Chapitre 3
Les structures arborescentes

Introduction:

Inconvénients des structures séquentielles :
– En format contigu, les mises à jour sont fastidieuses,
– En format chaîné, les parcours sont de complexité linéaire.

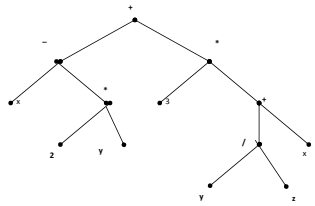
 Les structures arborescentes permettent une amélioration globale des accès aux informations

Place des arbres en informatique

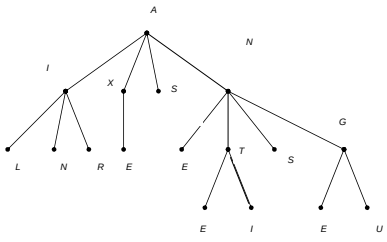
La structure d'arbre est l'une des structures les plus importantes et des plus spécifiques de l'informatique.

- Les systèmes d'exploitation (Windows, Unix) organisent les fichiers/répertoires sous forme d'arbre.
- Les compilateurs utilisent la structure d'arbre pour représenter la syntaxe d'un programme et détecter les éventuelles erreurs syntaxiques.
- Les arbres sont les structures de données, les mieux adaptées aux algorithmes issus de l'intelligence artificielle.

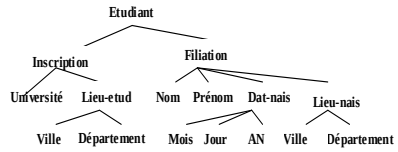
***Exemple* 1**: une expression arithmétique peut être représentée par un arbre en utilisant les priorités usuelles:
((x – *(2*y)*) + (3*(**(y / z)** + x)))



***Exemple* 2 :** Structure d'un dictionnaire composé des mots suivants
{AIL, AIN, AIR, AXE, AS, ANE, ANTE, ANTI, ANS, ANGE, ANGU}



***Exemple* 3**: Un type de donnée structuré peut être représenté sous format d'arbre dans tout langage de programmation :
Type Date **= structure {**Jour, Mois, An : **entier} ;**
Type Lieu **= structure {** Ville : Chaîne (30) ; Département: Chaîne (20) **} ;**
Type Identité **= structure { Nom, Prénom**: Chaîne (30) ; Dat-Nais : date ;Lieu-nais : Lieu**} ;**
Type Etablissement **= structure {** Université: Chaîne (30) ; Lieu-etud : Lieu**} ;**
Type Etudiant **= structure {**Inscription **:** Etablissement ; Filiation : Identité**}** ;



Propriété des arbres

Une propriété inhérente à la structure d'arbre est la récursivité. On écrit très naturellement de manière récursive :

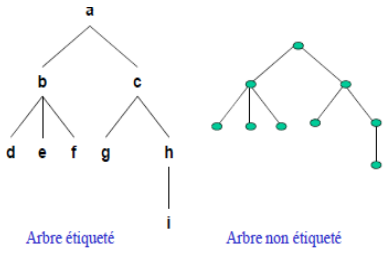
- Les définitions caractéristiques des arbres
- Les algorithmes qui manipulent les arbres

Qu'est-ce qu'un arbre ?

Un arbre A est un ensemble fini, éventuellement vide, de nœuds organisés hiérarchiquement comme suit :

- Il existe un nœud particulier appelé **racine de l'arbre A.**
- Les autres nœuds (s'ils existent) sont organisés en m classes disjointes A1, A2, Am. Chacune de ces classes étant elle même structurée en arbre : On parle des « sous-arbres » de la racine.

Exemples:



Terminologie associée aux arbres

La terminologie utilisée pour les arbres informatiques s'inspire du vocabulaire de la botanique et de la généalogie.

