IF5

Algorithmique et Structures de Données

Akkouche Samir :
E-mail : samir.akkouche@liris.cnrs.fr

Pronost Nicolas:
E-mail : nicolas.pronost@univ-lyon1.fr

Plan du cours

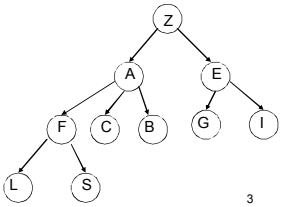
1. Rappels
2. Tableaux dynamiques
3. Listes chainées
4. Piles et File
5. Arbres
   1. Définitions et exemple
   2. Arbres binaires
   3. Tas binaire
   4. Arbres binaires de recherche

Définition d'un arbre n-aire

Un arbre est une structure de données hiérarchique, composée de nœuds et de relations de précédence entre ces nœuds.

Chaque nœud possède :
- **0,1,2…,n successeurs**
- **un et un seul prédécesseur (sauf la racine qui en a 0)**

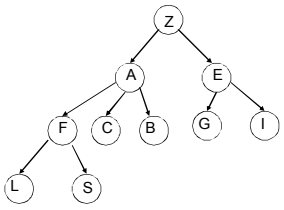
Un nœud ne peut pas être à la fois prédécesseur et successeur d'un autre nœud.



Vocabulaire

- Racine(Arbre) : {Z}
- fils(A) : {F , C , B}

- père(F) : {A}
- degré(A) = 3



Le type Arbre

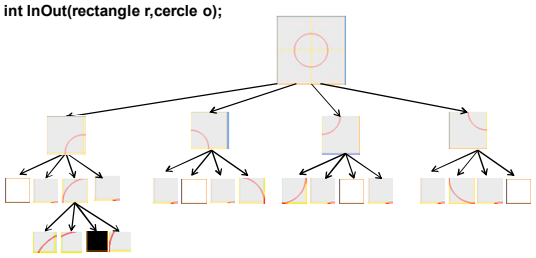
5

Exemple : Arbre quaternaire

```
Cellule = Structure
    info : Element
    tab : tableau[1..4] de pointeurs sur Cellule
Fin Structure
```

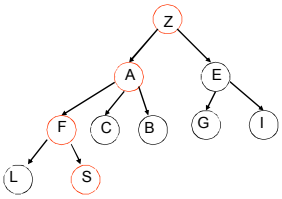
```
ArbreQuaternaire = Structure
    adRacine : pointeur sur Cellule
Fin Structure
```

```
int InOut(rectangle r,cercle o);
```



6

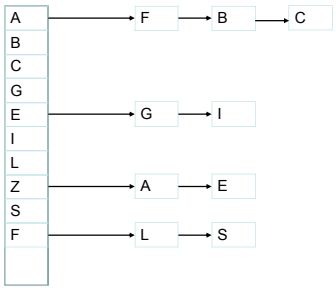
- Branche de l'arbre :
  Ensemble de nœuds successifs allant de la racine à une feuille



- Profondeur d'un nœud x
  p(x) = 0 si x est la racine
  p(x) = p(y) +1 si y est le père de x
- Profondeur d'un arbre
  profondeur maximale des noeuds

7

Représentation



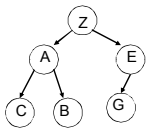
8

Représentation

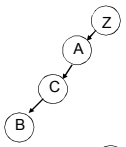
	A	B	C	G	E	I	L	Z	S	F
A	0	1	1	0	0	0	0	0	0	1
B	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0
E	0	0	0	1	0	1	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0
Z	1	0	0	0	1	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	1	0	1	0

9

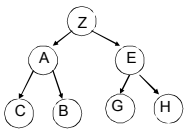
- Un nœud a au plus deux fils
  - ➡ Sous arbre gauche(sag)
  - ➡ Sous arbre droit(sad)



