Récursion et arbres

Algorithmes récursifs

Exemple : calculer le factoriel $n!$

On peut donner une solution itérative ou récursive

C'est vrai en général :

Chaque algorithme récursif peut être transformé en un algorithme non-récursif itératif et chaque boucle peut être transformé en récurrences sans boucles

Algorithmes récursifs 2

Pour que cela soit correct, une solution récursive doit

1. définir des cas de base
2. performer un calcul fini

ce dernier est plus simple à démontrer si l'arguments diminuent à chaque appel récursif

exemple simple : factoriel

autre exemple (Ackerman)

$$A(i,j) = \begin{cases} 2^j & \text{si } i = 1; \\ A(i-1,2) & \text{si } j = 1 \text{ et } i \geq 2 \\ A(i-1,A(i,j-1)) & \text{si } i,j \geq 2 \end{cases}$$

soit i soit j diminue à chaque appel récursif

Algorithme d'Euclid

```
E1  Algo gcd(a, b)
E2  tandis que (b ≠ 0)
E3      c ← a mod b
E4      a ← b
E5      b ← c
E6  retourner a
```

(on impose $b \leq a$)

```
rE1  Algo gcd(a, b) // récursion
rE2  si b = 0 alors retourner a
rE3  sinon retourner gcd(b, a mod b)
```

Algorithme d'Euclid 2

```
E1  Algo gcd(a, b) // avec b ≤ a
E2  tandis que (b ≠ 0)
E3      c ← a mod b
E4      a ← b
E5      b ← c
E6  retourner a
```

Temps de calcul : polynomial

Preuve : On a $a = kb + c \geq (k+1)c \geq 2c$ en Ligne E3. Donc $bc \leq ab/2$. Par conséquence, le nombre d'itérations est borné par $\lg(ab) = \lg a + \lg b$. Si $a \bmod b$ prend $O(\log^2 a)$ temps, alors l'exécution est en temps $O(\log^3 a)$.

Parcours de listes chaînées

compter le nombre des éléments

parcours de la liste

Diviser pour régner

dM1 **Algo** MAX(L, l, r) // trouve le max parmi $L[l], \ldots, L[r]$
dM2 **si** $l = r$ **alors** retourner $L[l]$
dM3 $m \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor$
dM4 $u \leftarrow$ MAX(L, l, m) ; $v \leftarrow$ MAX$(L, m+1, r)$
dM5 **si** $u > v$ **alors** retourner u **sinon** retourner v

Temps de calcul pour ce genre de solutions

$$T(n) = T(k) + T(n-k) + O(1)$$

Alors, $T(n) = O(n)$.

