CHAPITRE 8

Les arbres binaires

1. Introduction

Un arbre impose une structure hiérarchique à un ensemble d'objets. Un arbre généalogique ou l'organigramme d'une entreprise en sont des exemples qui nous sont familiers.

L'arbre est une structure de données fondamentale en informatique, très utilisé dans tous les domaines, parce que bien adaptée à la représentation naturelle d'informations homogènes organisées, et d'une grande commodité et rapidité de manipulation.

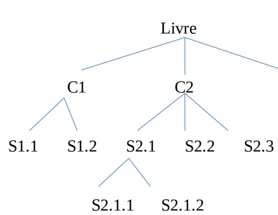
L'usage des arbre est multiple, car il capte l'idée de hiérarchie; à titre d'exemples, nous pouvons citer:

- découpage d'un livre en parties, chapitres, sections, paragraphes…,
- hiérarchies de fichiers,
- expressions arithmétiques
- …

1. Introduction

Exemples:

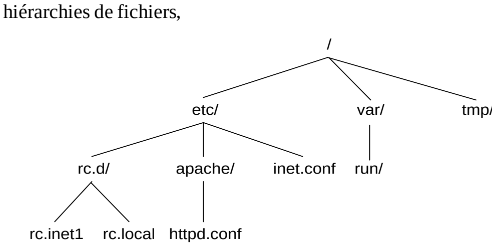
- découpage d'un livre en parties, chapitres, sections, paragraphes…,



1. Introduction

Exemples:

- hiérarchies de fichiers,

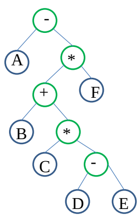


1. Introduction

Exemples:

- expressions arithmétiques,

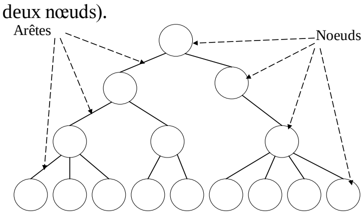
L'expression A - (B + C * (D - E)) * F se représente facilement par un arbre où apparaît clairement la priorité des opérations:



2. Définitions et Terminologie

- **Définition non récursive**

Un arbre est un ensemble (éventuellement vide) de nœuds (un nœud peut contenir un certain nombre d'information) et d'arêtes (lien entre deux nœuds).



→ Un arbre est une structure de données hiérarchique (non linéaire) représenté de haut en bas

2. Définitions et Terminologie

- **Définition récursive**

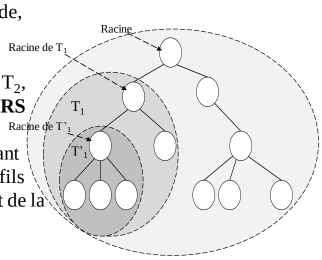
Cas particulier:

NIL est un arbre (l'arbre vide, contenant zéro nœud)

Cas général:

SI n est un nœud et si T_1, T_2, ...T_m sont des arbres, **ALORS** on peut construire un nouvel arbre en connectant T_1, T_2, ...T_m comme des fils à n. Chaque T_i est définit de la même manière (récursivement).

T_1, T_2, ...T_m sont alors des sous- arbres de n.



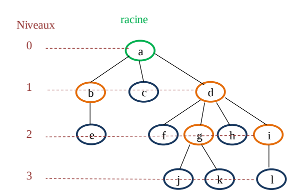
2. Définitions et Terminologie

- **Terminologies**

☑ Racine : C'est le nœud qui n'a pas de prédécesseur. La racine constitue la caractéristique d'un arbre. Dans l'exemple c'est a.

☑ Feuille : c'est le nœud qui n'a pas de successeur. Une feuille est aussi appelée un nœud externe.

Dans l'exemple, les feuilles sont : e, c, f, j, k, h, et l.



☑ Nœud interne : c'est tout nœud qui admet au moins un successeur. Dans l'exemple, les nœuds internes sont : a, b, d, g et i.

☑ Fils d'un nœud : ce sont ses successeurs.

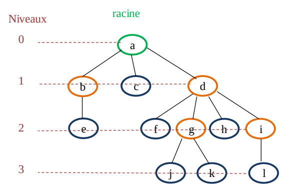
Dans l'exemple, f, g, h, et i sont les fils du nœud d.

☑ Frères : ce sont les successeurs issus d'un même nœud.

Dans l'exemple, b, c et d sont des frères.

2. Définitions et Terminologie

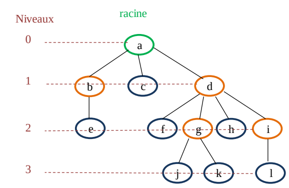
- **Terminologies**
  - ☑ Père : c'est un nœud qui admet au moins un successeur. Dans l'exemple, d est le père des nœuds f, g, h et i.
  - ☑ Sous arbre : C'est une portion de l'arbre. Dans l'exemple, le nœud g avec ces deux fils j et k constituent un sous arbre.