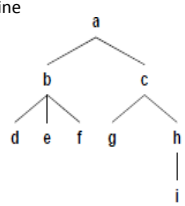
Notion de noeud:

Tout nœud d'un arbre donne accès à :

- Un élément de l'arbre (si l'arbre est étiqueté)
- Aux nœuds du sous-arbre dont il est la racine

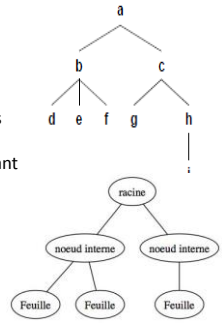
Relations entre nœuds:

Le noeud b est **père des nœuds** d, e et f
Les noeuds d, e et f sont les **fils** du noeud b
Les noeuds d, e et f sont **frères**
Les noeuds a, c et h sont des **ascendants ou ancêtres du** noeud i
Le noeud i est un **descendant des nœuds a, c et h**



Feuilles et branches:

- Un nœud qui n'a pas de fils est un Nœud **externe ou nœud terminal** ou **feuille. Les nœuds d, e, f, g et i** sont des feuilles.
- Un nœud qui a au moins un fils est un nœud interne ou nœud non terminal. Les nœuds **b, c, h** sont internes
- On appelle branche de l'arbre A tout chemin (suite de nœuds consécutifs) allant de la racine à une feuille de A.
- Un arbre a donc autant de branches que de feuilles.



Pour calculer la hauteur d'un arbre, nous allons nous baser sur la définition récursive :

Notion de Profondeur (Hauteur)

❖**La profondeur d'un nœud : (ou *hauteur ou niveau) est le*** nombre de liens du chemin allant de la racine à ce nœud.

La profondeur P d'un nœud x d'un arbre A est définie récursivement comme suit :

$$P(x) = 0 \text{ si x est la racine de l'arbre A}$$
$$P(x) = P(y) + 1 \text{ si y est le père de x}$$

❖**La profondeur d'un arbre : (ou *hauteur) est le niveau*** maximal de ses feuilles.

$$P(A) = \max(h(x)) \quad x \text{ feuilles de l'arbre A}$$

Donc Pour calculer la hauteur d'un arbre, nous allons nous baser sur la définition récursive :

- -un arbre vide est de hauteur 0
- -un arbre non vide a pour hauteur 1 + la hauteur maximale entre ses fils.

Importants

- La hauteur de l'arbre est alors la profondeur maximale de ses nœud. C'est à dire la profondeur à laquelle il faut descendre dans l'arbre pour trouver le nœud le plus loin de la racine.
- On peut aussi définir la hauteur de manière récursive : la hauteur d'un arbre est le maximum des hauteur de ses fils.
- C'est à partir de cette définition que nous pourrons exprimer un algorithme de calcul de la hauteur de l'arbre.
- La hauteur d'un arbre est très importante. En effet, c'est un repère de performance. La plupart des algorithmes que nous verrons dans la suite ont une complexité qui dépend de la hauteur de l'arbre.
- Ainsi plus l'arbre aura une hauteur élevée, plus l'algorithme mettra du temps à s'exécuter.

3-1 LES ARBRES BINAIRES

Qu'est-ce qu'un arbre binaire ?

•Un **arbre binaire est un arbre ordonné tel que tout noeud a** au plus deux fils.

•La structure d'arbre binaire est utilisée dans de très nombreuses applications informatiques.

•De plus les arbres généraux peuvent toujours être représentés par des arbres binaires.

•Pour les Exemples précédents:

L'exemple 1 est représenté par un arbre binaire, alors que les exemples 2 et 3 sont des arbres généraux.

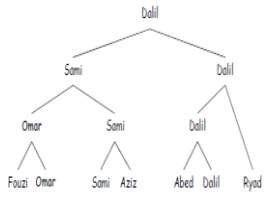
•***Définition*** : un arbre binaire est soit vide, soit de la forme B = <0, B1, B2> où B1 et B2 sont des arbres binaires disjoints et '0' un nœud appelé racine.