- Degré max = 2



- Arbre dégénéré
- Arbre équilibré



10

TDA Arbre binaire

module ArbreBinaire

importer module Element

exporter type ArbreBin

procedure initialiser(a : ArbreBin)
Précondition a non initialisé
Postcondition a est initialisé à vide
Paramètre en mode donnée-resultat: a

procedure initialiser(a : ArbreBin, b :ArbreBin)
Précondition a non initialisé
Postcondition a est initialisé avec les valeurs de b
Paramètre en mode donnée : b
Paramètre en mode résultat : a

Suite ……

Finmodule

11

TDA Arbre binaire

module ArbreBinaire

importer module Element

exporter type ArbreBin

fonction racine(a : ArbreBin) : *Element*
Précondition a initialisé et non vide
Résultat renvoie l'élément stocké à la racine
Paramètre en mode donnée : a

fonction estVide(a : ArbreBin) : *Booleen*
Précondition a bien initialisé
Postcondition renvoie vrai si a est vide
Paramètre en mode donnée : a

Suite ……

Finmodule

12

TDA Arbre binaire(suite)

procedure testament(a : ArbreBin)
Précondition a bien initialisé
Postcondition L'espace réservé par a est libéré.
Paramètre en mode donnée-résultat : a

procedure afficher(a : ArbreBin)
Précondition a bien initialisé
Postcondition : il faut définir comment afficher
Paramètre en mode donnée : a

implantation

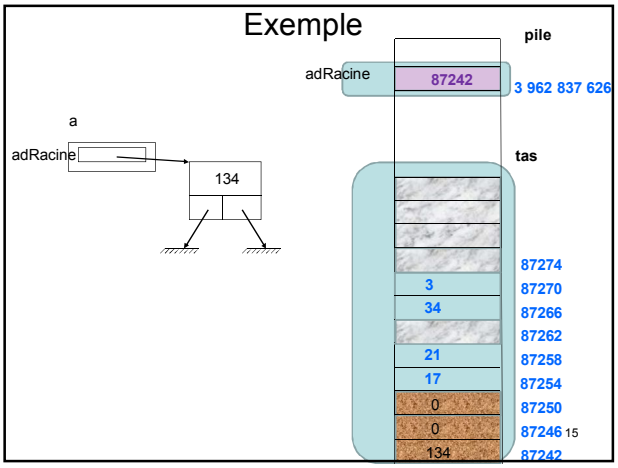
Finmodule

Structure de données

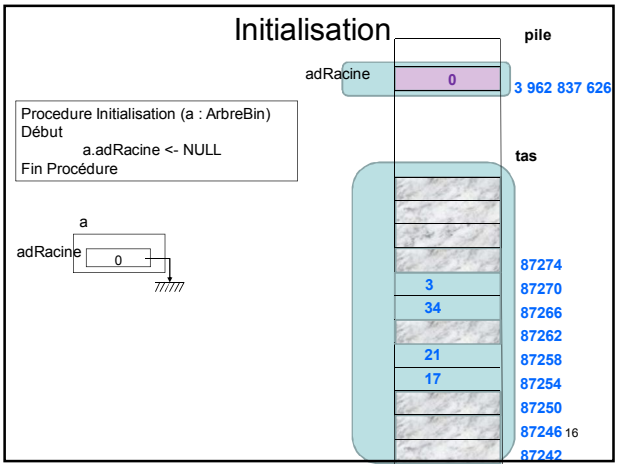
```
Cellule = Structure
                info : Element
                fg,fd : pointeur sur Cellule
          Fin Structure
```

```
ArbreBin = Structure
                adRacine : pointeur sur Cellule
           Fin Structure
```