  - ☑ Descendants d'un nœud : ce sont tous les nœuds du sous arbre de racine nœud. Dans l'exemple, les descendants de d sont d, f, g, h, i, j, k et l.
  - ☑ Ascendants d'un nœud : ce sont tous les nœuds se trouvant sur la branche de la racine vers ce nœud. Dans l'exemple, les ascendants de j sont g, d et a. Les ascendants de e sont b et a.
  - ☑ Branche : est une suite de nœuds connectés de père en fils (de la racine à une feuille). Dans l'exemple, les branches de l'arbre sont a-b-e, a-c, a-d-f, a-d-g-j, …

2. Définitions et Terminologie

- **Terminologies**
  - ☑ Degré d'un nœud : C'est le nombre de ses fils. Dans l'exemple, le degré de b est 1, le degré de d est 4.
  - ☑ Niveau d'un nœud : est la distance qui le sépare de la racine. La racine a le niveau 0, ses fils ont le niveau 1, les fils des fils ont le niveau 2. etc… Dans l'exemple, a est de niveau 0, d est de niveau 1, g est de niveau 2, et k est de niveau 3.



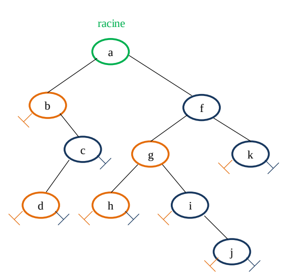
  - ☑ Profondeur de l'arbre : (ou sa hauteur) est le plus grand niveau (càd la distance entre la racine et la feuille la plus lointaine). L'arbre de l'exemple est de profondeur 3.
  - ☑ Forêt : C'est un ensemble d'arbres.

3. Les arbres binaires

3. 1. Définition

- Un arbre binaire est un arbre où chaque nœud est connecté à deux sous-arbres (un sous-arbre gauche et un sous-arbre droit). Donc un arbre binaire est un arbre de degré 2, c'est-à-dire que chaque nœuds a **au plus deux fils**.

On désigne chacun des fils par les appellations *fils gauche* et *fils droit*.



3. Les arbres binaires

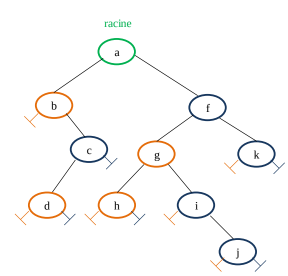
3. 1. Définition

- **Arbre strictement binaire**

Un arbre est dit strictement binaire si chaque nœud qui n'est pas une feuille a exactement deux fils.

Si un arbre strictement binaire a n feuilles Alors

- ❑ le nombre total de ses nœuds = 2n-1.
- ❑ le nombre de ses nœuds non feuilles (nœuds internes) = n-1



Cet arbre n'est pas strictement binaire

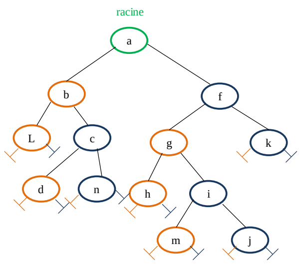
3. Les arbres binaires

3. 1. Définition

- **Arbre strictement binaire**

Dans l'exemple:

- ❑ Nombre de feuilles n = 7
  L, d, n, h, m, j, k
- ❑ Nombre total des nœuds
  = 2n-1 = 13
  a, b, L, c, d, n, f, g, h, i, m, j, k
- ❑ Nombre des nœuds internes
  = n-1= 6
  b, a, c, f, g, i



Cet arbre est strictement binaire

3. Les arbres binaires

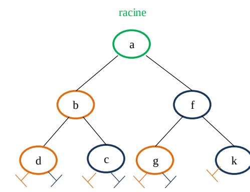
3. 1. Définition

- **Arbre binaire complet**

C'est un arbre strictement binaire où toutes les feuilles sont au même niveau.

Dans un arbre binaire complet de profondeur d :

- ❑ le nombre total de nœuds n
  $= 2^0 + 2^1 + 2^2 + ... 2^d = 2^{d+1}-1$
- ❑ le nombre de nœuds internes
  $= 2^d-1$
- ❑ le nombre de feuilles = 2^d
- ❑ le nombre de nœuds dans le niveau i = 2^i



On peut déduire une équation entre la profondeur (d) d'un arbre binaire complet et le nombre total de nœuds (n) :