Remarque:

'0' est appelé racine de l'arbre B, B1 est appelé sous arbre gauche, B2 sous arbre droit de B.

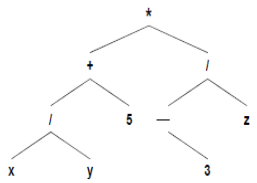
Il faut noter la non symétrie gauche et droite des arbres binaires <0, B1, B2> ≠ <0, B2, B1>.

Exemple 1



Représentation des résultats d'un tournoi de tennis

Exemple 2



Représentation de l'expression : (x/y+5)*(-3/z)

Définition

Un **arbre binaire est un ensemble fini qui est :**

➢ou bien vide

➢ou bien composé d'une racine et de deux sous-arbres binaires respectivement appelés sous-arbre gauche et sous-arbre droit.

➢**Notations**

Arbre binaire B vide

B = Φ

Arbre binaire B non vide

B = < r, Bg, Bd >

Définition récursive d'un arbre binaire

B = Φ + < r, B, B >

Arbres binaires particuliers :

1-Arbre binaire dégénéré

Un arbre binaire *dégénéré ou filiforme est un arbre où tout nœud* autre qu'une feuille a un seul fils.

Exemple d'arbre binaire dégénéré

Un arbre binaire dégénéré est un arbre formé uniquement de points simples: c'est une liste.

2-Arbre binaire complet

•Un arbre binaire *complet est un arbre où tout noeud autre qu'une* feuille a deux fils et toutes les feuilles ont même niveau.

•On dit que chaque niveau d'un tel arbre est complètement rempli, c'est-à-dire, que l'arbre contient :

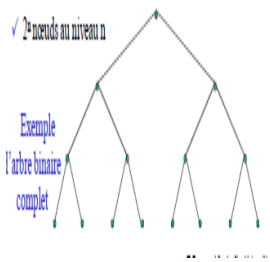
1 nœud au niveau 0

2 nœuds au niveau 1

4 nœuds au niveau 2

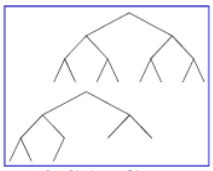
…

2^n nœuds au niveau n

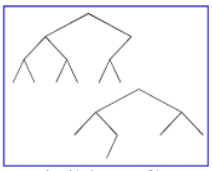


Arbre binaire parfait

Un arbre binaire *parfait est un arbre où chaque niveau est complètement* rempli, sauf éventuellement le dernier niveau. Cependant, si le dernier niveau n'est pas complètement rempli, les Nœuds présents sont regroupés à gauche de l'arbre.



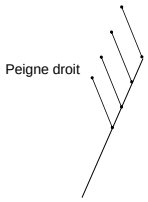
arbres binaires parfaits



arbres binaires non parfaits

•Un arbre binaire <0, B1, B2> est dit arbre équilibré si les tailles de B1 et B2 sont égales.

•Un peigne gauche (respectivement peigne droit) est un arbre binaire localement complet dans lequel tout fils droit (respectivement gauche) est une feuille.



3-2 Représentation chaînée par pointeurs

Compte tenu de la nature récursive d'un arbre, la représentation la plus couramment utilisée est celle chaînée dont le principe est de désigner un arbre binaire par un ensemble de cellules mémoire à trois champs :

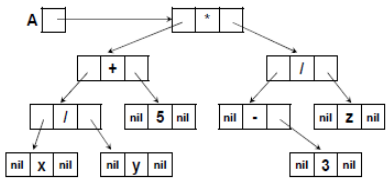
G	Val	D

➢G est un pointeur vers le sous arbre gauche.
➢ Val représente l'information apportée par le nœud pour un arbre étiqueté.
➢ D est un pointeur vers le sous arbre droit.

➢L'accès à un arbre est donc déterminé par l'adresse (pointeur) de sa racine.

