Algorithmique fonctionnelle - Généralités sur les arbres Arbres binaires

Introduction

Les arbres représentent une structure de données omniprésente en informatique.

Exemples

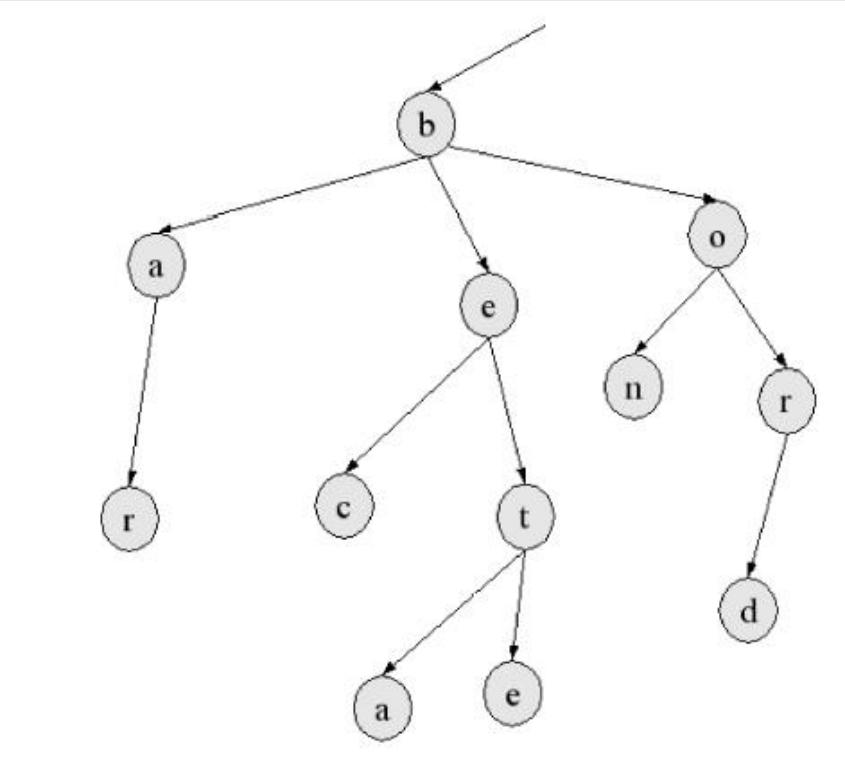
- la structure de stockage des fichiers sur un disque dur
- les résultats d'un tournoi
- une expression formelle, comme $x - (2 * y + 3)$

Intérêt des structures arborescentes

Exemple : recherche d'un mot dans un dictionnaire

- Structure linéaire ("tableau") triée
  - recherche dichotomique : complexité logarithmique par rapport à la taille du dictionnaire
  - insertion d'un mot : complexité linéaire
- Structure arborescente : recherche + insertion
  - indépendant de la taille du dictionnaire !
  - ne dépend que de la longueur du mot !

Exemple : recherche d'un mot dans un dictionnaire



Définitions

Soit S un ensemble fini et soit $s \longrightarrow s'$ une relation binaire sur S. Si $s \longrightarrow s'$, on dit que s est un **père** de s' ou que s' est un **fils** de s.

Définition : Arbre

On dit que $A = (S, \longrightarrow)$ est un **arbre** si S est l'ensemble vide ou si $S \neq \varnothing$ et les conditions suivantes sont satisfaites :

1. S contient un seul élément s_0 qui n'a pas de père. L'élément s_0 s'appelle la **racine** de l'arbre ;
2. pour tout $s \in S - \{s_0\}$, s a un seul père ;
3. si $s_1 \longrightarrow s_2 \longrightarrow \ldots \longrightarrow s_{n-1} \longrightarrow s_n$ $(n \geq 2)$ est une descendance, tous les éléments s_i, avec $1 \leq i \leq n$, sont distincts.

Les éléments de S sont appelés des **sommets**, ou des **nœuds**.