Exemple



Initialisation



Initialisation (suite)

```
Procedure Initialisation (a : ArbreBin, b : ArbreBin)
  Début   Donnée Résultat : a Donnée :b
  a.adRacine <-NULL
  InitialisationApartirdeCellule(a.adRacine, b.adRacine)
Fin Procédure
```

```
Procedure  InitialisationApartirdeCellule
(p1 : Pointeur sur Cellule, p2 : Pointeur sur cellule)
      Donnée Résultat :p1
      Donnée :P2
       Si(p2 <> NULL) alors
            p1 <- reserve Cellule, p1 -> info <-p2 -> Info,
            p1 ->fg<-NULL, p1 -> fd<-NULL
            InitialisationApartirdeCellule(p1 -> fg, p2 -> fg)
            InitialisationApartirdeCellule(p1 -> fd, p2 -> fd)
         Finsi
Fin Procedure²
```


Initialisation

```
Procedure Initialisation (a : ArbreBin)
Début
          a.adRacine <- NULL
Fin Procédure
```

```
Procedure Initialisation (a : ArbreBin, b : ArbreBin)
  Début   Donnée Résultat :a Donnée : b
  a.adRacine <-NULL
  InitialisationApartirdeCellule(a.adRacine, b.adRacine)
Fin Procédure
```

```
Procedure  InitialisationApartirdeCellule
(p1 : Pointeur sur Cellule, p2 : Pointeur sur cellule)
      Donnée Résultat :p1
      Donnée :P2
       Si(p2 <> NULL) alors
            p1 <- reserve Cellule, p1 -> info <- p2 -> Info,
            p1 -> fg<-NULL, p1 -> fd<-NULL
            InitialisationApartirdeCellule(p1 -> fg, p2 -> fg)
            InitialisationApartirdeCellule(p1 -> fd, p2 -> fd)
         Finsi
Fin Procedure
```


Les différents types de parcours

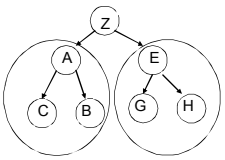
- Parcours en profondeur d'abord
  - Parcours en ordre (infixé) sag,Racine,sad
  - Parcours en pré-ordre (préfixé) Racine,sag,sad
  - Parcours en post-ordre (postfixé) sag,sad,Racine

- Parcours en largeur : Parcours niveau après niveau

Les différents types de parcours(suite)

- Exemple

Parcours en ordre (infixé)
C,A,B,Z,G,E,H
Parcours en pré-ordre (préfixé)
Z,A,C,B,E,G,H
Parcours en post-ordre (postfixé)
C,B,A,G,H,E,Z

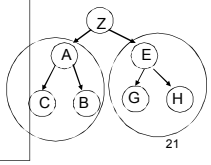


Les différents types de parcours (suite)

Méthode récursive

```
procédure afficheArbre(a : ArbreBin)
   précondition a est initialisé
   postcondition affiche l'arbre en mode infixé
  afficheApartirdeCellule (a.adRacine)
fin afficher
```

```
procédure  afficheApartirdeCellule (p : pointeur sur Cellule)
Début
 si (p <>NULL) alors
    afficheApartirdeCellule(p -> fg)
     Affiche(p)
    afficheApartirdeCellule(p -> fd)
 finsi
fin Procédure
```

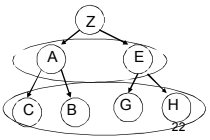


Les différents types de parcours (suite)

Exercice : Ecrire l'algorithme itératif du parcours infixé

Parcours en largeur
Z,A,E,C,B,G,H

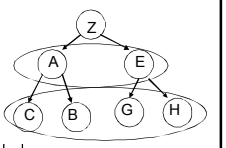
Exercice : Ecrire l'algorithme de parcours en Largeur

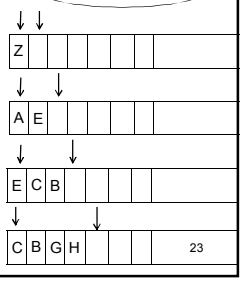


Parcours en largeur dans un arbre

- Utilisation d'une file d'attente

```
Début
    f : File, p pointeur sur Cellule
    initialiser(f)
    enfiler(f, a.adRacine)
    tant que non estvide(f)
      p<- premier(f)
      defiler(f)
      si(p -> fg <> NULL) alors enfiler(p -> fg ) finsi
      si(p -> fd<> NULL) alors enfiler(p -> fd) finsi
       Afficher( p -> info)
    fin tant que
    testament(f)
Fin
```