On a la définition (**déclaration**) suivante :
Type cellule = Structure {Val : type; G, D : pointeur sur cellule};
Type Arbre = pointeur sur cellule;

Exemple:



Représentation chainée de l'arbre A de l'expression : (x/y+5)*(-3/z)

Remarques:

Arbre A est vide : A = nil
Accès à l'élément (étiquette) du nœud A : *A.val
Le sous-arbre gauche de A est non vide si : *A.g ≠ nil
Accès au sous-arbre gauche de A : *A.g
Le sous-arbre droit de A est non vide si : *A.d ≠ nil
Accès au sous-arbre droit de A : *A.d

3-3 Parcours d'un arbre binaire

Parcours d'un arbre binaire:

➢L'opération de parcours d'un arbre consiste à examiner systématiquement (visiter), dans un certain ordre, chacun des nœuds de l'arbre pour y effectuer un même traitement.

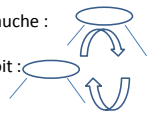
➢L'algorithme le plus utilisé appelé parcours en profondeur à main gauche consiste à tourner autours de l'arbre en partant à gauche de la racine et en allant toujours le plus à gauche possible en suivant l'arbre.

➢Dans ce parcours, chaque nœud est rencontré exactement 3 fois:

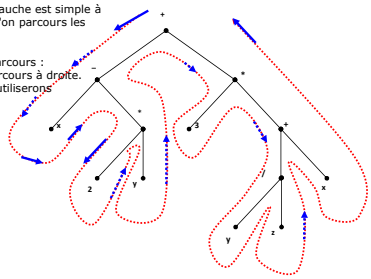
1- A la descente:

2- En remontée à gauche après le sous arbre gauche :

3- En remontée à droite après le sous arbre droit :



Un parcours en profondeur à gauche est simple à comprendre, cela signifie que l'on parcours les branches de gauche
avant les branches de droite.
On aura donc deux types de parcours :
un parcours à gauche et un parcours à droite.
Dans la plupart des cas, nous utiliserons un parcours à gauche.



Exemple : Parcours en profondeur à main gauche de l'arbre binaire associé à l'expression arithmétique (X-(2*Y))+(3*((Y/Z)+X))

Algorithme récursif de parcours :

❖Lors d'un parcours en profondeur à main gauche d'un arbre B, on peut faire correspondre à chacune des rencontres (passage) d'un nœud un traitement particulier.

❖Soient traitement 1 (respectivement traitement 2 puis traitement 3) la suite d'actions à exécuter lorsqu'on passe pour la première fois (respectivement la deuxième fois puis la troisième fois).

❖On envisage pour les arbres vides un traitement spécial appelé TERMINAISON.

❖On Suppose de plus que B est implémenté sous forme chaînée.

Algorithme récursif de parcours

Procédure **Parcours** (↓↑ B :ArbreP)
Début
Si (B = Vide) alors TERMINAISON
Sinon
Traitement 1;
Parcours (*B.G);
Traitement 2;
Parcours (*B.D);
Traitement 3;
FSi;
FIN.

La deuxième caractéristique de notre arbre est:

➢ le parcours dit préfixe. Cela signifie que l'on affiche la racine de l'arbre, on parcourt tout le sous arbre de gauche, une fois qu'il n'y a plus de sous arbre gauche on parcourt les éléments du sous arbre droit.

Ce type de parcours peut être résumé en trois lettres :
R G D (pour Racine Gauche Droit).
On a aussi deux autres types de parcours :
le parcours infixe et le parcours suffixe (appelé aussi postfixe).

➢Le parcours infixe affiche la racine après avoir traité le sous arbre gauche, après traitement de la racine, on traite le sous arbre droit
(c'est donc un parcours G R D).

➢ Le parcours postfixe effectue donc le dernier type de schéma :
sous arbre gauche, sous arbre droit puis la racine, c'est donc un parcours G D R.

Bien sur, nous avons fait l'hypothèse
d'un parcours à gauche d'abord mais on aurait pû aussi faire un parcours à droite d'abord.