Définitions

Définition : Descendance

Soit $A = (S, \rightarrow)$ un arbre non vide et soit $s \in S$.

- On note $F(s)$ l'ensemble des fils de s.
- On note $s \xrightarrow{*} s'$ et on dit que s' est un **descendant** de s si $\exists n \in N$ tel que : $s_1 \rightarrow s_2 \rightarrow ... \rightarrow s_{n-1} \rightarrow s_n$ est une descendance avec $s_1 = s$, $s_n = s'$.
- On note $S(s)$ l'ensemble des descendants de s.

Définitions

Lemme

Soit $A = (S, \rightarrow)$ un arbre et soit $s \in S$. Alors $(S(s), \rightarrow)$ est un arbre de racine s qui est noté A_s, et qui est appelé la coupe de A en s.

- On dit qu'un sommet s est une **feuille** si $S(s) = \{s\}$. Dans le cas contraire, on dit que s est un **nœud interne**.
- On appelle **branche** issue de s toute suite $s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_{n-1} \rightarrow s_n$ où $s_1 = s$ et s_n est une feuille.
- On note S_I l'ensemble des nœuds internes de A et S_F l'ensemble des feuilles de A.

Définitions

Lemme

Soit $A = (S, \rightarrow)$ un arbre non vide :

- l'ensemble des feuilles S_F est non-vide ;
- tout sommet s a au moins une feuille comme descendant ;
- pour tout sommet s, il existe une branche et une seule qui le relie à la racine.

Définitions

Définition : Hauteur d'un arbre

Soit $A = (S, \rightarrow)$ un arbre.

- Si A est vide, sa **hauteur** est par définition égale à -1.
- Si A est non vide, sa **hauteur** est la longueur de la plus grande branche issue de la racine.

Définitions

Définition : Hauteur d'un nœud

Soit $A = (S, \rightarrow)$ un arbre non vide et soit $s \in S$. On appelle **hauteur** de s la longueur de l'unique branche qui le relie à la racine. On note hauteur(s) ce nombre.

Lemme

Soit $A = (S, \rightarrow)$ un arbre. La fonction hauteur est définie par :

$$hauteur(A) = \begin{cases} -1 & \text{si } A \text{ est vide} \\ 0 & \text{si } A \text{ est réduit à une feuille} \\ 1 + \max_{s \in F(s_0)}(\text{hauteur}(A_s)) & \text{si } A \text{ a pour racine } s_0 \text{ et n'est pas réduit à une feuille} \end{cases}$$

Définitions

Définition

Soit $A = (S, \rightarrow)$ un arbre non vide et soit $s \in S$. On appelle :

- **degré de branchement** de s le nombre : $deg(s) = |F(s)|$
- **degré de branchement** de l'arbre A le nombre : $deg(A) = max_{s \in S}(|F(s)|)$

Lemme

Pour tout arbre $A = (S, \rightarrow)$ ayant un degré de branchement inférieur ou égal à d et qui a $n > 0$ éléments, on a les inégalités :

$$\lfloor \log_d n \rfloor \leq hauteur(A) \leq n - 1$$

Définitions

Définition

Soit $A = (S, \rightarrow)$ un arbre non vide. On dit que :

- A est **localement complet** si, pour tout nœud interne s de A, on a $deg(s) = deg(A)$;
- A est **complet** s'il est localement complet et si, pour toute feuille f de A, on a $hauteur(f) = hauteur(A)$;
- A est **parfait** si l'arbre A' obtenu en supprimant les feuilles du dernier niveau de A est complet et si les feuilles de l'arbre A sont regroupées le plus à gauche possible.

Définitions

Définition

Soit $A = (S, \rightarrow)$ un arbre, on appelle :

- **longueur de cheminement** de A, le nombre :
$$LC(A) = \sum_{s \in S} \text{hauteur}(s)$$
- **longueur de cheminement interne** de A, le nombre :
$$LCI(A) = \sum_{s \in S_I} \text{hauteur}(s)$$
- **longueur de cheminement externe** de A, le nombre :
$$LCE(A) = \sum_{s \in S_F} \text{hauteur}(s)$$

Introduction

Définition

Un **arbre binaire** est soit vide, soit constitué d'une **racine** contenant un élément et de deux arbres binaires disjoints, appelés respectivement **fils gauche** et **fils droit**.