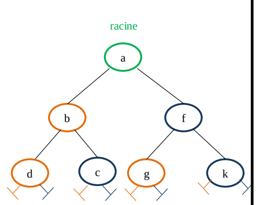
$d = \log_2(n+1) - 1$

3. Les arbres binaires

3. 1. Définition

- **Arbre binaire complet**

Dans l'exemple, la profondeur d = 2



- ❑ le nombre total de nœuds n
  $= 2^{d+1}-1 = 7$
- ❑ le nombre de nœuds internes
  $= 2^d-1 = 3$
- ❑ le nombre de feuilles = $2^d = 4$
- ❑ le nombre de nœuds dans le niveau i = 2^i
  niveau 0 = 1, niveau 1 = 2, niveau 2 = 4
- ❑ l'équation suivante: **$d = \log_2(n+1) - 1$** est bien vérifié
  avec n = 7 nœuds, $d = \log_2(n+1) - 1 = 2$

3. Les arbres binaires

3. 2. Le modèle

On définit le modèle (machine abstraite) suivant d'un arbre binaire:

Info(p)	permet d'accéder à l'information du nœud p
FG(p)	permet d'accéder à l'information de fils gauche du nœud p
FD(p)	permet d'accéder à l'information de fils droit du nœud p
Aff_info(p, x)	permet de modifier l'information du nœud p
Aff_FG(p, x)	permet de modifier l'information de fils gauche du nœud p
Aff_FD(p, x)	permet de modifier l'information de fils droit du nœud p
Creernoeud(x)	crée un nœud avec x comme information et retourne la référence du nœud. Le nœud créé a Nil comme fils gauche et droit.
Liberernoeud(p)	libère le nœud référencé par p.

3. Les arbres binaires

3. 3. Les parcours

- La façon de traiter l'information contenue dans un arbre binaire dépend de la manière dont ses nœuds sont visités.
- Trois méthodes de parcours sont couramment utilisées. Chacune correspond à une écriture différente.
  - Parcours postordre
  - Parcours préordre
  - Parcours en inordre
- Ces trois méthodes font appel à la nature récursive des arbres.
- Ces méthodes font tous parti des parcours en profondeur (depth-first)
- On peut aussi parcourir un arbre en largeur (par niveau ou breadth-first)

3. Les arbres binaires

3. 3. Les parcours

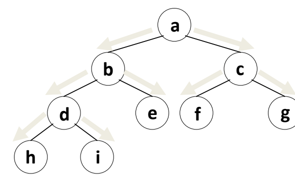
1) POSTORDRE (GDn)

Dans le parcours postordre, les descendants d'un nœud sont traités avant lui:

1. Fils de gauche
2. Fils de droite
3. Nœud

Résultat de parcours:

h i d e b f g c a



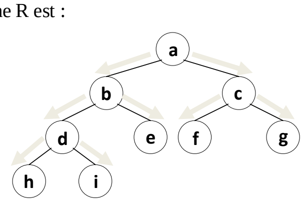
3. Les arbres binaires

3. 3. Les parcours

1) POSTORDRE (GDn)

La procédure (récursive) qui affiche les valeurs en parcours postordre d'un arbre de racine R est :

```
Postordre( R:ptr )
  debut
    SI R <> NIL
        Postordre( FG(R) )
        Postordre( FD(R) )
        ecrire( Info(R) )
    FSI
  fin
```



Résultat de parcours:

h i d e b f g c a

3. Les arbres binaires

3. 3. Les parcours

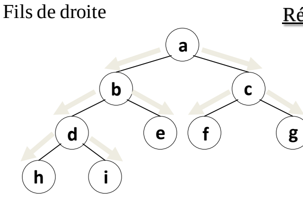
2) PREORDRE (nGD)

Dans le parcours préordre, les descendants d'un nœud sont traités après lui:

1. Nœud
2. Fils de gauche
3. Fils de droite

Résultat de parcours:

a b d h i e c f g



2) PREORDRE (nGD)

La procédure (récursive) qui affiche les valeurs en parcours préordre d’un arbre de racine R est :

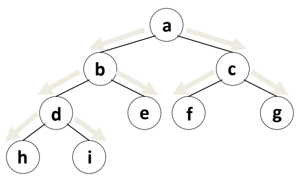
```
Préordre( R:ptr )
  debut
    SI R <> NIL
         ecrire( Info(R) )
         Préordre( FG(R) )
         Préordre( FD(R) )
    FSI
  fin
```

3. Les arbres binaires

3. 3. Les parcours

➔ **Par NIVEAU (en largeur) = breadth-first**

Dans le parcours par niveau, tous les nœuds d'un même niveau sont traités avant de descendre au niveau suivant.



