Parcours
  - Représentations en machine
- Arbres binaires
  - Définition, implémentation
  - Hauteur

Mise en garde

- Deux manières de définir les arbres :
  - Comme une structure hiérarchique, la racine (d'habitude) vers le haut
  - Comme un ensemble de relations entre objets, qui ne crée pas de cycle (mais qui n'est pas hiérarchique).
- Conséquences :
  - Dans le contexte « structures de données », on va utiliser la première
  - Dans le contexte « théorie des graphes », on va utiliser la seconde.

Exemples





Aucune hiérarchie

Hiérarchie sous-entendue (pas d'arcs)

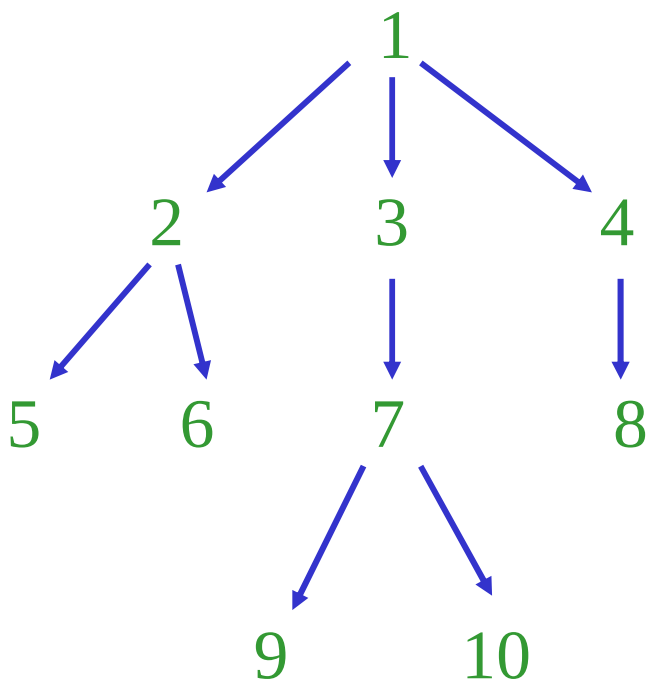
Sommaire

- Arbres (rappels)
  - Généralités
  - Parcours
  - Représentations en machine
- Arbres binaires
  - Définition, implémentation
  - Hauteur

Arbres

Arbre ordinaire : *A* défini par

- *N* ensemble des nœuds
- *P* relation binaire « parent de »
  (définit l'ensemble d'arcs Arc(A))
- $r \in N$ la racine



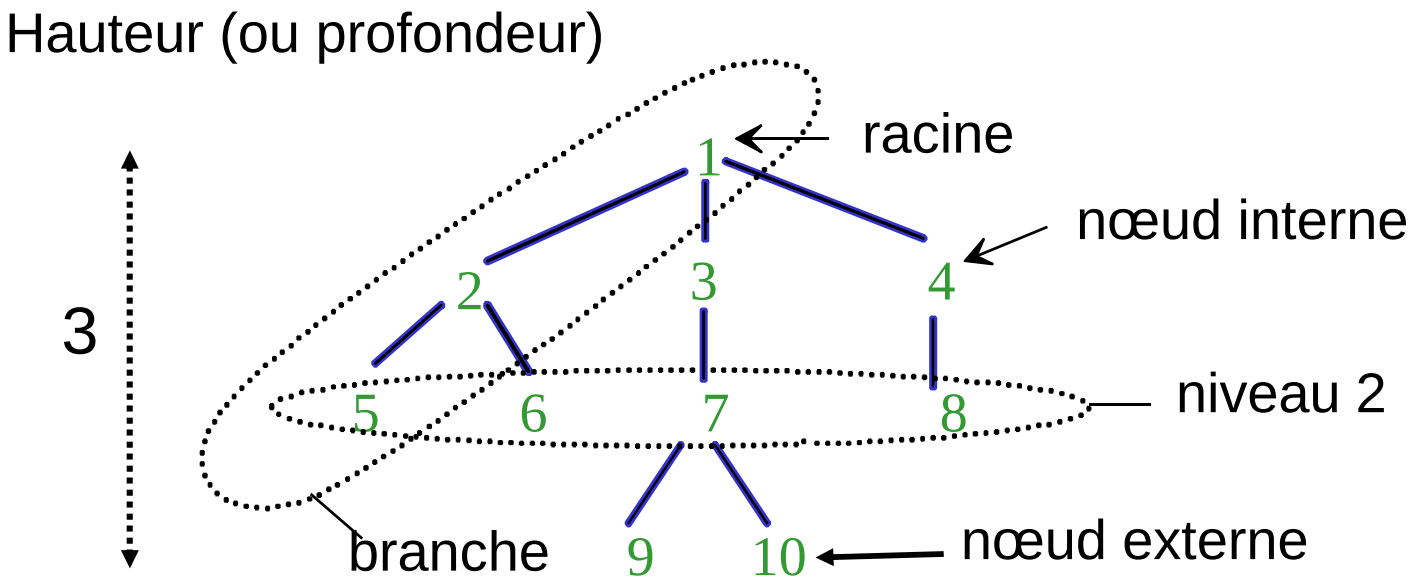
$\forall x \in N \ \exists$ un seul chemin de r vers x