L'intérêt des arbres binaires :

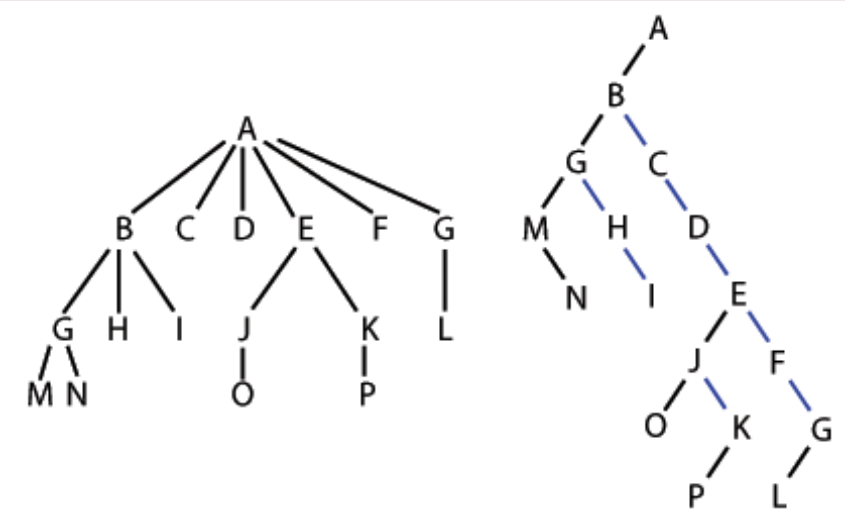
- simplifier la spécification et l'implémentation des opérations générales sur les arbres

Introduction

Notez

Tout arbre possède un arbre binaire équivalent.

Exemple



Définitions

Un arbre binaire est dit :

1. **dégénéré** (ou filiforme) s'il ne contient qu'une feuille ;
2. **complet** si
   - tous ses noeuds internes ont deux fils
   - et si toutes ses feuilles sont au même niveau.
3. **parfait** si tous ses niveaux sont remplis, sauf éventuellement le dernier (et dans ce cas les feuilles du dernier niveau sont regroupées le plus possible sur la gauche de l'arbre).

Définitions

1. La **taille** d'un arbre est le nombre de ses nœuds. Un arbre vide a donc une taille de 0.
2. La **hauteur** d'un nœud est le nombre de liens du chemin menant de la racine ce nœud. Par convention, un arbre vide a une hauteur de -1.
3. La **longueur de cheminement** (resp. **longueur de cheminement interne, externe**) est la somme des hauteurs des nœuds (resp. des nœuds internes, des feuilles) de l'arbre.

Types d'un arbre binaire

Comme une liste, un arbre binaire ne peut contenir que des éléments d'un certain type dans ses nœuds :

- Arbre binaire d'entiers
- Arbre binaire de chaînes
- Arbre binaire de booléens

Si un algorithme donné est applicable à tous les types d'arbre binaire, pour simplifier, pas besoin de mentionner le type des nœuds.

Arbre binaire

Constante

arbreVide

Fonctions de base

1. *estVide* : *ArbreBinaire* $\longrightarrow$ *Booleen*
   tester si l'arbre binaire est vide
2. *cons* : $T \times$ *ArbreBinaire* $\times$ *ArbreBinaire* $\longrightarrow$ *ArbreBinaire*
   construire un arbre par l'ajout de la racine à deux arbres existants
3. *racine* : *ArbreBinaire* $\longrightarrow T$
   accéder à l'élément racine de l'arbre
4. *filsGauche* : *ArbreBinaire* $\longrightarrow$ *ArbreBinaire*
   accéder au fils gauche
5. *filsDroit* : *ArbreBinaire* $\longrightarrow$ *ArbreBinaire*
   accéder au fils droit

