Les structures de données arborescentes

Isabelle Comyn-Wattiau

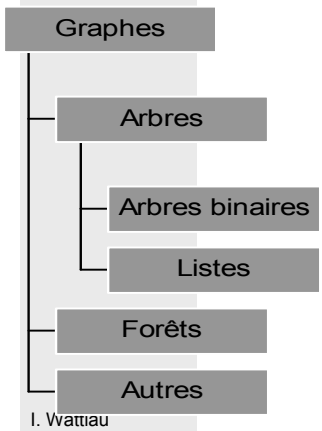
Introduction

- 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

Structure arborescente

- Organisation hiérarchique des informations
  - intuitive
  - conforme à beaucoup de situations :
    - arbre généalogique
    - tournois sportifs
    - structures syntaxiques
    - relations d'inclusions entre ensembles : classification des êtres vivants
    - décomposition de structures : type structuré, expression arithmétique, langue naturelle

Classification des structures

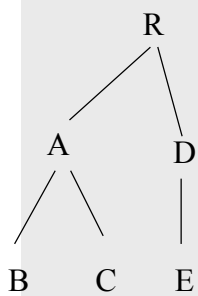


- Arbre = graphe purement hiérarchique
  - pas de cycle
  - un seul chemin d'un nœud à un autre
- Arbre binaire = tout nœud a au plus deux fils
- Liste = arbre dégénéré
- Forêt = ensemble d 'arbres

Définitions

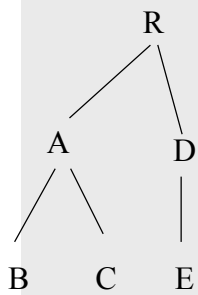
- Définition récursive d'un arbre :
  - Un arbre est
    - vide
    - ou constitué d'un élément et d'un ou plusieurs arbres

Définitions (suite)



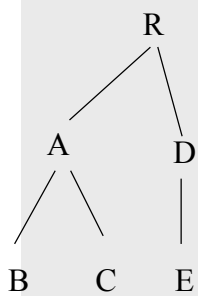
- R, A, B, C, D,E : nœuds ou sommets
- R racine, D - E sous-arbre, A – B – C sous-arbre
- Un nœud peut être fils d'un autre
  - C est un fils de A
- Un nœud peut être père d'un autre
  - D est le père de E, A est le père de C
- B, C, E sont des nœuds terminaux ou feuilles
- Tout chemin de la racine à une feuille est une branche

Définitions (suite)



- Nœuds d'un même niveau = issus d'un même nœud avec même nb de filiations
- Parcours en largeur =
  - par niveau R-A-D-B-C-E
- Taille d'un arbre = nb de nœuds = ici 6
- Hauteur d'un arbre = niveau maximum = ici 2
- Arbre dégénéré = au plus un descendant par noeud

Définitions (suite)

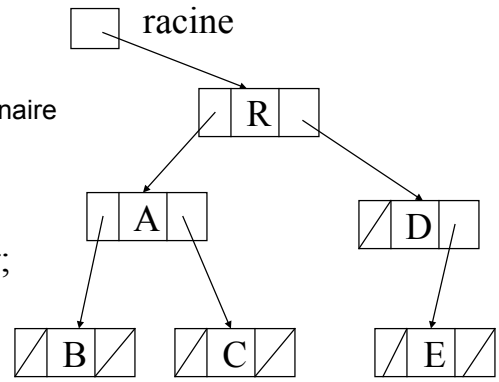


- Hauteur de l'arbre : longueur de sa plus longue branche
- Définition récursive
  - la hauteur d'un arbre est égale à la hauteur de sa racine
  - la hauteur d'un arbre vide est nulle
  - la hauteur d'un nœud est égale au maximum des hauteurs de ses sous-arbres plus un

Représentations d'un arbre

- graphique
- parenthésée
- chaînée

} binaire

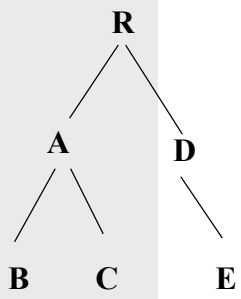



```
nœud = record
        gauche: pointeur;
        info:élément;
        droite:pointeur;
fin record;
```