$r = y_0 \ P \ y_1 \ P \ y_2 \ \ldots \ P \ y_n = x$

$\Rightarrow r$ n'a pas de parent

$\Rightarrow \forall x \in N - \{ r \}$ x a exactement un parent

Terminologie



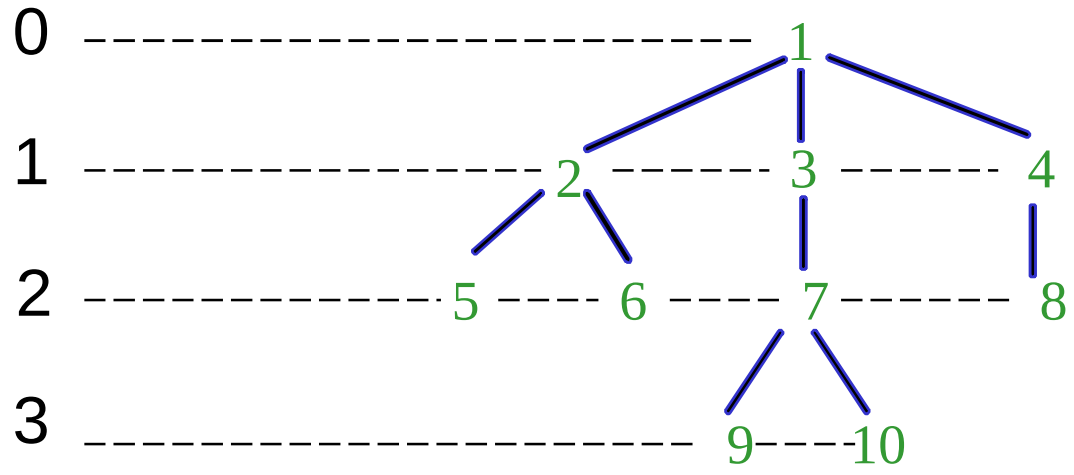
2, 3, 4	enfants de 1
3, 4	frères de 2
1, 3, 7	ancêtres de 7
7, 9, 10	descendants de 7