Résultat de parcours:

a b c d e f g h i

3. Les arbres binaires

3. 4. Les arbres de recherche binaires (ARB)

Définition

- Les arbres de recherche binaires sont des arbres binaires structurés de façon à respecter la propriété suivante:

  Pour chaque nœud i :

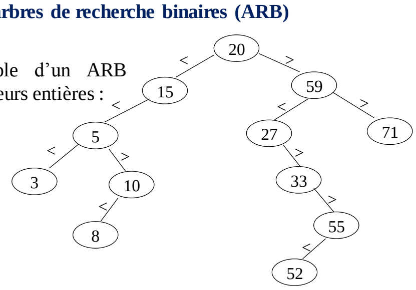
  - Tous les éléments de son **sous-arbre de gauche** ont une valeur **inférieure** à celle du nœud i.
  - Tous les éléments de son **sous-arbre de droit** ont une valeur **supérieure** à celle du nœud i.
- Toutes les valeurs dans un ARB sont distinctes (il n'y a pas de valeur en double).
- Le parcours en inordre d'un ARB donne la liste ordonnée de tous ses éléments.

3. Les arbres binaires

3. 4. Les arbres de recherche binaires (ARB)

Exemple

Voici un exemple d'un ARB contenant des valeurs entières :



Le parcours inordre de cet arbre donne la liste ordonnée suivante :
3, 5, 8, 10, 15, 20, 27, 33, 52, 55, 59, 71

3. Les arbres binaires

3. 4. Les arbres de recherche binaires (ARB)

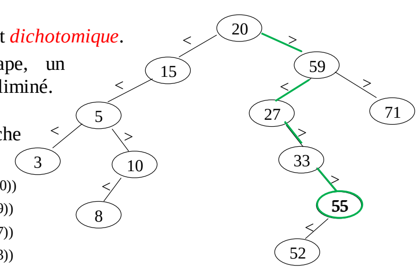
Opérations

1) Recherche

La recherche est *dichotomique*.

A chaque étape, un sous arbre est éliminé.
Par exemple,
on veut recherche
la valeur 55:

➔Rechercher (FD(20))
➔Rechercher(FG(59))
➔Rechercher(FD(27))
➔Rechercher(FD(33))
➔Élément trouvé



3. Les arbres binaires

3. 4. Les arbres de recherche binaires (ARB)

Opérations

2) Insertion

L'insertion d'un élément se fait toujours au niveau d'une feuille.

Cette insertion dans un ARB doit maintenir la propriété des arbres de recherche .

Algorithme d'insertion

- Rechercher la position d'insertion
- Raccorder le nouveau nœud à son parent

3. Les arbres binaires

3. 4. Les arbres de recherche binaires (ARB)

Opérations

2) Insertion

Pour insérer la valeur 25:

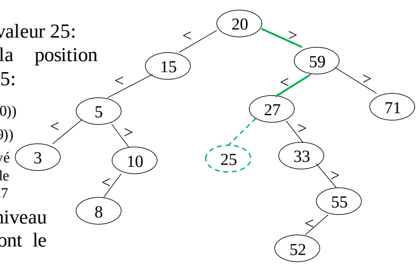
- Rechercher la position d'insertion de 25:

➔Rechercher (FD(20))

➔Rechercher(FG(59))

➔Position trouvé pour l'insertion, le père est le nœud 27

- Insérer 25 au niveau de la feuille dont le père est 27.



3. Les arbres binaires

3. 4. Les arbres de recherche binaires (ARB)

Opérations

3) Suppression

La suppression est un peu plus compliquée. Pour supprimer le nœud n d'un ARB, il faudra le rechercher. Un fois le nœud n trouvé, on se trouve dans une des situations suivantes :

n		Action
FG	**FD**	
Nil	Nil	Remplacer n par Nil
Nil	#Nil	Remplacer n par Fd(n)
#Nil	Nil	Remplacer n par Fg(n)
#Nil	#Nil	1. Rechercher le plus petit descendant du sous-arbre droit, soit p. 2. Remplacer Info(n) par Info(p) 3. Remplacer p par Fd(p)

3. Les arbres binaires

3. 4. Les arbres de recherche binaires (ARB)

Opérations

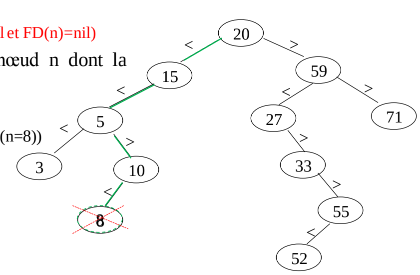
3) Suppression

Cas 1: FG(n)=nil et FD(n)=nil)

supprimer le nœud n dont la valeur est 8

➔Rechercher(valeur(n=8))

➔Libérer le nœud n



3. Les arbres binaires

3. 4. Les arbres de recherche binaires (ARB)

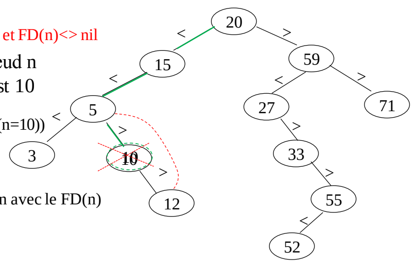
Opérations

3) Suppression

Cas 2: FG(n)=nil et FD(n)<> nil

supprimer le nœud n dont la valeur est 10

➔ Rechercher(valeur(n=10))



