Les arbres (II)

Nour-Eddine Oussous, Éric Wegrzynowski

Licence ST-A, USTL - API2

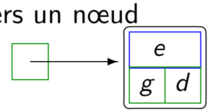
23 novembre 2009

Représentation chaînée

- l'arbre vide est représenté par un pointeur vers rien

$\Delta =$ 

- un arbre non vide $< e; g; d >$ est représenté par un pointeur vers un nœud

$A =$ 

Représentation concrète d'un arbre binaire

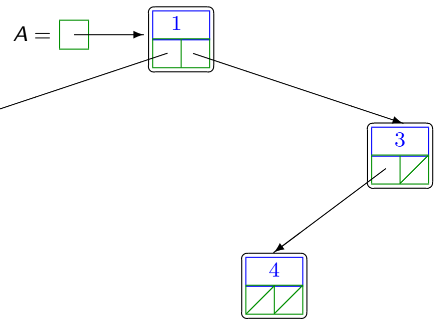


Fig.: Représentation concrète de
$A = <1; <2; \Delta; \Delta>; <3; <4; \Delta; \Delta>; \Delta>>$

Déclaration en Pascal

Définition de types

```
const
   ARBREVIDE = NIL;
type
   ARBRE = ^NOEUD;
   NOEUD = record
      info : ELEMENT;
      gauche, droit : ARBRE;
   end {NOEUD};
```

Le type ELEMENT doit être déclaré par ailleurs.

Constructeur

Créer un arbre

```
// creerArbre(e,g,d) = <e;g;d>
function creerArbre(e : ELEMENT;
                    g,d : ARBRE) : ARBRE;
var
   a1 : ARBRE;
begin
   new(a1);
   a1^.info := e;
   a1^.gauche := g;
   a1^.droit := d;
   creerArbre := a1;
end {creerArbre};
```


Sélecteurs

Racine d'un arbre

```
// racine(a)= élément situé à la racine de a
// CU : a non vide
function racine(a : ARBRE) : ELEMENT;
begin
   racine := a^.info;
end {racine};
```


Sélecteurs

Sous-arbre gauche d'un arbre

```
// gauche(a) = sous-arbre gauche de a
// CU : a non vide
function gauche(a : ARBRE) : ARBRE;
begin
   gauche := a^.gauche;
end {gauche};
```

Sélecteurs

Sous-arbre droit d'un arbre

```
// droit(a) = sous-arbre droit de a
// CU : a non vide
function droit(a : ARBRE) : ARBRE;
begin
   droit := a^.droit;
end {droit};
```


Prédicat

Test de vacuité d'un arbre

```
// estArbreVide(a) ⇔ a est vide
function estArbreVide(a:  ARBRE): BOOLEAN;
begin
   estArbreVide := (a = NIL);
end {estArbreVide};
```


Opérations modificatrices

Modifier la racine d'un arbre

```
// modifierRacine(a,e) modifie
// la valeur de l'élément à la racine de a
// CU : a non vide
procedure modifierRacine(const a : ARBRE;
                         const e : ELEMENT);
begin
   a^.info := e;
end {modifierRacine};
```


Opérations modificatrices

Modifier le sous-arbre gauche d'un arbre

```
// modifierGauche(a,g) modifie
// le sous-arbre gauche de a
// en lui attribuant g
// CU : a non vide
procedure modifierGauche(const a : ARBRE;
                         const g : ARBRE);
begin
   a^.gauche := g;
end {modifierGauche};
```

Opérations modificatrices

Modifier le sous-arbre droit d'un arbre

```
// modifierDroit(a,d) modifie
// le sous-arbre droit de a
// en lui attribuant d
// CU : a non vide
procedure modifierDroit(const a : ARBRE;
                        const d : ARBRE);
begin
   a^.droit := d;
end {modifierDroit};
```


Différents parcours

Contrairement aux listes que l'on parcourt de manière séquentielle, il y a plusieurs parcours possibles d'un arbre :

- le parcours préfixé ;
- le parcours postfixé ;
- et le parcours infixé.