Niveaux

A arbre *x* nœud de *A*

$niveau_A(x)$ = distance de *x* à la racine

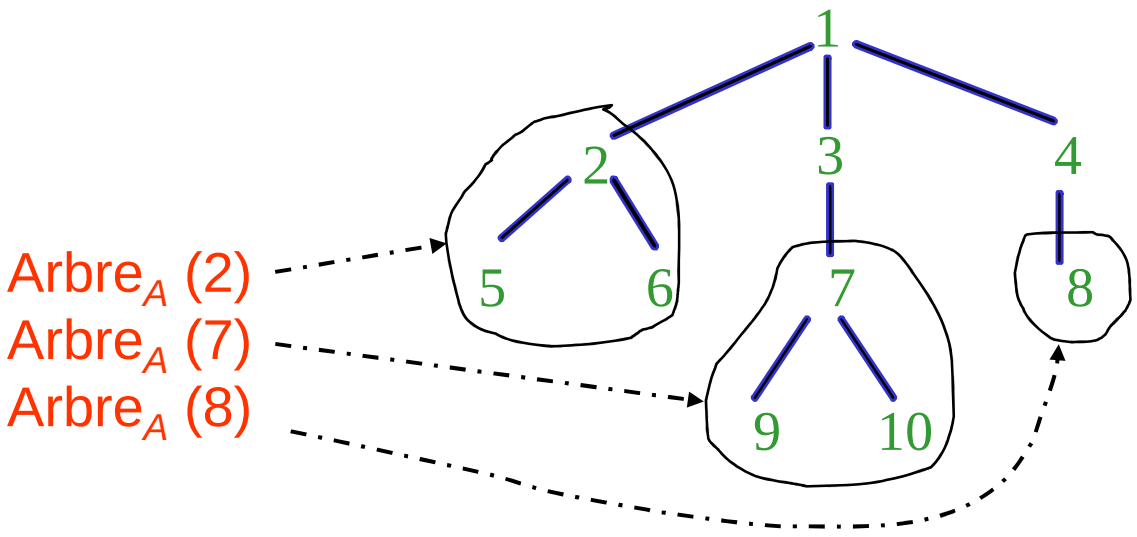
$$niveau_A(x) = \begin{cases} 0 & \text{si } x = racine(A) \\ 1 + niveau\ (parent\ (x)\ ) & \text{sinon} \end{cases}$$



Sous-arbres

A arbre *x* nœud de *A*

Arbre$_A$(*x*) = sous-arbre de *A* qui a racine *x*



Sommaire

- Arbres (rappels)
  - Généralités
  - Parcours
  - Représentations en machine
- Arbres binaires
  - Définition, implémentation
  - Hauteur

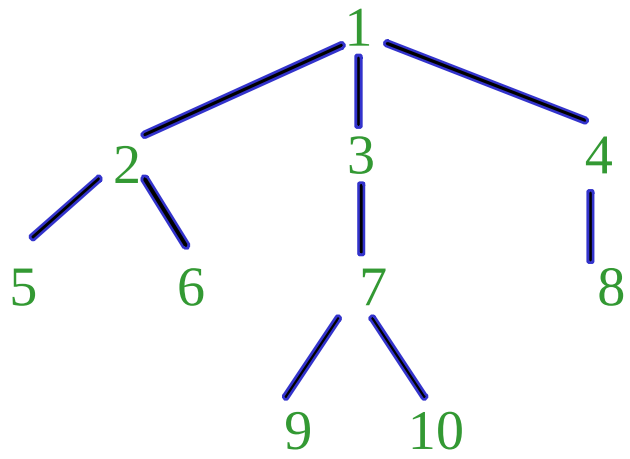
Parcours

Utiles pour l ’exploration des arbres

Deux types :

- parcours en profondeur
  - préfixe, suffixe, symétrique
  - parcours branche après branche
- parcours en largeur ou hiérarchique
  - parcours niveau après niveau

Parcours en profondeur



Arbre non vide $A = (r, A_1, A_2, \ldots, A_k)$

Parcours préfixe

$P(A) = (r).P(A_1). \ldots .P(A_k)$ (1, 2, 5, 6, 3, 7, 9, 10, 4, 8)

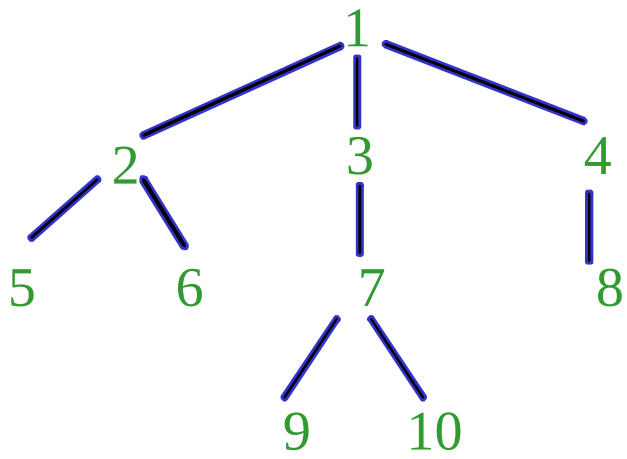
Parcours suffixe

$S(A) = S(A_1). \ldots .S(A_k).(r)$ (5, 6, 2, 9, 10, 7, 3, 8, 4, 1)

Parcours symétrique (ou interne)

$I(A) = I(A_1).(r).I(A_2). \ldots .I(A_k)$ (5, 2, 6, 1, 9, 7, 10, 3, 8, 4)

Parcours en largeur



Arbre non vide $A = (r, A_1, A_2, \ldots, A_k)$

Parcours hiérarchique

$$H(A) = (r, x_1, \ldots, x_i, x_{i+1}, \ldots x_j, x_{j+1}, \ldots, x_n)$$

nœuds de niveau 0, 1, 2, ...