Plan du cours

1. Rappels
2. Tableaux dynamiques
3. Listes chainées
4. Piles et Files
5. Arbres
   1. Définitions et exemple
   2. Arbres binaires
   3. Tas binaire
   4. Arbres binaires de recherche

Un Arbre Binaire de recherche est une structure permettant de ranger des informations ordonnées

➡ « SAG » < Racine < « SAD »

Les opérations de base sont :

- ➡ Inserer : insère un élément dans l'arbre
- ➡ Existe : cherche si un élément est dans l'arbre
- ➡ Supprimer : retire un nœud de l'arbre

25

Arbres Binaires: structure de donnée

Rappel

```
Cellule = Structure
            info : Element
            fg,fd : pointeur sur Cellule
          Fin Structure
```

```
ArbreBin = Structure
             adRacine : pointeur sur Cellule
           Fin Structure
```

26

Insertion : Méthode récursive

```
Procedure insertion(ab : ArbreBin, e : Element)
  Précondition ab bien initialisée
  Postcondition après insertion, on a toujours sag < Racine < sad
  Donnée-résultat ab
  Donnée e
 insertionApartirdeCellule( ab.adRacine, e )
Fin Procedure
```

27

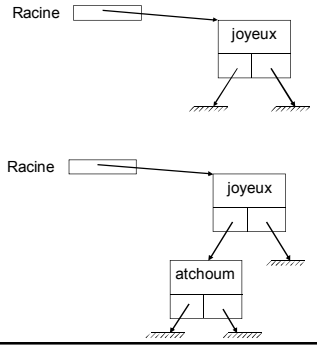
Insertion : Méthode récursive

```
Procedure insertionApartirdeCellule(a : pointeur sur Cellule, e : Element)
  Précondition a bien initialisée
  Postcondition après insertion, on a toujours sag < Racine < sad
  Donnée-résultat a
  Donnée e
 si (a = NULL)  alors    a ←reseve Cellule, a -> fg ← NULL, a -> fd ← NULL, a -> info ←e
       sinon
            si( a -> info <> e) alors
                        si (a -> info < e) alors
                                    insertionApartirdeCellule  (a -> fd,e)
                        sinon
                                    insertionApartirdeCellule  (a -> fg,e)
                        finsi
            finsi
 finsi
Fin Procedure
```

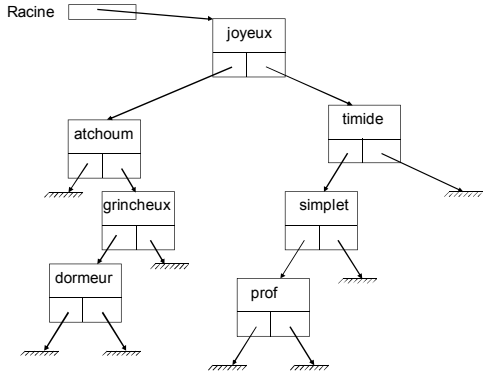
28

On insère toujours dans une feuille de l'arbre.

