



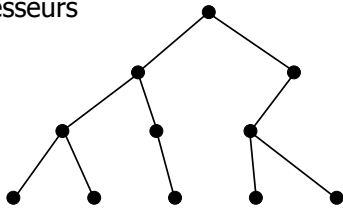
Les arbres



Structure arborescente

2

- Listes, files, piles: structures à 1 dimension
- Arbres: structure à plusieurs dimensions
- Arbre N-aire: chaque cellule possède au plus N successeurs






Structure arborescente

3

- Listes, files, piles: structures à 1 dimension
- Arbres: structure à plusieurs dimensions
- Arbre N-aire: chaque cellule possède au plus N successeurs
- Liste = arbre particulier (chaque cellule n'a qu'un seul successeur)
- Comme les listes, les arbres sont complètement dynamiques
- L'accès est séquentiel mais reste assez rapide


4



Arbres (dénomination)

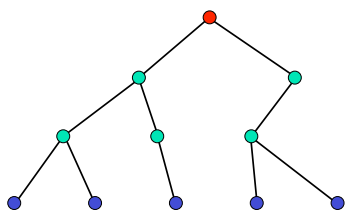
- Chaque cellule est nommée **nœud**
- Un nœud possède 1 seul prédécesseur (**parent**) et peut avoir plusieurs successeurs (**fil**s)
- Certains nœuds sont particuliers
 - **racine**: ne possède pas de parent
 - **feuille** de l'arbre: nœud sans successeur

5



Arbres (illustration)


- Arbre binaire (chaque nœud a au plus 2 fils)



● racine
● noeud
● feuille

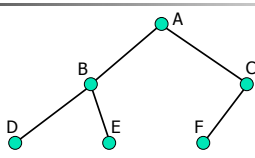
Dans la suite du cours, nous utiliserons le plus souvent les arbres binaires pour nos exemples

6



Arbres binaires représentation

- Graphique



- Parenthésée
 - $A (B (D , E) , C (F ,))$
- Structure
 - Définition structure{
 - info: typeVal;
 - *fdroit, *fgauche: Nœud;

Arbres: ex. d'applications

■ Représentation d'expressions

■ $(5 * (((9 + 8) * (4 * 6)) + 7))$

Arbres: ex. d'applications

■ Classification

Arbres: ex. d'applications

■ Traitement d'images

■ Quadtree (arbre quaternaire)

■ Déterminer des objets dans une image

■ arbre: quadtree

image initiale

Principe:
Si un critère défini au départ n'est pas vérifié, on divise l'image en 4 images plus petites, et ainsi de suite pour chaque portion d'image.

Critère: tous les pixels ont même valeur

Arbres: ex. d'applications

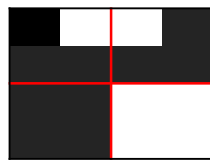
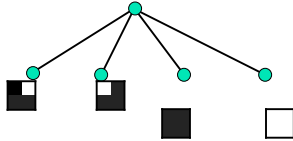
- Quadtree



Critère non vérifié:
division en 4

Arbres: ex. d'applications

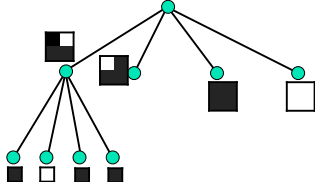
- Quadtree



Critère non vérifié:
division en 4

Arbres: ex. d'applications

- Quadtree



Critère non vérifié:
division en 4

[illegible]

13

Arbres: ex. d'applications

- Quadtree

Critère non vérifié:
division en 4

14

Arbres: ex. d'applications


- Quadtree

2ème phase: parcours
de l'arbre pour
regrouper les zones
adjacentes

15

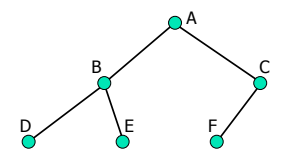
Arbres: définitions

- Niveau d'un nœud: hauteur du nœud dans l'arbre
 - exemple: niveau(racine) = 0
 - si $k \neq \text{racine}$, $\text{niveau}(k) = \text{niveau}(\text{parent de } k) + 1$
- Taille d'un arbre: nombre de nœuds de l'arbre
- Hauteur d'un arbre: $\max\{\text{niveau}(\text{feuilles})\}$
- Arbre binaire complet
 - chaque nœud (\neq feuille) admet 2 fils (exactement), et toutes les feuilles sont au même niveau
 - Taille d'un arbre binaire complet: $2^{k+1}-1$, où k est le niveau des feuilles (ou hauteur de l'arbre)




Arbres: définitions (ex.)