Arbre binaire

- Si chaque nœud a au plus 2 fils, l'arbre est dit binaire
- Plus généralement, si chaque nœud a au plus n fils, l'arbre est n-aire
- Un arbre binaire est vide ou composé d'un élément auquel sont rattachés un sous-arbre gauche et un sous-arbre droit

Exemple d'arbre binaire



- Notation parenthésée :

  R (A (B,C), D (, E))

- On peut manipuler un arbre binaire avec trois primitives : valeur d'un nœud, fils gauche d'un nœud et fils droit d'un nœud.

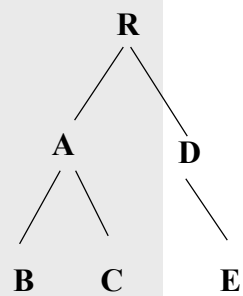
Arbre binaire complet

- Chaque nœud non terminal a exactement deux fils
- Arbre binaire complet au sens large ou arbre parfait :
  - l'avant-dernier niveau est complet
  - les feuilles du dernier niveau sont groupées le plus à gauche possible

Parcours d ’un arbre binaire

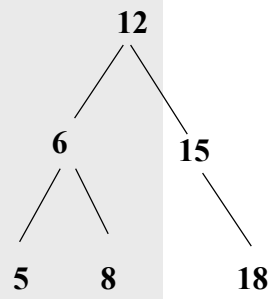
- Trois parcours possibles :
  - préfixé (préordre): on traite la racine, puis le sous-arbre gauche, puis le sous-arbre droit
  - infixé (projectif ou symétrique) : on traite le sous-arbre gauche, puis la racine, puis le sous-arbre droit
  - postfixé (ordre terminal) : on traite le sous-arbre gauche, le sous-arbre droit, puis la racine

Exemples de parcours



- Préfixé : R-A-B-C-D-E
- Infixé : B-A-C-R-D-E
- Postfixé : B-C-A-E-D-R

Arbre binaire ordonné



- La chaîne infixée des valeurs est ordonnée
- Tous les éléments dans le sous-arbre gauche d'un nœud lui sont inférieurs
- Tous les éléments dans son sous-arbre droit lui sont supérieurs

Parcours préfixé récursif

```
procédure préfixé(racine:pointeur in);
début
si racine < > nil alors
        traiter(racine);
        préfixé(racine↑.gauche);
        préfixé(racine↑.droite);
finsi
fin
```

Parcours infixé récursif

```
procédure infixé(racine:pointeur in);
début
si racine < > nil alors
        infixé(racine↑.gauche);
        traiter(racine);
        infixé(racine↑.droite);
finsi
fin
```


Parcours postfixé récursif

```
procédure postfixé(racine:pointeur in);
début
si racine <> nil alors
        postfixé(racine↑.gauche);
        postfixé(racine↑.droite);
        traiter(racine);
finsi
fin
```

Transformation du parcours préfixé en itératif - Etape 1

```
procédure préfixébis(racine:pointeur in);
racinebis:pointeur;
début
racinebis:=racine;
tant que racinebis <> nil faire
        traiter(racinebis);
        préfixébis(racinebis↑.gauche);
        racinebis:=racinebis↑.droite;
fintantque
fin
```

si devient tant que

facile car il n'y a pas de traitement après la récursivité

Transformation du parcours préfixé en itératif - Etape 2

```
procédure préfixébis(racine:pointeur in);
racinebis:pointeur;
début
initpilevide;
racinebis:=racine;
tant que (racinebis <> nil) ou (non pilevide) faire
        tant que racinebis <> nil faire
                traiter(racinebis);
                empiler(racinebis);
                racinebis:=racinebis↑.gauche
        fintantque;
        dépiler(racinebis);
        racinebis:=racinebis↑.droite;
fintantque
fin
```

Il faut une pile pour préserver les valeurs successives de la racine et passer au sous-arbre droit à chaque retour