Autres récurrences typiques

Recherche binaire : $T(n) = T(n/2) + O(1)$
solution $T(n) = O(\log n)$

Tri par fusion : $T(n) = 2T(n/2) + O(n)$
solution $T(n) = O(n \log n)$

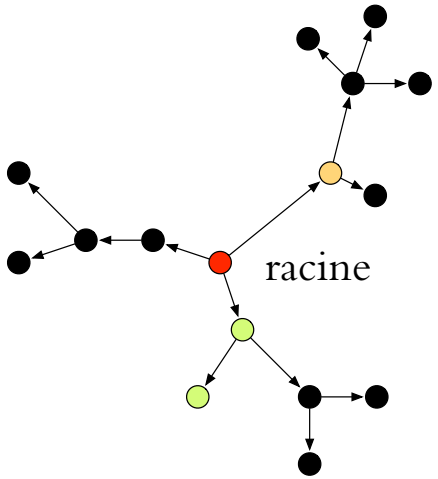
Arbres — terminologie

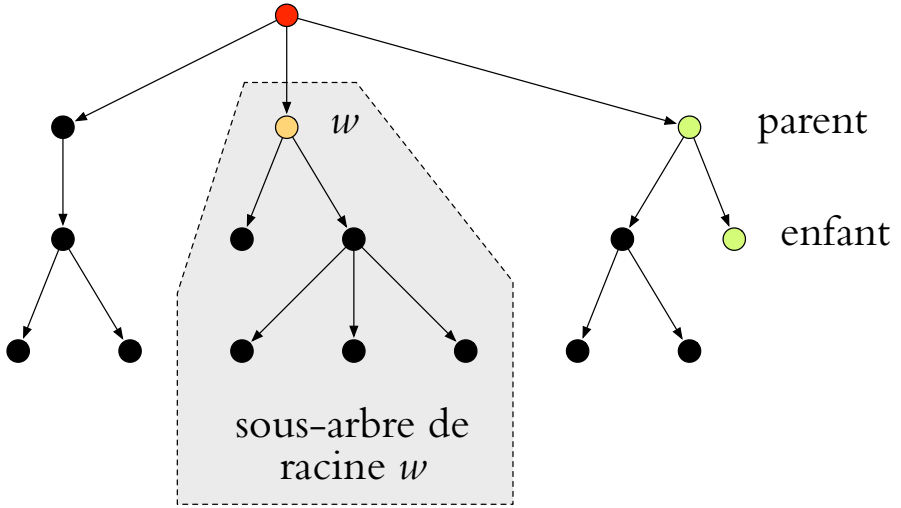
Graphe : sommet et arêtes

Arbre (libre) : graphe non-orienté, connexe et acyclique

Arbre raciné ou **arborescence** (en théorie des graphes) : graphe orienté, et connexe avec un sommet spécial, la **racine**, dans lequel il y a 1 chemin simple de la racine à chaque sommet.

- orientation des arêtes : relations parent-enfant
- u est dans le sous-arbre enraciné à w ssi w est sur le chemin de la racine à u





Arbres — utilité

Les arbres jouent un rôle central dans la conception et analyse d'algorithmes

- arbres pour décrire les propriétés dynamiques des algorithmes

- structures de données explicites qui sont des réalisations concrètes d'arbres

Arbres — terminologie 2

Ancêtre : w est l'ancêtre de u ssi u est dans le sous-arbre enraciné à w

Descendant : u est un descendant de w ssi u est dans le sous-arbre enraciné à w

Dans des graphes orientés (y inclut les arborescences) les sommets s'appellent aussi des nœuds, et les arêtes s'appellent aussi des arcs

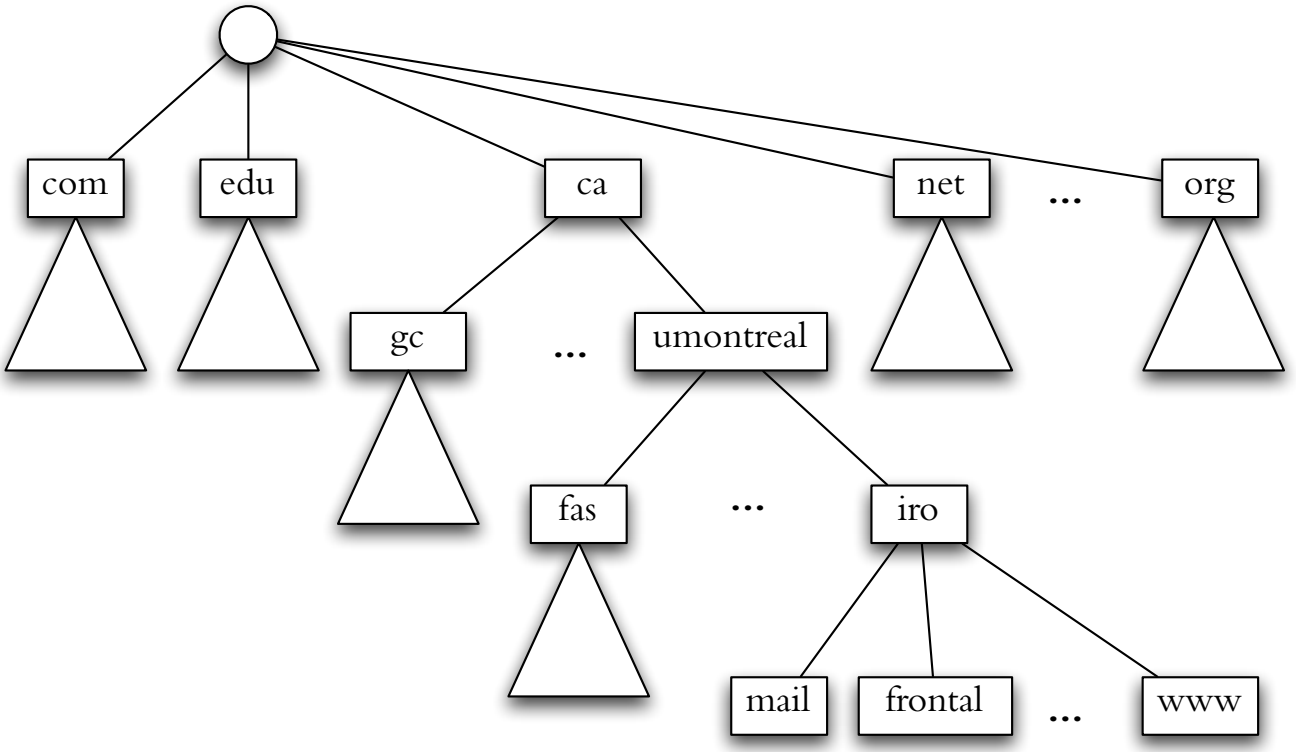
Degré d'un nœud : nombre d'arcs sortants (ou nombre d'enfants)

Nœud externe ou feuille : aucun arc sortant (aucun enfant)

Nœud interne : tous les autres (au moins 1 enfant)

Arborescence — exemples

Domaines internet :



Packages en Java, répertoires sous Unix, . . .