16



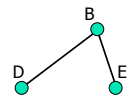
- niveau(A)=0
- niveau(B)=1
- niveau(F)=2
- Taille=6
- Hauteur=2



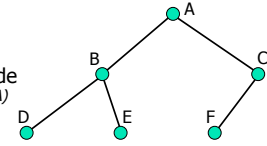
Arbres binaires: définitions


17

- Sous-arbre: T' est un sous-arbre de T si T' est contenu dans T



est un sous-arbre de
(sous-arbre gauche de A)





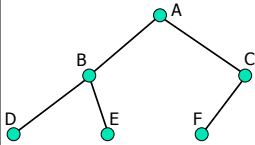
Arbres binaires: parcours

18

- Parcours = visiter les nœuds de l'arbre les uns après les autres dans un ordre donné
- Plusieurs types de parcours
 - **préfixe**
visiter le nœud racine puis son ss-arbreG de façon préfixe puis son ss-arbreD de façon préfixe
 - **infixe**
traverse le ss-arbreG de façon infixe puis la racine puis le ss-arbreD de façon infixe.
 - **postfixe**
traverse le ss-arbreG de façon postfixe puis le ss-arbreD de façon postfixe et enfin la racine.
 - **par niveau**
visite les nœuds niveau par niveau (début: racine)

19

Arbres: parcours (exemple)



- Préfixe (racine,ssG,ssD)
 - A , B , D , E , C , F
- Infixe (ssG,racine,ssD)
 - D , B , E , A , F , C
- Postfixe (ssG,ssD,racine)
 - D , E , B , F , C , A
- Par niveau
 - A , B , C , D , E , F

20

Arbres binaires: algorithmes de base

- Algorithmes présentés à l'aide de piles/files (algorithmes récursifs seront présentés la prochaine fois)
- Algorithmes
 - Parcours d'un arbre binaire
 - Taille d'un arbre binaire
 - Nombre de feuilles d'un arbre binaire
- Structure
 - struct
 - int valeur
 - Nœud *filsD, *filsG
 - fin struct Nœud

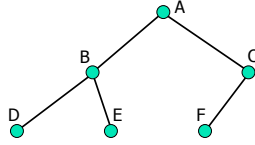
21

Algo de base (1)

- Parcours arbre binaire en préfixe (avec Pile)
 - Afficher le contenu des informations dans l'ordre donné

```

Procédure Parcours1(Nœud *racine, int val
Déclaration pile p
Nœud *n
Début
  si racine ≠ NULL alors
    empiler(p,racine);
    tantque est_vide(p) = faux faire
      n←p.sommet(p)
      dépiler(p);
      ecrire(n->valeur);
      si n->filsD ≠ NULL alors
        empiler(p,n->filsD);
      fin si
      si n->filsG ≠ NULL alors
        empiler(p,n->filsG);
      fin si
    fintantque
  fin
Fin
          
```

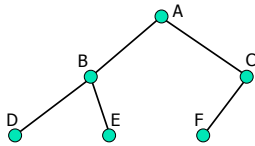




Algo de base (2)

■ Parcours arbre binaire par niveau (Avec File)

- Afficher le contenu des informations dans l'ordre donné



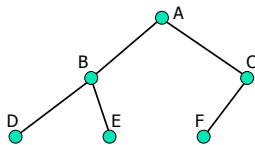
```

Procédure Parcours2(Noeud *racine, int val) )
Déclaration pile p
Noeud *n
Début
  si racine ≠ NULL alors
    ajoute_en_queue(f,racine);
    tantque est_vide(f)=faux faire
      n<-tete(f)
      retire_tete(f)
      ecrire(n->valeur);
      si n->filsG ≠ NULL alors
        ajoute_en_queue (f,n->filsG);
      finsi
      si n->filsD ≠ NULL alors
        ajoute_en_queue (f,n->filsD);
      finsi
    fintantque
  fin
Fin
  
```



Algo de base (3)

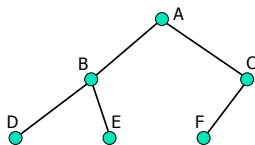
■ Ecrire l'algorithme qui donne la taille d'un arbre binaire






Algo de base (3)

