Les arbres (III)

Nour-Eddine Oussous, Éric Wegrzynowski

Licence ST-A, USTL - API2

30 novembre 2009

Plan du cours

Spécification

Spécification

$$\texttt{recherche} : E \times AB(E) \longrightarrow Booleen$$
$$e, a \longmapsto \begin{cases} \text{Vrai} & \text{si } e \in a \\ \text{Faux} & \text{sinon} \end{cases}$$

Exemples

$$\texttt{recherche}(1, <1;<3;\Delta;\Delta>;\Delta>) = \text{Vrai}$$
$$\texttt{recherche}(2, <1;<3;\Delta;\Delta>;\Delta>) = \text{Faux}$$

Implémentation en Pascal

Recherche

```
// recherche(e,a) = VRAI si e ∈ a
//                  FAUX sinon
function recherche(e : ELEMENT;
                   a : 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};
```

Coût de la recherche

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

Conclusion

- meilleur cas : e se trouve à la racine, on fait un seul test
- pire cas : e n'est pas dans l'arbre ou se trouve dans la dernière feuille, $c(a)$ est linéaire en n

Définition

On suppose que l'ensemble E des étiquettes des nœuds des arbres est un ensemble totalement ordonné par une relation notée $\leq$.

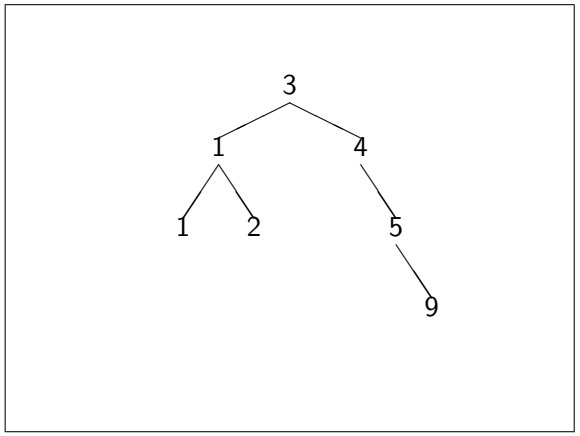
Définition

Un arbre $A \in AB(E)$ est ordonné si

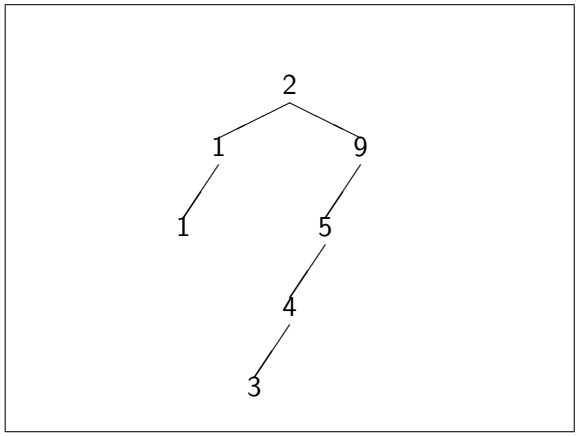
1. il est vide, $A = \Delta$
2. ou bien, il ne l'est pas, $A = < e; g; d >$, et il vérifie les quatre conditions suivantes
   - g est un arbre binaire ordonné ;
   - d est un arbre binaire ordonné ;
   - e est inférieur ou égal à toutes les étiquettes de d ;
   - e est supérieur ou égal à toutes les étiquettes de g ;

L'ensemble des arbres binaires ordonnés est noté $ABO(E)$.

Exemples



Exemples



Propriété caractéristique

Théorème

Soit $A \in AB(E)$ un arbre binaire.
A est un arbre binaire ordonné si et seulement si la liste des étiquettes des nœuds dans l'ordre infixé est une liste triée dans l'ordre croissant.