(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

Sommaire

- Arbres (rappels)
  - Généralités
  - Parcours
  - Représentations en machine
- Arbres binaires
  - Définition, implémentation
  - Hauteur

Représentations en machine (1)

Représentation de la relation *P*

table des parents

Avantages

représentation simple

parcours faciles vers la racine

économique en mémoire

Inconvénients

accès difficiles aux nœuds depuis la racine

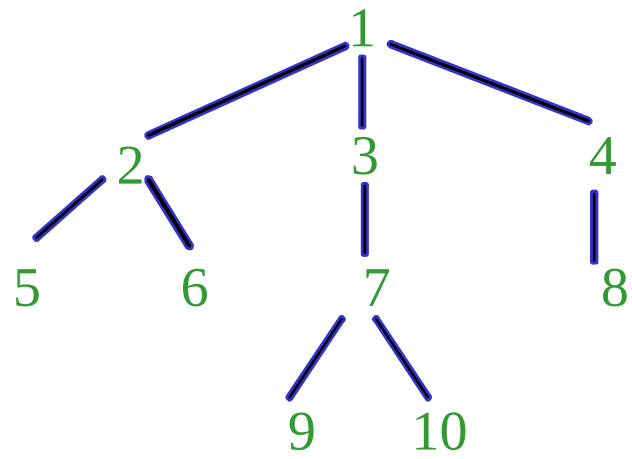
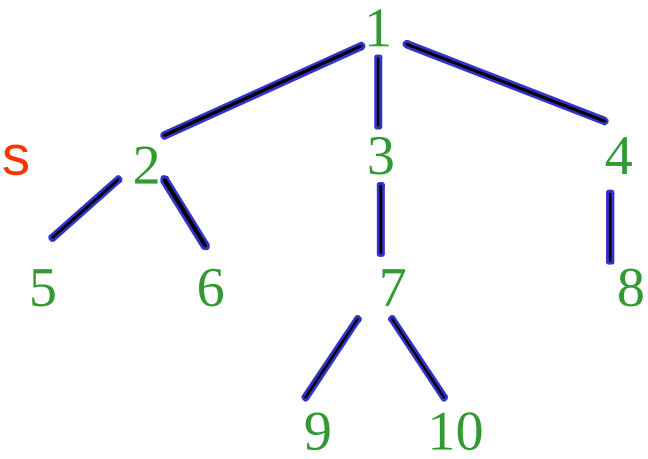


table des parents *P*

1	2	3	4	5	6	7	8	9	10
-	1	1	1	2	2	3	4	7	7

Représentations en machine (2)

Représentation des listes de sous-arbres par chaînage

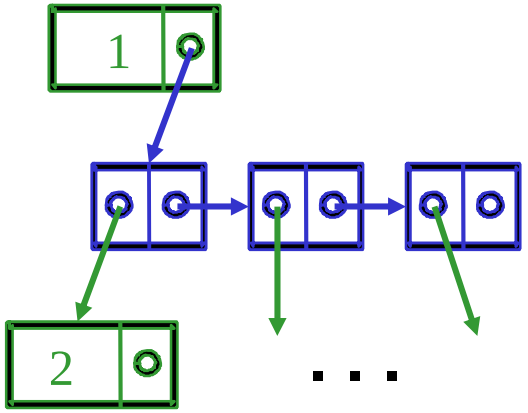


Avantages

- accès faciles depuis la racine
- correspond à la définition récursive

Inconvénients

- parcours difficiles vers la racine
- relativement gourmand en mémoire



Note. Si nombre de fils constant, tableau à la place de liste.