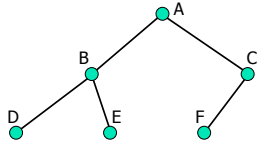
■ Ecrire l'algorithme qui donne le nombre de feuilles d'un arbre binaire






Réversivité (pour les arbres)

■ Parcours préfixe arbre binaire



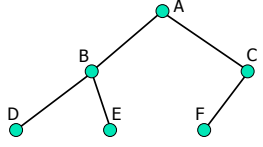
```

Procédure ParcoursPrefixe(Nœud *N)
Début
    si N ≠ NULL alors
        écrire(N->valeur);
        ParcoursPrefixe(N->filsG);
        ParcoursPrefixe(N->filsD);
    fin
Fin
                
```



Réversivité (pour les arbres)

■ Parcours préfixe arbre binaire




```

Procédure ParcoursPrefixe(Nœud *N)
Début
    si N ≠ NULL alors
        écrire(N->valeur);
        ParcoursPrefixe(N->filsG);
        ParcoursPrefixe(N->filsD);
    fin
Fin
                
```

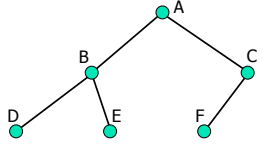
```

Procédure ParcoursPrefixe(N=A)
Début
    si N ≠ NULL alors
        écrire(N->valeur);
        ParcoursPrefixe(B);
        ParcoursPrefixe(C);
    fin
Fin
                
```



Réversivité (pour les arbres)

■ Parcours préfixe arbre binaire



```

Procédure ParcoursPrefixe(Nœud *N)
Début
    si N ≠ NULL alors
        écrire(N->valeur);
        ParcoursPrefixe(N->filsG);
        ParcoursPrefixe(N->filsD);
    fin
Fin
                
```

```

Procédure ParcoursPrefixe(N=A)
Début
    si N ≠ NULL alors
        écrire(N->valeur);
        ParcoursPrefixe(B);
        ParcoursPrefixe(C);
    fin
Fin
                
```

```

Procédure ParcoursPrefixe(N=B)
Début
    si N ≠ NULL alors
        écrire(N->valeur);
        ParcoursPrefixe(D);
        ParcoursPrefixe(E);
    fin
Fin
                
```

28

Récursivité (pour les arbres)

■ Parcours préfixe arbre binaire

Procédure **ParcoursPrefixe**(Noeud *N)
 Début
 si N ≠ NULL alors
 écrire(N->valeur);
 ParcoursPrefixe(N->filsG);
 ParcoursPrefixe(N->filsD);
 fin
 Fin

Procédure **ParcoursPrefixe(N=A)**
 Début
 si N ≠ NULL alors
 écrire(N->valeur);
 ParcoursPrefixe(B);
 ParcoursPrefixe(C);
 fin
 Fin

Procédure **ParcoursPrefixe(N=B)**
 Début
 si N ≠ NULL alors
 écrire(N->valeur);
 ParcoursPrefixe(D);
 ParcoursPrefixe(E);
 fin
 Fin

Procédure **ParcoursPrefixe(N=C)**
 Début
 si N ≠ NULL alors
 écrire(N->valeur);
 ParcoursPrefixe(NULL);
 ParcoursPrefixe(NULL);
 fin
 Fin

29

Récursivité (pour les arbres)

■ Parcours préfixe arbre binaire

Procédure **ParcoursPrefixe**(Noeud *N)
 Début
 si N ≠ NULL alors
 écrire(N->valeur);
 ParcoursPrefixe(N->filsG);
 ParcoursPrefixe(N->filsD);
 fin
 Fin

Procédure **ParcoursPrefixe(N=A)**
 Début
 si N ≠ NULL alors
 écrire(N->valeur);
 ParcoursPrefixe(B);
 ParcoursPrefixe(C);
 fin
 Fin

Procédure **ParcoursPrefixe(N=B)**
 Début
 si N ≠ NULL alors
 écrire(N->valeur);
 ParcoursPrefixe(D);
 ParcoursPrefixe(E);
 fin
 Fin

30

Récursivité (pour les arbres)

■ Parcours préfixe arbre binaire

Procédure **ParcoursPrefixe**(Noeud *N)
 Début
 si N ≠ NULL alors
 écrire(N->valeur);
 ParcoursPrefixe(N->filsG);
 ParcoursPrefixe(N->filsD);
 fin
 Fin