Arbre ordonné

Arbre ordonné : il existe un ordre parmi les enfants de chaque nœud interne

Arbre numeroté : les enfants d'un nœud sont étiquetés par des entiers positifs distincts.

i-ème enfant absent : si aucun enfant n'est étiqueté par i

Arité k : ssi tous les enfants avec étiquettes $> k$ sont absents.

Arbre binaire : arbre numeroté d'arité 2

enfant gauche ou droit : enfant étiqueté par 1 ou 2

frères ou sœurs : nœuds avec le même parent

Arbre numeroté (cont.)

Définition altérnative par récurrence :

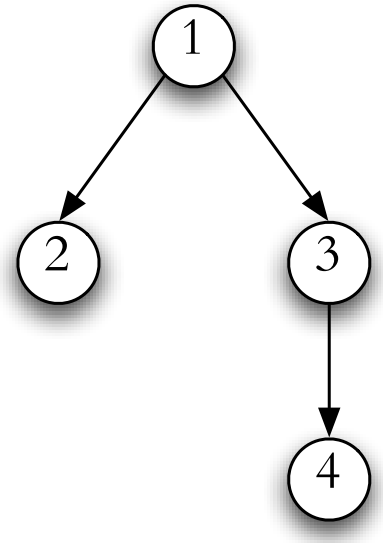
Déf. *Un arbre k-aire T est une structure définie sur un ensemble fini de nœuds qui*

1. *ne contient aucun nœud,* ***ou***
2. *est composé de $(k+1)$ ensembles de nœuds disjoints : un nœud* ***racine*** *r, et les arbres k-aires $T_1, T_2, \ldots, T_k$.*

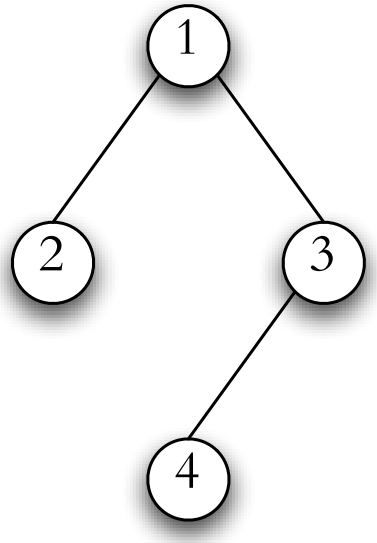
(En 2, la racine de T_i quand T_i est non-vide est l'enfant de r étiqueté par i.)

Arbre numeroté (cont.)

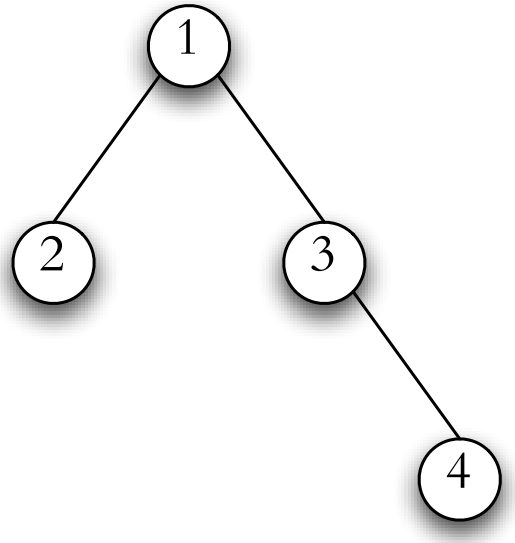
Attention : l'ordre des enfants est important dans un arbre numeroté



arborescence
(l'ordre des enfants
n'est pas important)



arbre binaire
(«4» est l'enfant gauche)



arbre binaire
(«4» est l'enfant droit)