Sommaire

- Arbres (rappels)
  - Généralités
  - Parcours
  - Représentations en machine
- Arbres binaires
  - Définition, implémentation
  - Hauteur

Arbres binaires

Arbre binaire : tout nœud possède deux sous-arbres (vides ou non)

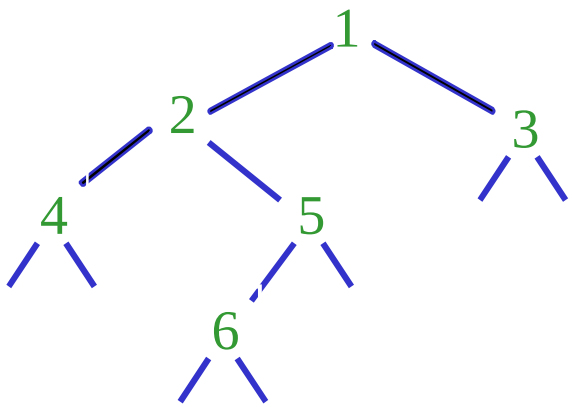
$$A = \begin{cases} \Lambda & \text{arbre vide ou} \\ (r, G, D) & \end{cases}$$

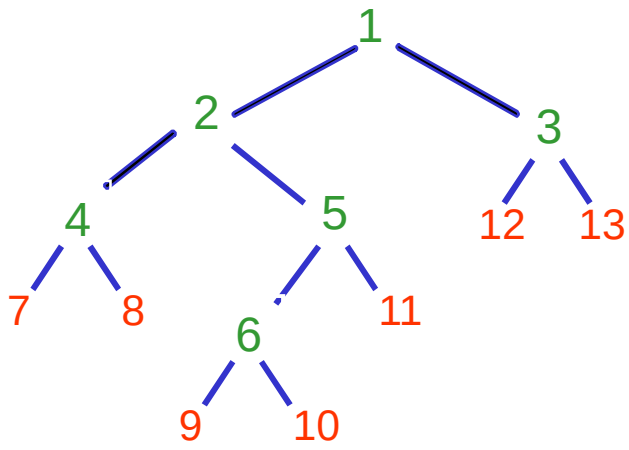
r élément, *G*, *D* arbres binaires

Nœuds (*A*) = {*r*} ∪ Nœuds (*G*) ∪ Nœuds (*D*)