Arbre binaire

Exemple 1 : Calculer la taille (le nombre d'éléments) d'un arbre

```
Fonction taille(A : ArbreBinaire) : Entier
Début
    Si estVide(A) Alors
        retourner 0
    Sinon
        retourner 1 + taille(filsGauche(A)) + taille(filsDroit(A))
    FinSi
Fin
```

Arbre binaire

Exemple 2 : Calculer la hauteur d'un arbre : **à vous !**

Arbre binaire

Exemple 2 : Calculer la hauteur d'un arbre : **à vous !**

```
Fonction hauteur(A : ArbreBinaire) : Entier
Début
    Si estVide(A) Alors
        retourner -1
    Sinon
        retourner 1 + max(hauteur(filsGauche(A)),hauteur(filsDroit(A)))
    FinSi
Fin
```

Les parcours d'arbres

Il existe différentes façons de parcourir les noeuds d'un arbre :

1. **Parcours en largeur** :
   - Visiter les frères avant les fils
   - Algorithme itératif en utilisant une file (2ème semestre)
2. **Parcours en profondeur** :
   - Visiter les fils avant les frères
   - *Algorithme récursif*
   - Algorithme itératif en utilisant une pile (2ème semestre)

Les parcours d'arbres

Parcours en profondeur **à main gauche**

```
Fonction parcoursProfondeur(A : ArbreBinaire)
Début
    Si estVide(A) Alors
        terminer
    Sinon
        traitement1
        parcoursProfondeur(filsGauche(A))
        traitement2
        parcoursProfondeur(filsDroit(A))
        traitement3
    FinSi
Fin
```

Parcours en profondeur

Il existe trois cas particuliers :

1. le parcours **préfixe** : seul traitement1 est non-vide ;
   - *nœud courant est traité avant les fils*
2. le parcours **infixe** : seul traitement2 est non-vide ;
   - *nœud courant est traité entre le fils gauche et le fils droit*
3. le parcours **postfixe** : seul traitement3 est non-vide.
   - *nœud courant est traité après les fils*

Les arbres binaires en Scheme

- En Scheme, un arbre n'est pas un type de base. Nous pouvons le créer via une liste :

  (racine filsGauche filsDroit)

- L'arbre vide s'écrit '().

Les arbres binaires en Scheme

Fonctions de base

- define (consArbre r g d)
  (cons r (cons g (cons d '()))))
- (define (racine A)
  (car A))
- (define (filsGauche A)
  (cadr A))
- (define (filsDroit A)
  (caddr A))
- (define (estVide A)
  (null ? A))

Les arbres binaires en Scheme

Exemple

> (define A (consArbre 1 (consArbre 2 (consArbre 3 '() '()) (consArbre 4 '() '())) '()))

Dessinez l'arbre construit. Que donnent les expressions :
> (racine A)
> (filsGauche A)
> (filsDroit A)
> (filsDroit (filsGauche A))
> (filsDroit (filsDroit A))

Les arbres binaires en Scheme

Exemple

```
> (define A (consArbre 1 (consArbre 2 (consArbre 3 '() '()) (consArbre 4 '() '())) '()))

> (racine A)
1
> (filsGauche A)
(2 (3 () ()) (4 () ()))
> (filsDroit A)
()
> (filsDroit (filsGauche A))
(4 () ())
> (filsDroit (filsDroit A))
erreur
```

Les arbres binaires en Scheme

Exercice : Calculer la taille d'un arbre

Exercice : Calculer la hauteur d'un arbre

Les arbres binaires en Scheme

Exercice : Calculer la taille d'un arbre

```
(define (taille A)
    (if (estVide A)
        0
        (+ 1 (taille (filsGauche A)) (taille (filsDroit A)))))
```

Exercice : Calculer la hauteur d'un arbre

```
(define (hauteur A)
    (if (estVide A)
        -1
        (+ 1 (max (hauteur (filsGauche A)) (hauteur (filsDroit A))))))
```