➔ Chainer le père de n avec le FD(n)

➔ Libérer le nœud n

3. Les arbres binaires

3. 4. Les arbres de recherche binaires (ARB)

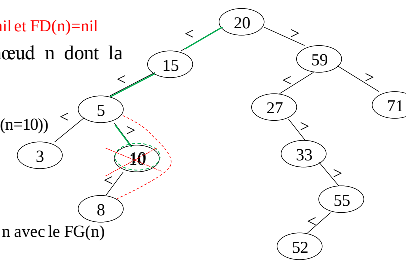
Opérations

3) Suppression

Cas 3: FG(n)<>nil et FD(n)=nil

supprimer le nœud n dont la valeur est 10

➔ Rechercher(valeur(n=10))



➔ Chainer le père de n avec le FG(n)

➔ Libérer le nœud n

3. Les arbres binaires

3. 4. Les arbres de recherche binaires (ARB)

Opérations

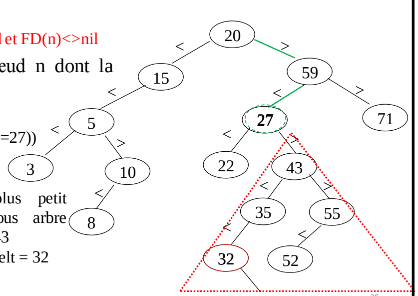
3) Suppression

Cas 4: FG(n)<>nil et FD(n)<>nil

supprimer le nœud n dont la valeur est 27

➔Rechercher(valeur(n=27))

➔ Rechercher le plus petit descendant du sous arbre droit: - sa racine = 43
- le plus petit elt = 32



3. Les arbres binaires

3. 4. Les arbres de recherche binaires (ARB)

Opérations

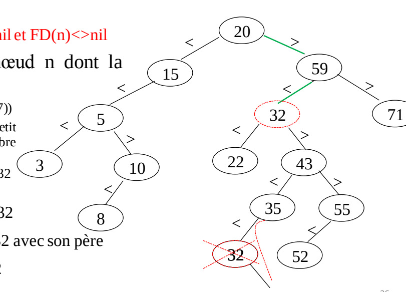
3) Suppression

Cas 4: FG(n)<>nil et FD(n)<>nil

supprimer le nœud n dont la valeur est 27

➔ Rechercher(valeur(n=27))

➔ Rechercher le plus petit descendant du sous arbre droit: - sa racine = 43
- le plus petit elt = 32



➔ Remplacer 27 par 32

➔ Chainer le FD de 32 avec son père

➔ Libérer le nœud 32

3. Les arbres binaires

3. 4. Les arbres de recherche binaires (ARB)

Opérations

Remarques

Toutes les opérations (recherche, insertion et suppression) sont efficaces (rapides) si l'arbre binaire est équilibré (ou presque), car le temps moyen est de l'ordre ***𝒪(log n)*** pour la plupart des opérations.

Par exemple, dans un arbre de recherche binaire équilibré contenant 1000 valeurs, il faudrait au maximum 10 tests (ou itérations) pour une recherche, insertion ou suppression.

De même qu'il en faudrait 20 tests si l'arbre renfermait 1000000 de valeurs, ...etc.

3. Les arbres binaires

3. 4. Les arbres de recherche binaires (ARB)

Opérations

Remarques

Le problème est que rien n'assure que l'arbre restera toujours équilibré (ou presque), si par exemple on insère, dans un arbre initialement vide, une suite ordonnée de 1000 valeurs, le résultat obtenu sera un arbre complètement 'dégénéré' qui ressemble en fait à une simple liste linéaire chaînée de 1000 maillons. Dans ce cas, le coût des opérations d'accès deviendra linéaire ***𝒪(n)*** (proportionnel au nombre de valeur).

Par exemple une recherche dans un tel arbre coutera au maximum 1000 tests, alors que dans un arbre dégénéré d'un million de valeurs, le coût sera d'un million de tests, ...etc.

3. Les arbres binaires

3. 4. Les arbres de recherche binaires (ARB)

Opérations

Remarques

Il existe d'autres algorithmes d'insertion et de suppression (un peu plus complexes) permettant de garder l'arbre équilibré dans tous les cas.

Parmi les techniques les plus connues, il y a : AVL-trees, RedBlack-trees, B-trees, T-trees, ...etc.

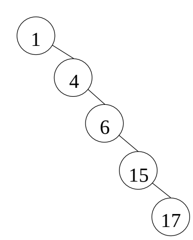
3. Les arbres binaires

3. 4. Les arbres de recherche binaires (ARB)

Opérations

Remarques

Exemple:
insérer en ordre les éléments 1,4,6,15,17



On tombe sur une simple liste.
Temps de recherche: 𝒪(n)

Solution: arbres équilibrés…

3. Les arbres binaires

3. 5. Implémentation des arbres binaires

Les arbres sont généralement des structures de données dynamiques. On peut donc représenter les arbres en dynamique et même en statique.