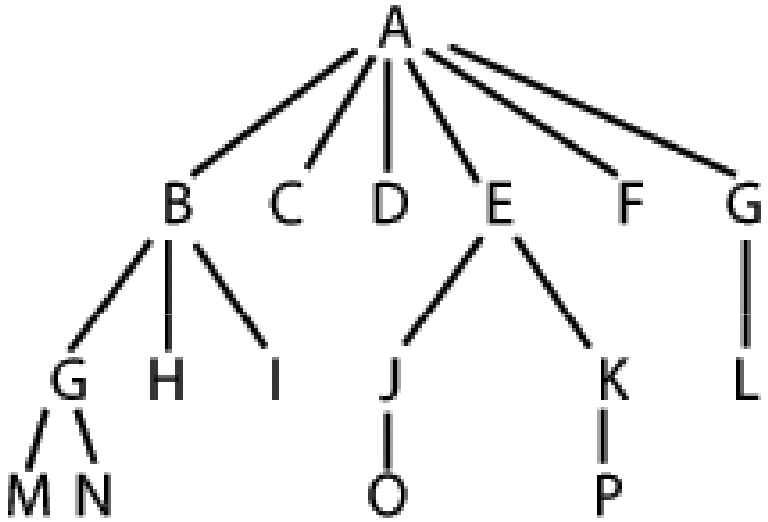
unions disjointes

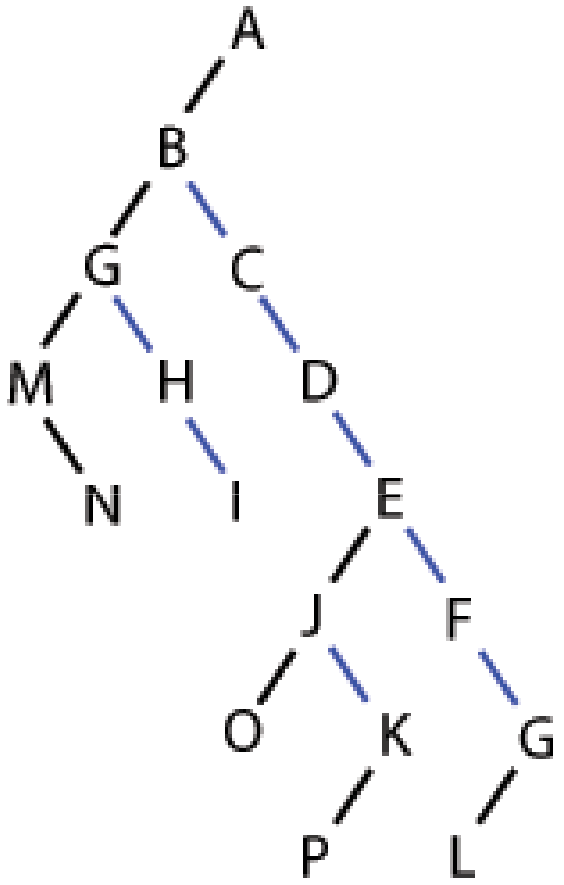
Arbre binaire complet : tout nœud interne possède deux enfants





Transformation d'un arbre n-aire en arbre binaire





- Arbre binaire = Vision « inclinée » de l'arbre de départ
- Parcours en profondeur préfixe identique
- La profondeur augmente, mais l'implémentation est plus aisée

Algorithme de transformation (1)

- Arbre n-aire T fourni : les fils d'un nœud peuvent être parcourus par une boucle, selon les représentations vues précédemment
- Arbre binaire U construit
- **Idée :**
  - La racine de U est une copie de la racine de T
  - Parcours en profondeur préfixe de l'arbre T, en récursif
  - Lorsqu'on est sur un nœud X de T, sa copie Y est déjà dans U, et
    - Le premier fils de X dans T devient fils gauche de Y dans U
    - Le deuxième fils de X dans T devient fils droit de Y dans U
    - Le troisième fils de X devient fils droit du fils droit de Y
    - Etc.

Algorithme de transformation (2)

copie(X) crée un nœud de U copie du nœud pointé par X dans T

```
Si T ≠ arbre vide, alors U ←copie(nœud racine de T)
 (void) transfoNen2(T: arbre N-aire, U: arbre binaire);
      X ← racine de T; Y← racine de U;                    // variables locales
     si (X a au moins un fils)  {
          Dernier← arbre vide;                            // variable locale
          pour tout fils W de X faire {
               si (Dernier = arbre vide) alors{ Y.G←copie(W); Dernier ← Y.G;}
               sinon {Dernier.D←copie(W); Dernier ← Dernier.D;}
               transfoNen2(W, Dernier);
          }
```

Opérations de base avec les arbres binaires

(que nous n'écrirons pas)

Arbre-vide : → arbre // sans argument, crée un arbre binaire vide

G, D : arbre → arbre // étant donné un arbre binaire, retourne les sous-arbres G et D de la racine

Cons : nœud x arbre x arbre → arbre // étant donné un nœud et deux arbres, crée l'arbre binaire ayant comme racine ce nœud, et comme sous-arbres de la racine le premier arbre à G et le deuxième arbre à D

Elt : arbre → élément // étant donné un arbre, retourne l'élément situé à // sa racine

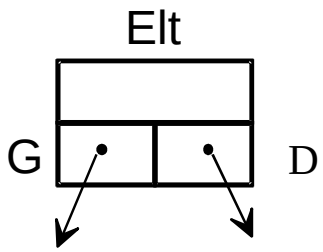
Vide : arbre → booléen // test arbre vide

Remarques : 1) un nœud est vu comme la racine de l'arbre situé « en dessous »

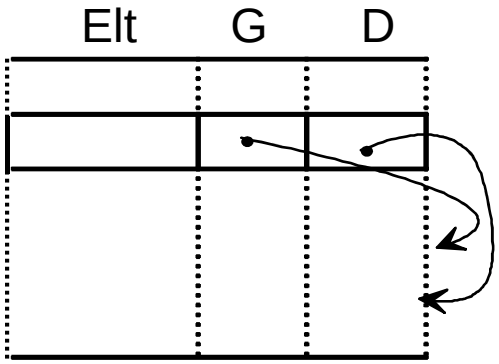
2) l'implémentation de l'arbre n'est pas visible à ce niveau.