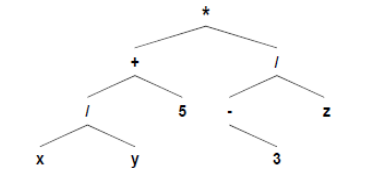
➢Les types de parcours infixe, suffixe et postfixe sont les plus importants, en effet chacun à son application particulière.

Ordres de parcours en profondeur:

Donc Le parcours en profondeur à main gauche contient comme cas particuliers 3 ordres classiques d'exploration d'arbre :

1. Ordre préfixé ou parcours en pré-ordre : les nœuds sont traités à la descente (1ère rencontre). Le père est traité avant les fils.
2. Ordre infixé ou symétrique : les nœuds sont traités à la remontée gauche (2ème rencontre). Le père est traité après son fils gauche et avant son fils droit.
3. Ordre postfixé ou parcours en post-ordre ou terminal : les Nœuds sont traités à la remontée droite (3ème rencontre). Le père est traité après les fils.

Exemple:



Résultat du parcours pour imprimer les éléments de l'arbre :

- En ordre préfixé : *+/xy5/-3z
- En ordre postfixé : xy/5+3-z/*
- En ordre infixé : x/y+5*-3/z

Procédure préordreréc (↓↑A : racine de l'arbre)
début
si (A ≠ arbrevide) Alors traiter (A)
préordreréc (g(A))
préordreréc (d(A))
Fsi;
Fin.
NB: A est un élément d'accès à la racine de l'arbre
(pointeur ou indice)

Traitement 2 et Traitement 3 n'existent pas, Traitement 1 est appliqué aux nœuds de l'arbre.
Ce parcours consiste à effectuer dans l'ordre :
•Le traitement de la racine ; **R**
•Le parcours du sous arbre gauche ; **G**
•Le parcours du sous arbre droit. **D**

Procédure infixéréc (↓↑A : racine de l'arbre)
début
Si (A ≠ arbrevide) alors infixéréc (g(A));
traiter (A);
infixéréc (d(A))
Fsi;
Fin.
NB: A est un élément d'accès à la racine de l'arbre
(pointeur ou indice)

C'est l'ordre obtenu lorsque seul traitement 2 (et terminaison) sont appliqués.
Ce parcours consiste à effectuer dans l'ordre :
•Le parcours du sous arbre gauche ; **G**
•Le traitement de la racine ; **R**
•Le parcours du sous arbre droit. **D**

Procédure postordreréc (↓↑A : racine de l'arbre)
début
si (A ≠arbrevide) alors postordreréc (g(A));
postordreréc (d(A));
traiter (A)
Fsi;
Fin.
NB: A est un élément d'accès à la racine de l'arbre
(pointeur ou indice)
seules les actions traitement 3 et terminaison sont appliquées.
Ce parcours consiste à effectuer dans l'ordre :
▪Le parcours du sous arbre gauche ; **G**
▪Le parcours du sous arbre droit ; **D**
▪Le traitement de la racine. **R**

Exemples:
1-soit la fonction taille qui définie la taille d'un arbre Binaire ABR

Fonction Taille (ABR : PTR) :Entier
Début
Si (ABR = NIL) alors Taille ← 0
Sinon Taille ← 1+ Taille(*ABR.G)+ Taille (*ABR.D)
Fsi ;
Fin.
2-soit la fonction Somme qui calcule la somme des nœuds d'un arbre Binaire ABR.

Fonction Somme (ABR : PTR) : Entier
Début
Si (ABR= Nil) alors Somme ← 0
Sinon Somme ← Somme (*ABR.G) + Somme (*ABR.D) + *ABR.val
Fsi ;
Fin.

3- la fonction moyenne qui calcule la moyenne des nœuds d'un Arbre Binaire ABR.

Fonction Moyenne (ABR : PTR) : Réel
Début
Si (Taille (ABR)≠ 0)) alors Moyenne ← Somme(ABR) / Taille (ABR)
Fsi ;
Fin.