<u>En dynamique</u>

L'arbre binaire est un ensemble de nœuds (maillons) alloués dynamiquement. La structure d'un nœud de l'arbre est la suivante :

```
TYPE Tnoeud = STRUCTURE
    Info : Typeqq
    FG : POINTEUR(Tnoeud)
    FD : POINTEUR(Tnoeud)
FIN
VAR  Arbre : POINTEUR(Tnoeud)
```

```
struct Tnoeud {
    int Info;
    Tnœud* FG;
    Tnœud* FD;
  };
typedef  Tnœud* Arbre;
```


3. Les arbres binaires

3. 5. Implémentation des arbres binaires

<u>En statique</u>

1) Représentation standard

On range l'arbre dans un tableau. Chaque élément du tableau possède trois champs: un pour l'information, un pour le fils gauche et un pour le fils droit.

La structure du nœud est:

```
TYPE    Tnoeud= STRUCTURE
    Info : Typeqq
    FG : ENTIER
    FD :  ENTIER
FIN
```

Si la racine de l'arbre est toujours à la <u>position 1</u> du tableau, l'arbre sera défini:

```
VAR Arbre  =  TABLEAU[1..M] de Tnoeud
```

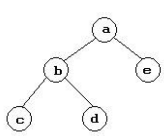

3. Les arbres binaires

3. 5. Implémentation des arbres binaires

<u>En statique</u>

1) Représentation standard

<u>Exemple</u>:



Racine → 1

	FG	info	FD
1	2	a	5
2	3	b	4
3	0	c	0
4	0	d	0
5	0	e	0

. . .
. . .

3. Les arbres binaires

3. 5. Implémentation des arbres binaires

<u>En statique</u>

1) Représentation standard

Si on veut que la racine soit n'importe où dans le tableau, alors l'arbre sera défini (le nœud reste inchangé):

```
TYPE Tarbre = STRUCTURE
      T : TABLEAU[1..M] de Tnoeud
      Racine : ENTIER
FIN
VAR  Arbre : Tarbre
```

Racine → 3

	FG	info	FD
1	0	c	0
2	1	b	4
3	2	a	5
4	0	d	0
5	0	e	0

. . .
. . .

3. Les arbres binaires

3. 5. Implémentation des arbres binaires

<u>En statique</u>

2) Représentation séquentielle

L'idée dans la représentation séquentielle c'est de ne pas représenter les indices des fils. On convient donc de ranger les éléments de l'arbre selon un ordre prédéfini.

Une représentation séquentielle pourrait être la suivante:

- La racine est à la position 1.
- Le nœud à la position p est le père des nœuds 2p (fils gauche) et 2p+1 (fils droit).
- Si un fils gauche est à la position p, son frère droit est à la position p+1. Si un fils droit est à la position p, son frère gauche est la position p-1.
  Père(p) c'est le nœud p DIV 2.

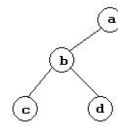
3. Les arbres binaires

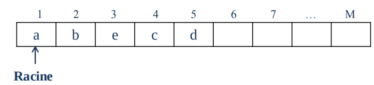
3. 5. Implémentation des arbres binaires

<u>En statique</u>

2) Représentation séquentielle

<u>Exemple</u>:





Chaque nœud à l'indice i a pour:

- nœud à l'indice (2i) = fils de gauche.
- nœud à l'indice (2i+1) = fils de droite.
- nœud à l'indice (i div 2) = père.

3. Les arbres binaires

3. 6. Exemples d'application des arbres binaires

Un arbre binaire est utile quand une décision sur deux doit être prise à chaque étape d'un traitement.

Nous donnons, dans ce qui suit, trois applications des arbres de recherche binaire:

1) Représentation des expressions arithmétiques
2) Représentation d'une LLC par un arbre binaire
3) Codage de Huffman

3. Les arbres binaires

3. 6. Exemples d'application des arbres binaires

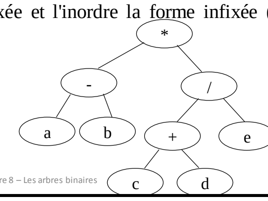
1) Représentation des expressions arithmétiques

- Les expressions arithmétiques peuvent êtres représentées sous forme d'arbre binaire. Les nœuds internes contiennent des opérateurs, alors que les feuilles contiennent des valeurs (opérandes).
- Le parcours en préordre donne la forme polonaise préfixée, le postordre la forme postfixée et l'inordre la forme infixée (forme normale sans parenthèses)

  Par exemple, l'expression:

  (a-b)((c+d)/e)*

  sera représentée par l'arbre suivant :