Procédure **ParcoursPrefixe(N=A)**
 Début
 si N ≠ NULL alors
 écrire(N->valeur);
 ParcoursPrefixe(B);
 ParcoursPrefixe(C);
 fin
 Fin

Procédure **ParcoursPrefixe(N=B)**
 Début
 si N ≠ NULL alors
 écrire(N->valeur);
 ParcoursPrefixe(D);
 ParcoursPrefixe(E);
 fin
 Fin

Procédure **ParcoursPrefixe(N=C)**
 Début
 si N ≠ NULL alors
 écrire(N->valeur);
 ParcoursPrefixe(NULL);
 ParcoursPrefixe(NULL);
 fin
 Fin

31

Récursivité (pour les arbres)

■ Parcours préfixe arbre binaire

```

graph TD
    A((A)) --- B((B))
    A --- C((C))
    B --- D((D))
    B --- E((E))
    C --- F((F))
        
```

Procédure **ParcoursPrefixe**(Noeud *N)

Début

si N ≠ NULL alors

écrire(N->valeur);

ParcoursPrefixe(N->filsG);

ParcoursPrefixe(N->filsD);

fin

Fin

Procédure **ParcoursPrefixe**(N=A)

Début

si N ≠ NULL alors

écrire(N->valeur);

ParcoursPrefixe(B);

ParcoursPrefixe(C);

fin

Fin

32

Récursivité (pour les arbres)

■ Parcours préfixe arbre binaire

```

graph TD
    A((A)) --- B((B))
    A --- C((C))
    B --- D((D))
    B --- E((E))
    C --- F((F))
        
```

Procédure **ParcoursPrefixe**(Noeud *N)

Début

si N ≠ NULL alors

écrire(N->valeur);

ParcoursPrefixe(N->filsG);

ParcoursPrefixe(N->filsD);

fin

Fin

Procédure **ParcoursPrefixe**(N=A)

Début

si N ≠ NULL alors

écrire(N->valeur);

ParcoursPrefixe(B);

ParcoursPrefixe(C);

fin

Fin

Procédure **ParcoursPrefixe**(N=C)

Début

si N ≠ NULL alors

écrire(N->valeur);

ParcoursPrefixe(F);

ParcoursPrefixe(NULL);

fin

Fin

33

Récursivité (pour les arbres)

■ Parcours préfixe arbre binaire

```

graph TD
    A((A)) --- B((B))
    A --- C((C))
    B --- D((D))
    B --- E((E))
    C --- F((F))
        
```

Procédure **ParcoursPrefixe**(Noeud *N)

Début

si N ≠ NULL alors

écrire(N->valeur);

ParcoursPrefixe(N->filsG);

ParcoursPrefixe(N->filsD);

fin

Fin

Procédure **ParcoursPrefixe**(N=A)

Début

si N ≠ NULL alors

écrire(N->valeur);

ParcoursPrefixe(B);

ParcoursPrefixe(C);

fin

Fin

Procédure **ParcoursPrefixe**(N=C)

Début

si N ≠ NULL alors

écrire(N->valeur);

ParcoursPrefixe(F);

ParcoursPrefixe(NULL);

fin

Fin

Procédure **ParcoursPrefixe**(N=F)

Début

si N ≠ NULL alors

écrire(N->valeur);

ParcoursPrefixe(NULL);

ParcoursPrefixe(NULL);

fin

Fin

Réversivité (pour les arbres)

■ Parcours préfixe arbre binaire

```

graph TD
    A((A)) --- B((B))
    A --- C((C))
    B --- D((D))
    B --- E((E))
    C --- F((F))
    
```

Procédure **ParcoursPrefixe**(Noeud *N)

Début

si N ≠ NULL alors

 écrire(N->valeur);

 ParcoursPrefixe(N->filsG);

 ParcoursPrefixe(N->filsD);

fin

Fin

Procédure **ParcoursPrefixe**(N=A)

Début

si N ≠ NULL alors

 écrire(N->valeur);

 ParcoursPrefixe(B);

 ParcoursPrefixe(C);

fin

Fin

Procédure **ParcoursPrefixe**(N=C)

Début

si N ≠ NULL alors

 écrire(N->valeur);

 ParcoursPrefixe(F);

 ParcoursPrefixe(NULL);

fin

Fin

Réversivité (pour les arbres)