Spécification

Spécification

$$\texttt{rechercheABO} : E \times ABO(E) \longrightarrow Booleen$$
$$e, a \longmapsto \begin{cases} \text{Vrai} & \text{si } e \in a \\ \text{Faux} & \text{sinon} \end{cases}$$

Exemples

$$\texttt{rechercheABO}(1, \langle 3;<1;<1;\Delta;\Delta>;<2;\Delta;\Delta>> <4;\Delta;<5;\Delta;<9;\Delta;\Delta>>>\rangle) = \text{Vrai}$$
$$\texttt{rechercheABO}(6, \langle 3;<1;<1;\Delta;\Delta>;<2;\Delta;\Delta>> <4;\Delta;<5;\Delta;<9;\Delta;\Delta>>>\rangle) = \text{Faux}$$

Implémentation en Pascal

Recherche dans ABO

```
// rechercheABO(e,a) = VRAI si e ∈ a
//                     FAUX sinon
function rechercheABO(e : ELEMENT;
                      a : ARBRE) : BOOLEAN;
begin
  if estArbreVide(a) then
     rechercheABO := false
  else if racine(a) = e then
     rechercheABO := true
  else if racine(a) > e then
     rechercheABO := rechercheABO(e,gauche(a))
  else rechercheABO := rechercheABO(e,droit(a));
end {rechercheABO};
```


Coût de la recherche

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

- meilleur cas : l'élément se trouve à la racine, un seul test
- pire cas : ce nombre dépend de la hauteur de l'arbre
  - si l'arbre est bien équilibré, pour un ABO de n nœuds, ce nombre sera majoré par $\log_2(n)$
  - si l'arbre est un peigne, ce nombre sera proportionnel à n

Conclusion

Le coût de la fonction `rechercheABO` dépend de la forme de l'arbre.

Spécification

Spécification

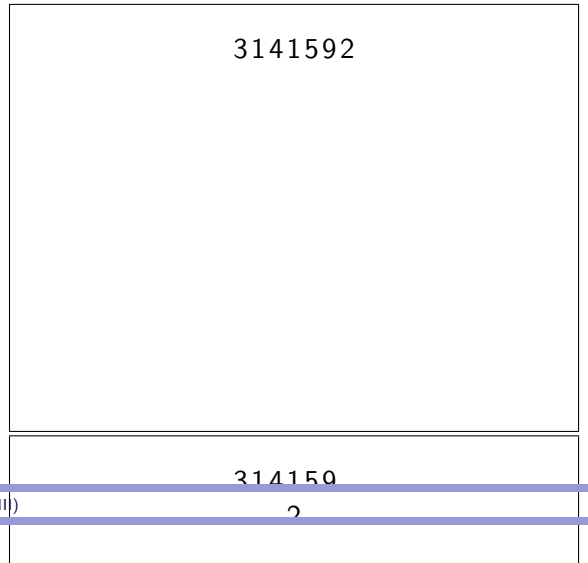
$$\texttt{insererABO} : \begin{array}{ccc} E \times ABO(E) & \longrightarrow & ABO(E) \\ e, a & \longmapsto & a' \end{array}$$

a' = arbre a modifié par l'ajout en feuille du nouvel élément e.

Exemples

$$\texttt{insererABO}(4, <3;<1;\Delta;\Delta>;\Delta>) = <3;<1;\Delta;\Delta>;<4;\Delta;\Delta>>$$
$$\texttt{insererABO}(1, <3;<1;\Delta;\Delta>;<4;\Delta;\Delta>>) = <3;<1;<1;\Delta;\Delta>;\Delta>;<4;\Delta;$$

Exemples

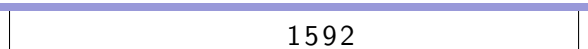


Exemples

Implémentation en Pascal

Insertion dans ABO

```
// insererABO(e,a)
// ajoute un nouvel élément e en feuille de
// l'arbre a. L'arbre a est modifié
procedure insererABO(e : ELEMENT;
                     var a : ARBRE) ;
var
   b: ARBRE ;
begin
  if estArbreVide(a) then
     a := creerArbre(e, ARBREVIDE, ARBREVIDE)
  else if racine(a) <= e then begin
     b := droit(a) ;
     insererABO(e, b) ;
     modifierDroit(a, b) ;
  end else begin
     b := gauche(a) ;
     insererABO(e, b) ;
     modifierGauche(a, b) ;
  end ;
end {insererABO};
```

Coût de l'insertion

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

- L'insertion se fait en feuille
- Ce nombre dépend de la hauteur de l'arbre
  - si l'arbre est bien équilibré, pour un ABO de n nœuds, ce nombre sera majoré par $\log_2(n)$
  - si l'arbre est un peigne, ce nombre sera proportionnel à n
- Il est donc souhaitable d'équilibrer les arbres au fur et à mesure des opérations d'insertion

Conclusion

Le coût de la fonction `insererABO` dépend de la forme de l'arbre.