3. Les arbres binaires

3. 6. Exemples d'application des arbres binaires

1) Représentation des expressions arithmétiques

Pour développer l'algorithme récursif d'évaluation de l'expression arithmétique représentée par un arbre binaire, on utilise:

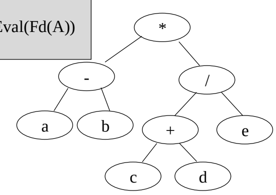
- le prédicat **Opérande(T)**:
  Si Info(T) est une opérande Alors Opérande = vrai
  Sinon Opérande = faux
- la fonction **Oper(Op, a, b)** effectue l'opération Op entre deux valeurs a et b.
- Soit A un arbre représentant une expression arithmétique. L'algorithme d'évaluation utilise le parcours postordre et est le suivant :

3. Les arbres binaires

3. 6. Exemples d'application des arbres binaires

1) Représentation des expressions arithmétiques

```
Fonction  Eval(A):entier
  SI Operande(A) alors
     Eval = Info(A)
  SINON
     Eval = Oper(Info(A), Eval(Fg(A), Eval(Fd(A))
  FSI
```



3. Les arbres binaires

3. 6. Exemples d'application des arbres binaires

2) Représentation d'une LLC par un arbre binaire

On représente une LLC par un arbre de recherche binaire comme suit:

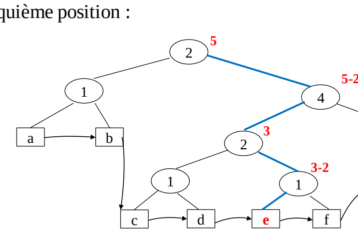
- Les feuilles de l'arbre représente les éléments de la LLC alors que les nœuds internes contient le nombre de feuilles (entier) de son sous arbre gauche.
- La représentation en arbre binaire d'une liste répond avec efficacité au problème de la recherche du K^{ème} élément et à la suppression d'un élément donné.
- Si la position recherchée est inférieure ou égale à l'information du nœud interne on descend à gauche, sinon on descend à droite et en retranche à la position recherchée l'information du nœud.

3. Les arbres binaires

3. 6. Exemples d'application des arbres binaires

2) Représentation d'une LLC par un arbre binaire

Le schéma suivant montre le déroulement de la recherche de la cinquième position :



3. Les arbres binaires

3. 6. Exemples d'application des arbres binaires

2) Représentation d'une LLC par un arbre binaire

L'algorithme qui suit recherche le $K^{ème}$ élément d'une liste représentée dans un arbre de racine Arbre. On utilise le prédicat Feuille(P) tels que:

si P est une feuille alors
  le prédicat est vrai
sinon
  le prédicat est faux.

```
R := K
P := Arbre
TQ NON Feuille(P) :
      SI R <= Compte(P) :
          P := FG(P)
      SINON
          R := R - Compte(P)
          P := FD(P)
      FSI
FTQ
```


3. Les arbres binaires

3. 6. Exemples d'application des arbres binaires

3) Codage de Huffman

Il est souvent pratique de compacter des données. Ceci peut sauver de l'espace sur disque et accélérer la vitesse de transfert des informations le cas échéant.

En général, dans un programme, on représente un ensemble d'éléments en affectant un code unique à chaque élément. Par exemple, dans le code standard ASCII, chaque caractère est représenté par 8 bits. Il faut un nombre minimum de bits pour représenter de façon unique chaque caractère. De façon générale, il faut log(n) bits pour représenter n valeurs différentes. On parle dans ce cas de codification de longueur fixe. Cette méthode serait la plus efficace si tous les caractères étaient utilisés de façon équitable.

3. Les arbres binaires

3. 6. Exemples d'application des arbres binaires

3) Codage de Huffman

Une autre approche consiste à trouver des codes de longueur variable. Cette codification tient compte de la fréquence d'apparition d'un caractère. Il est évident que les caractères les plus fréquemment utilisés devraient avoir des codes plus petits contrairement aux caractères les moins utilisés ont des codes plus importants..

Cette approche est utilisée par Huffman pour la compression de fichiers.

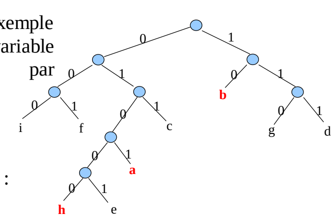
Pour coder ses caractères, Huffman commence par construire un arbre (appelé l'arbre de Huffman). Le code de chaque caractère est alors déduit de cet arbre.

3. Les arbres binaires

3. 6. Exemples d'application des arbres binaires

3) Codage de Huffman

L'arbre ci-contre est un exemple d'un codage à longueur variable pouvant être généré par l'algorithme de Huffman.