■ Parcours préfixe arbre binaire

```

graph TD
    A((A)) --- B((B))
    A --- C((C))
    B --- D((D))
    B --- E((E))
    C --- F((F))
    
```

Procédure **ParcoursPrefixe**(Noeud *N)

Début

si N ≠ NULL alors

 écrire(N->valeur);

 ParcoursPrefixe(N->filsG);

 ParcoursPrefixe(N->filsD);

fin

Fin

Réversivité (pour les arbres)

■ Parcours infixe arbre binaire

```

graph TD
    A((A)) --- B((B))
    A --- C((C))
    B --- D((D))
    B --- E((E))
    C --- F((F))
    
```

Procédure **ParcoursInfixe**(Noeud *N)

Début

si N ≠ NULL alors


 ParcoursInfixe(N->filsG);

 écrire(N->valeur);

 ParcoursInfixe(N->filsD);

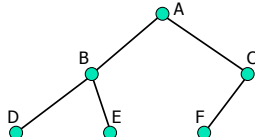
fin

Fin




Récursivité (pour les arbres)

■ Parcours postfixe arbre binaire




```

Procédure ParcoursPostfixe(Noeud *N)
Début
  si N ≠ NULL alors
    ParcoursPostfixe(N->filsG);
    ParcoursPostfixe(N->filsD);
    ecrire(N->valeur);
fin
Fin
          
```



Exercices sur les arbres

1. Ecrire une fonction qui affiche la hauteur d'un arbre binaire
2. Ecrire un algorithme récursif qui calcule la taille d'un arbre binaire
3. Ecrire une fonction qui cherche une valeur dans un arbre binaire
4. Ecrire une fonction qui calcule le minimum d'un arbre binaire
5. Ecrire une fonction qui insère une feuille dans la branche la plus à gauche d'un arbre binaire.
6. Ecrire une fonction qui supprime la feuille la plus à droite d'un arbre binaire.
7. Ecrire la fonction qui donne le miroir d'un arbre binaire
8. Ecrire une fonction qui supprime une valeur se trouvant dans un nœud interne d'un arbre binaire.




Solutions

1. Ecrire une fonction qui affiche la hauteur d'un arbre binaire

Implémentation en PASCAL

```

// hauteur(a) = hauteur de a
function hauteur(a : TARBRE) : INTEGER;
begin
  if estArbreVide(a) then
    hauteur := -1
  else
    hauteur := 1
              + max(hauteur(gauche(a)),
                    hauteur(droit(a)));
end (hauteur);
          
```



Solutions

40


2. Ecrire un algorithme récursif qui calcule la taille d'un arbre binaire

Implémentation en PASCAL

```

// taille(a) = nbre de noeuds dans a
function taille(a : T_ARBRE) : CARDINAL;
begin
  if estArbreVide(a) then
    taille := 0
  else
    taille := 1
      + taille(gauche(a))
      + taille(droit(a));
  end (taille);

```



Solutions

41


3. Ecrire une fonction qui cherche une valeur dans un arbre binaire

Implémentation en PASCAL

```

// recherche(e,a) = VRAI si e ∈ a
// FAUX sinon
function recherche(e : T_ELEMENT;
  a : T_ARBRE) : BOOLEAN;
begin
  if estArbreVide(a) then
    recherche := false
  else if racine(a)=e then
    recherche := true
  else
    recherche := recherche(e,gauche(a))
      or recherche(e,droit(a));
  end (recherche);


```



Exercices sur les arbres

42

- Ecrire une fonction qui calcule le minimum d'un arbre binaire
- Def Minimum (A)
- Precond : Max =99999
- If A==None: Return Max
- Else return (min(Minimum (A.fg), Minimum (A.fd), A.valeur)




Exercices sur les arbres

43

Ecrire une fonction qui supprime la feuille la plus à droite dans un arbre binaire

```
1. Def supp-feuille (A)
2.   If A == none :
3.     return -1
4.   If A==none :
5.     free (A);
6.     return 1
7.   Else if A.fg != none :
8.     supp-feuille (A.fg)
9.   else if A.fg != none :
10.    supp-feuille (A.fg)
```



Exercices sur les arbres

44

Ecrire la fonction qui donne le miroir d'un arbre binaire

```
1. Def miroir (A)
2.   If A != none :
3.     tmp = a.fg
4.     A.fg=A.fg
5.     A.fg=tmp
6.     miroir (A.fg)
7.     miroir (A.fg)
```