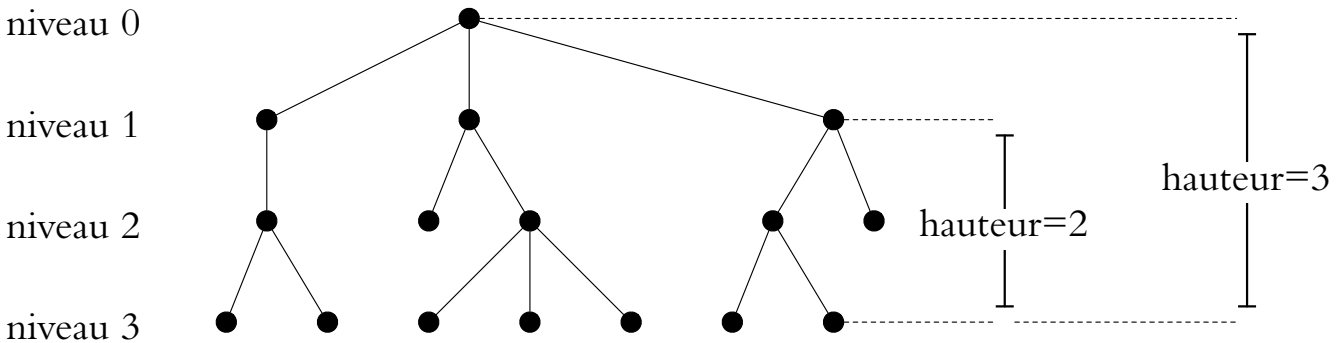
Dans ce cours, les nœuds internes dans les arbres binaires auront toujours 2 enfants

Hauteur et niveau

Niveau d'un nœud u : longueur du chemin qui mène à u à partir de la racine

Hauteur d'un nœud u : longueur du chemin à la feuille la plus distante dans le sous-arbre de u

Hauteur de l'arbre : hauteur de la racine



Longueur du chemin (interne/externe) : somme des niveaux de tous les nœuds (internes/externes)

Hauteur d'un arbre binaire

Thm. La hauteur d'un arbre binaire à n feuilles est entre $\lceil \lg n \rceil$ et $(n - 1)$.

Implantation d'un arbre numeroté

Arbre = ensemble d'objets représentant de nœuds + relations parent-enfant

En général, on veut retrouver facilement le parent et les enfants de n'importe quel nœud

Approche Java :

```
public class TreeNode {
   TreeNode parent;
   TreeNode enfant_gauche;
   TreeNode enfant_droit;
   ...
}
```

Si arbre k-aire, alors on peut avoir `TreeNode[] enfants`
avec `enfants.length` $= k$.

Parcours des arbres

Dans un parcours, tous les nœuds de l'arbre sont visités.

Déf. Dans un parcours préfixe (*preorder traversal*), chaque nœud est visité avant que ses enfants soient visités.

Déf. Dans un parcours postfixe (*postorder traversal*), chaque nœud est visité après que ses enfants sont visités.

Parcours préfixe et postfixe

```
   Algo PARCOURS-PRÉFIXE(x)
1  if x ≠ null then
2      visiter x
3      for i ← 1, ..., k do PARCOURS-PRÉFIXE(enfant(x, i))
```

```
   Algo PARCOURS-POSTFIXE(x)
1  if x ≠ null then
2      for i ← 1, ..., k do PARCOURS-POSTFIXE(enfant(x, i))
3      visiter x
```

(enfant(x, i) donne l'enfant de x étiqueté par i — s'il n'y en a pas, alors null)

Maintenant PARCOURS-PRÉFIXE(racine) va visiter tous les nœuds dans l'arbre dans l'ordre préfixe.

Parcours infixe

On peut parcourir un arbre binaire aussi dans l'ordre infixe

Déf. Dans un parcours infixe (*inorder traversal*), chaque nœud est visité après son enfant gauche mais avant son enfant droit.

```
   Algo PARCOURS-INFIXE(x)
1  if x ≠ null then
2      PARCOURS-INFIXE(gauche(x))
3      visiter x
4      PARCOURS-INFIXE(droit(x))
```

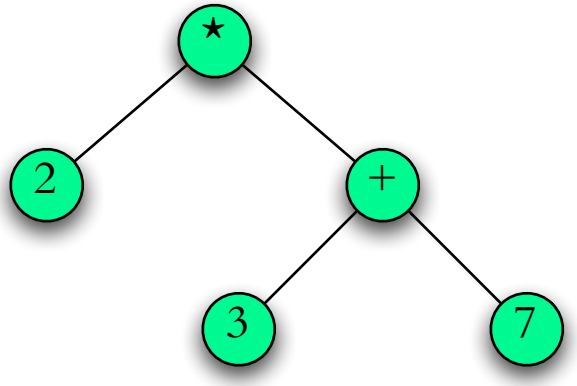
Récursions sur arbres binaires

La structure récursive de l'arbre permet des solutions naturelles par récurrences

Logique générale : traiter la racine, sous-arbre gauche, et le sous-arbre droit + calcul avec les valeurs

exemples : taille et hauteur

Arbre syntaxique



notation infixe: 2⋆(3+7)
notation préfixe: ⋆ 2 + 3 7
notation postfixe: 2 3 7 + ⋆