Dans cet exemple,

le code associé au symbole :

'a' est : 0101

'h' est : 01000

'b' est : 10

3. Les arbres binaires

3. 6. Exemples d'application des arbres binaires

3) Codage de Huffman

Avec ce type de codage (représenté par un arbre), la propriété du préfixe est toujours vérifiée:

«aucun code n'est le préfixe d'un autre» et c'est ce qui permet de décoder n'importe quelle chaîne de bits sans ambiguïté.

Par exemple, la chaîne **1001110101011110** ne peut être décodée que d'une seule manière, en la parcourant de gauche à droite, on est guidé dans l'arbre pour atteindre les feuilles. A chaque fois qu'on est sur une feuille, on aura décodé le caractère correspondant, on remonte à la racine et on continue le parcours de la chaîne de bit. On obtient alors le message décodé : **bcbbbdb**

3. Les arbres binaires

3. 6. Exemples d'application des arbres binaires

3) Codage de Huffman

Technique

1. Au départ on calcule les probabilités d'apparition de chaque symbole (caractère). C'est le nombre d'occurrence du caractère dans le message divisé par la taille du message (en nombre de caractères).
2. A chaque caractère du message, on construit un nœud (initialement isolé) contenant ce caractère.
3. Choisir les 2 nœuds ayant la plus faible probabilité et non encore choisit
4. Créer un nouveau nœud (un caractère fictif) en connectant les 2 nœuds de l'étape précédente comme fils gauche et fils droit du nouveau nœud. Associé à ce nouveau nœud la somme des probabilités de ces 2 fils.
5. Répéter les deux dernières étapes jusqu'à ce qu'il n'y ait qu'un seul nœud non encore traité. C'est alors la racine de l'arbre de Huffman.

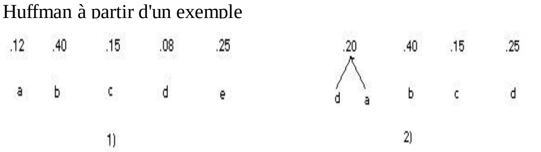
3. Les arbres binaires

3. 6. Exemples d'application des arbres binaires

3) Codage de Huffman

Scénario de l'algorithme de Huffman sur un exemple

Nous donnons, dans ce qui suit, le scénario de l'algorithme de Huffman à partir d'un exemple



calcul des probabilités d'apparition de chaque symbole et construction des nœuds pour chaque symbole

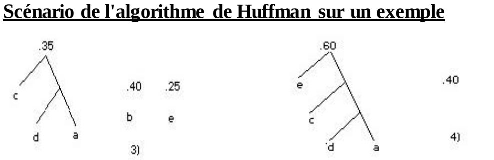
- choisir les 2 nœuds ayant la plus faible probabilité et non encore choisit
- création d'un nouveau nœud contenant la somme des probabilités de ces fils

3. Les arbres binaires

3. 6. Exemples d'application des arbres binaires

3) Codage de Huffman

Scénario de l'algorithme de Huffman sur un exemple



- choisir les 2 nœuds ayant la plus faible probabilité et non encore choisit
- création d'un nouveau nœud contenant la somme des probabilités de ces fils

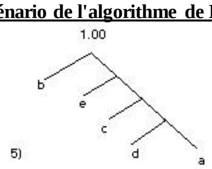
- choisir les 2 nœuds ayant la plus faible probabilité et non encore choisit
- création d'un nouveau nœud contenant la somme des probabilités de ces fils

3. Les arbres binaires

3. 6. Exemples d'application des arbres binaires

3) Codage de Huffman

<u>Scénario de l'algorithme de Huffman sur un exemple</u>



- choisir les 2 nœuds ayant la plus faible probabilité et non encore choisit
- création d'un nouveau nœud contenant la somme des probabilités de ces fils

Nous avons ainsi construit un arbre à partir d'une forêt. C'est ce qu'on appelle *l'arbre de Huffman*. Le code obtenu a une longueur moyenne de **2.15** et est le suivant:

a : 1111
b : 0
c : 110
d : 1110
e : 10

3. Les arbres binaires

3. 6. Exemples d'application des arbres binaires

3) Codage de Huffman

<u>Efficacité de l'algorithme de Huffman</u>

En théorie, cet algorithme est optimal en ce sens qu'il utilise moins d'espace pour effectuer un codage. Mais cela n'est vrai que si l'on connaît les vraies fréquences d'apparition des lettres, et cette fréquence est indépendante du contexte de la lettre dans le message. En pratique, la fréquence d'apparition d'une lettre change en fonction du contexte du message.

Le deuxième facteur qui peu affecter l'efficacité de la compression du codage de Huffman est la fréquence relative des lettres. Certaines fréquences ne donnent aucun gain par rapport au codage de longueur fixe, alors que d'autres vont générer un gain substantiel. Toutefois, on peut dire, qu'en général, le codage de Huffman performe mieux quand il y a une grande variation dans les fréquences d'apparition des lettres.