Calcul de la taille d'un arbre binaire

```
fonction taille(racine:pointeur in):entier;
début
si racine = nil
        alors retourner 0
        sinon retourner 1+taille(racine↑.gauche)+taille(racine↑.droite);

finsi
fin
```


Nombre de feuilles d'un arbre binaire

```
fonction nbfeuille(racine:pointeur):entier;
début
si racine = nil
        alors retourner 0
        sinon
        si feuille(racine)
                alors retourner 1
                sinon retourner nbfeuille(racine↑.gauche)+nbfeuille(racine↑.droite)
        finsi
finsi
fin
```

```
fonction feuille(noeud:pointeur in):booléen;
début
retourner  (noeud↑.gauche=nil)
        et (noeud↑.droite=nil);
fin;
```

Vérifier qu'un arbre n'est pas dégénéré

```
fonction nondegener(racine:pointeur):booléen;
début
si racine = nil
        alors retourner faux
        sinon
        si (racine↑.gauche < > nil ) et (racine↑.droite < > nil)
                alors retourner vrai
                sinon
                si (racine↑.gauche = nil)
                        alors retourner nondegener(racine↑.droite)
                        sinon retourner nondegener(racine↑.gauche)
                finsi
        finsi
finsi
fin
```


Recherche par association dans un arbre binaire

```
fonction rechassoc(racine:pointeur;val:element):booléen;
début
si racine = nil
        alors retourner faux
        sinon
        si (racine↑.info = val )
                alors retourner vrai
                sinon
                si rechassoc(racine↑.gauche,val)
                        alors retourner vrai
                        sinon retourner rechassoc(racine↑.droite,val)
                finsi
        finsi
finsi
fin
```

Recherche par association dans un arbre binaire ordonné

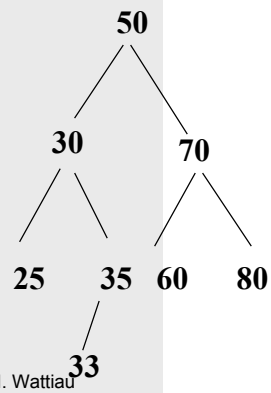
```
fonction rechdichoassoc(racine:pointeur;val:element):booléen;
début
si racine = nil
        alors retourner faux
        sinon
        si (racine↑.info =val )
                alors retourner vrai
                sinon
                si racine↑.info < val
                        alors retourner rechdichoassoc(racine↑.droite,val)
                        sinon retourner rechdichoassoc(racine↑.gauche,val)
                finsi
        finsi
finsi
fin
```

L'arbre est ordonné =>
On peut choisir le sous-arbre dans lequel il faut rechercher val

Insertion dans un arbre binaire ordonné (1)

- L'insertion doit maintenir l'ordre
- Si racine = nil, on crée un nœud racine avec l'élément à insérer
- Sinon on parcourt l'arbre en recherchant le nœud père de l'élément à insérer, puis on crée une feuille à rattacher à ce nœud, du bon côté
- Ce nœud père est soit une feuille soit un nœud avec un seul sous-arbre
- Il vérifie (pere↑.info <= val et pere↑.droite=nil) ou (pere↑.info > val et pere↑.gauche=nil)

Insertion dans un arbre binaire ordonné (2)



- Insertion de 27
- Le nœud père est 25
- 27 doit être inséré en sous-arbre droit

- Insertion de 40
- Le nœud père est 35
- 40 doit être inséré en sous-arbre droit

Insertion dans un arbre binaire ordonné (3)

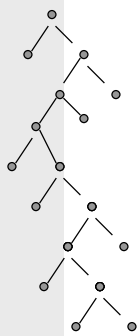
```
procedure insertion(racine:pointeur in out;
            val:element in);
début
si racine=nil
        alors creefeuille(val,racine);
        sinon
        si val >= racine↑.info
                alors insertion(racine↑.droite,val)
                sinon insertion(racine↑.gauche,val)
        finsi
finsi
fin
```

```
procedure creefeuille(val:element
in;feuille:pointeur out);
début
allouer(feuille);
feuille↑.info:=val;
feuille↑.gauche:=nil;
feuille↑.droite:=nil;
fin
```