Implémentations possibles

Type « pointeur »



Type « tableau »



Avantage :

Suppression d'un nœud en O(1) (càd temps constant), sans créer de « trous » dans la structure

Avantage :

Recherche possible d'un élément comme dans un tableau.

Mise en garde

- L'avantage du type « tableau » devient caduque lorsqu'on parle d'arbres binaires de recherche (surtout équilibrés)
- Par conséquent, tous nos arbres seront supposés implémentés sous la forme « pointeur »

→ Interdiction de parcourir les éléments d'un arbre comme si c'était un tableau

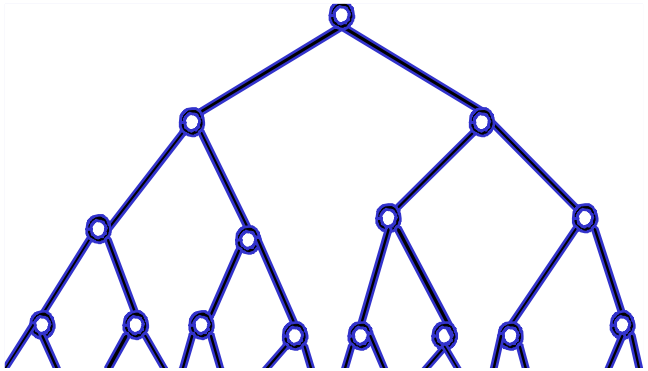
→ Besoin de traverser l'arbre avec un pointeur

Sommaire

- Arbres (rappels)
  - Généralités
  - Parcours
  - Représentations en machine
- Arbres binaires
  - Définition, implémentation
  - Hauteur

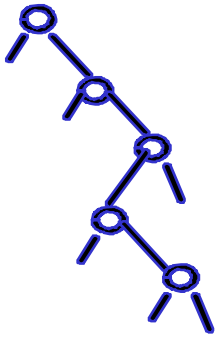
Mesures des arbres binaires

Arbre binaire, hauteur *h* et *n* nœuds



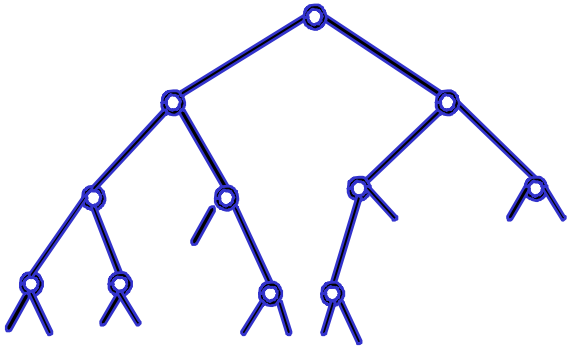
Arbre plein

2^i nœuds au niveau *i*
$n = 2^{h+1} - 1$



Arbre filiforme

1 nœud par niveau
$n = h + 1$



Arbre binaire

$h + 1 \leq n \leq 2^{h+1} - 1$
$\log_2 (n + 1) - 1 \leq h \leq n - 1$

Arbres et ensembles ordonnés

But : gérer des sous-ensembles de *E* (ordonné) avec les opérations

Ens_vide : → Ens // crée un ensemble vide

Ajouter : Ens x élément → Ens // ajoute un élément

Enlever : Ens x élément → Ens // enlève un élément

Elément : Ens x élément → booléen // teste l'appartenance

Vide : Ens → booléen // teste si ensemble vide

Min, Max : Ens → élément // calcule min, max

Place : Ens x élément → arbre // trouve la place de l'élément
// dans l'arbre

Implémentations possibles

Tables (triées ou non), Listes chaînées,

Tables de hachage, Arbres (équilibrés ou non)

A suivre … avec des arbres

- Structures Classe-Union
- ABRs (ou arbres binaires de recherche)
- AVLs (ou arbres binaires de recherche équilibrés en hauteur/profondeur)
- Arbres rouges et noirs (ou arbres binaires de recherche bicolores)