Un exemple d'arbre à parcourir

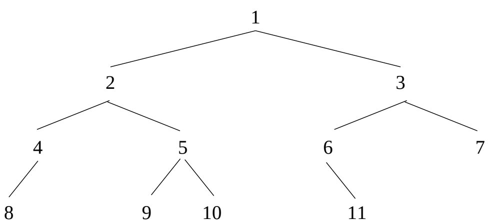


Fig.: Exemple d'arbre à parcourir

Parcours préfixé

Définition

Parcours consistant à

1. traiter la racine
2. parcourir le sous-arbre gauche
3. parcourir le sous-arbre droit

Exemple

Sur l'arbre de la figure 2, dans un parcours préfixé, les nœuds sont traités dans l'ordre

1, 2, 4, 8, 5, 9, 10, 3, 6, 11, 7

Parcours postfixé

Définition

Parcours consistant à

1. parcourir le sous-arbre gauche
2. parcourir le sous-arbre droit
3. traiter la racine

Exemple

Sur l'arbre de la figure 2, dans un parcours postfixé, les nœuds sont traités dans l'ordre

$$8,\ 4,\ 9,\ 10,\ 5,\ 2,\ 11,\ 6,\ 7,\ 3,\ 1$$

Parcours infixé

Définition

Parcours consistant à

1. parcourir le sous-arbre gauche
2. traiter la racine
3. parcourir le sous-arbre droit

Exemple

Sur l'arbre de la figure 2, dans un parcours infixé, les nœuds sont traités dans l'ordre

$$8,\ 4,\ 2,\ 9,\ 5,\ 10,\ 1,\ 6,\ 11,\ 3,\ 7$$

Spécification

Spécification

$$\begin{array}{rrcl} \texttt{taille} : & AB(E) & \longrightarrow & \mathbb{N} \\ & a & \longmapsto & \text{taille de } a = \text{nbre de nœuds} \end{array}$$

Exemples

$$\begin{array}{rcl} \texttt{taille}(\Delta) & = & 0 \\ \texttt{taille}(<1; <3; \Delta; \Delta>; \Delta>) & = & 2 \end{array}$$

Algorithme

Schéma récursif du calcul de la taille d'un arbre a

1. si $a = \Delta$, alors
$$\texttt{taille}(a) = 0$$
2. si $a = <e; g; d>$, alors
$$\texttt{taille}(a) = 1 + \texttt{taille}(g) + \texttt{taille}(d)$$

Implémentation en Pascal

Taille

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


Coût du calcul de la taille

$c(a) =$ nombre de tests de la condition `estArbreVide(a)`.

$$\begin{aligned} c(\Delta) &= 1 \\ c(<e; g; d>) &= 1 + c(g) + c(d) \end{aligned}$$

Conclusion

Le coût de la fonction `taille` est linéaire en le nombre de nœuds de l'arbre.

Rappel

Définition de la hauteur

La hauteur d'un arbre est la longueur de sa plus longue branche.

- cela impose que l'arbre n'est pas vide
- en conséquence la hauteur de l'arbre vide n'est pas définie par cette définition

Spécification

Spécification

$$\begin{aligned} \texttt{hauteur} : AB(E) &\longrightarrow \mathbb{N} \\ a &\longmapsto \text{hauteur de } a \end{aligned}$$

Exemples

$$\begin{aligned} \texttt{hauteur}(<1; \Delta; \Delta>) &= 0 \\ \texttt{hauteur}(<1; <3; \Delta; \Delta>; \Delta>) &= 1 \end{aligned}$$

Rappel

Convention

Pour faciliter le calcul (récursif) de la hauteur d'un arbre, nous conviendrons que la hauteur de l'arbre vide est égale à -1.

$$\texttt{hauteur}(\Delta) = -1$$

Spécification

Spécification

$$\begin{array}{rrcl} \texttt{hauteur} : & AB(E) & \longrightarrow & \mathbb{N} \cup \{-1\} \\ & a & \longmapsto & \text{hauteur de } a \end{array}$$

Exemples

$$\begin{array}{rcl} \texttt{hauteur}(\Delta) & = & -1 \\ \texttt{hauteur}(<1; \Delta; \Delta>) & = & 0 \\ \texttt{hauteur}(<1; <3; \Delta; \Delta>; \Delta>) & = & 1 \end{array}$$

Algorithme

Schéma récursif du calcul de la hauteur d'un arbre a

1. si $a = \Delta$, alors
$$\texttt{hauteur}(a) = -1$$
2. si $a = <e; g; d>$, alors
$$\texttt{hauteur}(a) = 1 + \max(\texttt{hauteur}(g), \texttt{hauteur}(d))$$

Implémentation en Pascal

Hauteur

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

Coût du calcul de la hauteur

$c(a)$ = nombre de tests de la condition `estArbreVide(a)`.

$$\begin{aligned} c(\Delta) &= 1 \\ c(< e; g; d >) &= 1 + c(g) + c(d) \end{aligned}$$

Conclusion

Le coût de la fonction `hauteur` est linéaire en le nombre de nœuds de l'arbre.