Notions supplémentaires

- Longueur de cheminement de l'arbre = somme des longueurs de tous les chemins issus de la racine : LC
- Longueur de cheminement externe = somme des longueurs de toutes les branches issues de la racine : LCE
- Profondeur moyenne (d'un nœud) de l 'arbre = moyenne des hauteurs de tous les nœuds : LC/taille
- Profondeur moyenne externe (d'une feuille) de l'arbre = moyenne des longueurs de toutes les branches :LCE/nbfeuilles

Exemple



Taille = 17
Nbfeuilles = 9
Hauteur = 8
LC = 72
LCE = 44
Profondeur moyenne = 72/17=4.2
Profondeur moyenne externe = 44/9=4.9

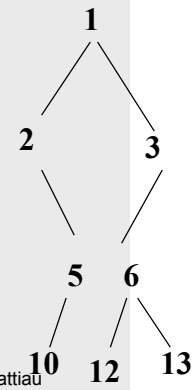
Complexité de la recherche d'un nœud ou d'une feuille

- Un nœud :
  - En moyenne, proportionnelle à la profondeur moyenne de l'arbre
  - Au pire, proportionnelle à la hauteur de l'arbre
- Une feuille :
  - En moyenne, proportionnelle à la profondeur moyenne externe de l'arbre
  - Au pire, proportionnelle à la hauteur de l'arbre

Quelques propriétés

- Taille >= 2 * nbfeuilles -1
- Taille = 2 * nbfeuilles - 1 pour un arbre complet
- Hauteur <= Taille - 1
- Hauteur = Taille - 1 pour un arbre dégénéré
- Hauteur = ent($\log_2$(taille)) pour un arbre parfait
- Taille <= $2^{\text{hauteur+1}} - 1$
- Taille = $2^{\text{hauteur+1}} - 1$ pour un arbre complet
- $\log_2$ (Taille) <= Hauteur <= Taille-1
- $\log_2$ (nbfeuilles) <= Hauteur
- Profondeur moyenne externe >= $\log_2$ (nbfeuilles)

Représentation contiguë dans un arbre binaire

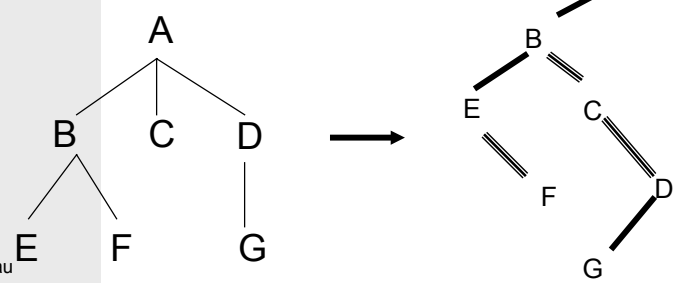


- Principe : on représente l'arbre en largeur
- Insertion et suppression immédiates
- Occupation mauvaise sauf pour les arbres complets

1	2	3		5	6				10		12	13

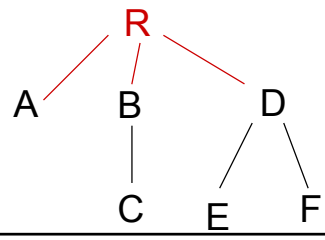
Arbres n-aires

- Principe : on peut se ramener à un arbre binaire avec des primitives fils aîné (fa ———) et frère cadet (fc ═══)



Forêt

- Un ensemble d'arbres n-aires
- On se ramène à un arbre n-aire en créant une racine virtuelle à laquelle rattacher les racines de tous les arbres n-aires



Conclusion

- Beaucoup d'applications : B-arbres, analyse syntaxique, etc.
- Généralisation des listes
- La recherche est généralement proportionnelle à la hauteur de l'arbre, intéressant si l'arbre est bien géré, c'est-à-dire dont la hauteur est proche du log de la taille
- L'implantation chaînée est généralement la plus adaptée