Exemple : { joyeux , atchoum, grincheux, timide, simplet, prof,dormeur}



29

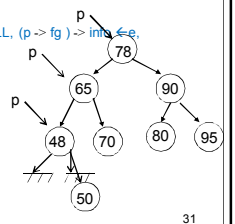


{ joyeux , atchoum, grincheux, timide, simplet, prof,dormeur}

30

Insertion : Méthode itérative

```
Procedure insertionApartirdeCellule( a : pointeur sur Cellule, e : Element)
si (a =NULL) alors
    a ←reseve Cellule, a -> fg ← NULL, a ↑.fd ← NULL, a -> info ←e
sinon
    p : pointeur sur Cellule <- a, Fini : booleen <-Faux
    tant que(Non Fini) faire
      si( p -> info <> e) alors
          si (p -> info > e) alors
            si (p -> fg = NULL) alors
                p -> fg←reserve Cellule,
                (p -> fg ) -> fg← NULL, (p -> fg ) ->fd ← NULL, (p -> fg ) -> info ←e,
                Fini <-Vrai
             sinon p<-p -> fg
            finsi
          Sinon
              idem faire la même chose a droite
          finsi
        sinon Fini <-Vrai
    fintantque
Finsi
Fin Procedure
```



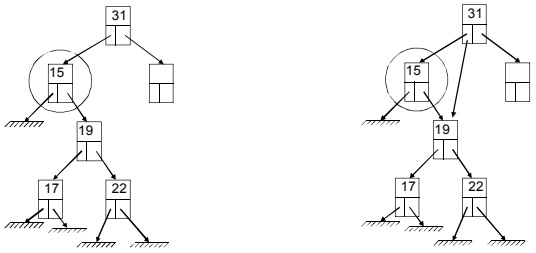
31

Recherche : Méthode récursive

```
fonction rechercheApartirdeCellule( a : pointeur sur Cellule, e : Element) :
                                                        Pointeur sur Cellule
  precondition : a bien initialisée
  résultat : si e existe, renvoie le pointeur sur la cellule, NULL sinon
si(a <> NULL) alors
  si (a -> info = e) alors retourne a
   sinon
     si (e < a -> info) alors recherche (a -> fg,e)
      sinon recherche(a -> fd,e)
     finsi
  finsi
 finsi
 retourne NULL
Fin fonction
```

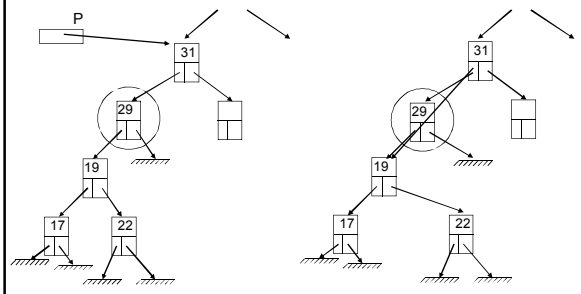
32

- Comment supprimer un nœud ?
  - si le sag de l'élément à supprimer est null



33

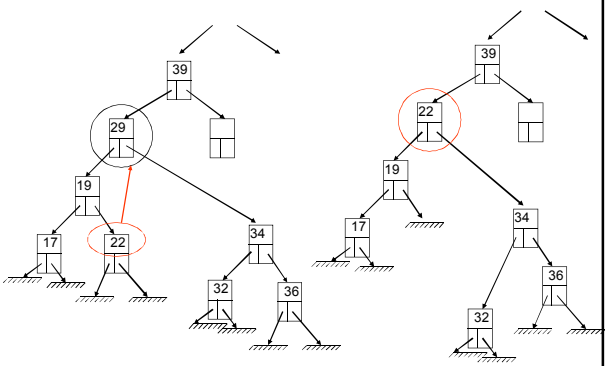
- si le sad de l'élément à supprimer est null



Attention : il faut pointer sur le père de la Cellule à enlever sauf si c'est la racine

34

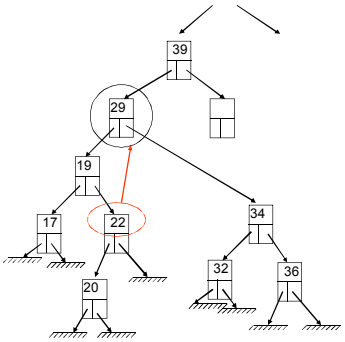
- si aucun n'est null



On remplace par le max à gauche et on supprime la cellule du max

35

- Autre cas pour la position du max.



36