ALGORITHME ET STRUCTURE DE DONNÉES

LES ARBRES BINAIRES

Par Joel Sandé

Introduction

- Les arbres binaires sont l'une des fondamentales structures de stockage de données en programmation.

- Pourquoi les utiliser ? Parcequ'ils combinent les avantages de 2 autres structures : les tableaux ordonnés et les lists chainées (recherche rapides, suppressions rapides, ..).

Introduction

- Les insertions et deletions sont rapides à effectuer en listChainées. On change quelque references.

- Malheureusement, trouver un élément dans une liste chainée n'est pas facile; Vous devez commencer par le début de la liste et visiter tous les éléments jusqu'à tomber sur l'élément désiré. Il faut suivre la chaine.

Introduction

- Les arbres viennent resoudre ce problème.
- Ils offrent la possibilité de faire des insertions et délétions rapides dans une liste chainée, et une recherche rapide dans un tableau ordonné.
- Les arbres sont donc l'une des plus interessants structures de données.
- Dans ce cours, laboratoire, nous focusserons sur les arbrea binaires.

Arbre binaire

- Un noeud a tout au plus deux enfants : un enfant de gauche, et un enfant de droite.
- Les enfants de la dernière génération sont des feuilles.
- Définition : l'enfant de gauche d'un Noeud doit avoir une clée de valeur inferieure à celle de son parent, et l'enfant de droite doit avoir une clé supérieure à celle de son parent.

Arbre Binaire

- Traversée d'un Arbre : Il y a 3 facons de traverser un arbre.

  - Inorder : Traverser les Noeuds en ordre ascendant de la valeur de la clee, se fait tres bien avec la récursivité : La fonction s'appelle pour traverser le sous-arbre de gauche, visite le noeud, et s'appelle à nouveau pour traverser le sous-arbre de droite.
  - Preorder ou Postorder : utiles lorsqu'on écrit un programme qui analyse des expressions algébriques : pensez Infixe - Postfixe

Concentrons nous sur l'appel réccurssif de la méthode inOrder

```
private void inOrder(node localRoot)
   {
   if(localRoot != null)
      {
      inOrder(localRoot.leftChild);

      System.out.print(localRoot.iData + " ");
      inOrder(localRoot.rightChild);
      }
   }
```

Dans les exemples que je vous montrerai au laboratoire, j'ai fait deux fonction dont l'une retourne un vecteur. NB : Il faut que vous soyez capables de modifier une fonction selon vos gouts.

Concentrons nous sur l'appel récurssif de la méthode inOrder

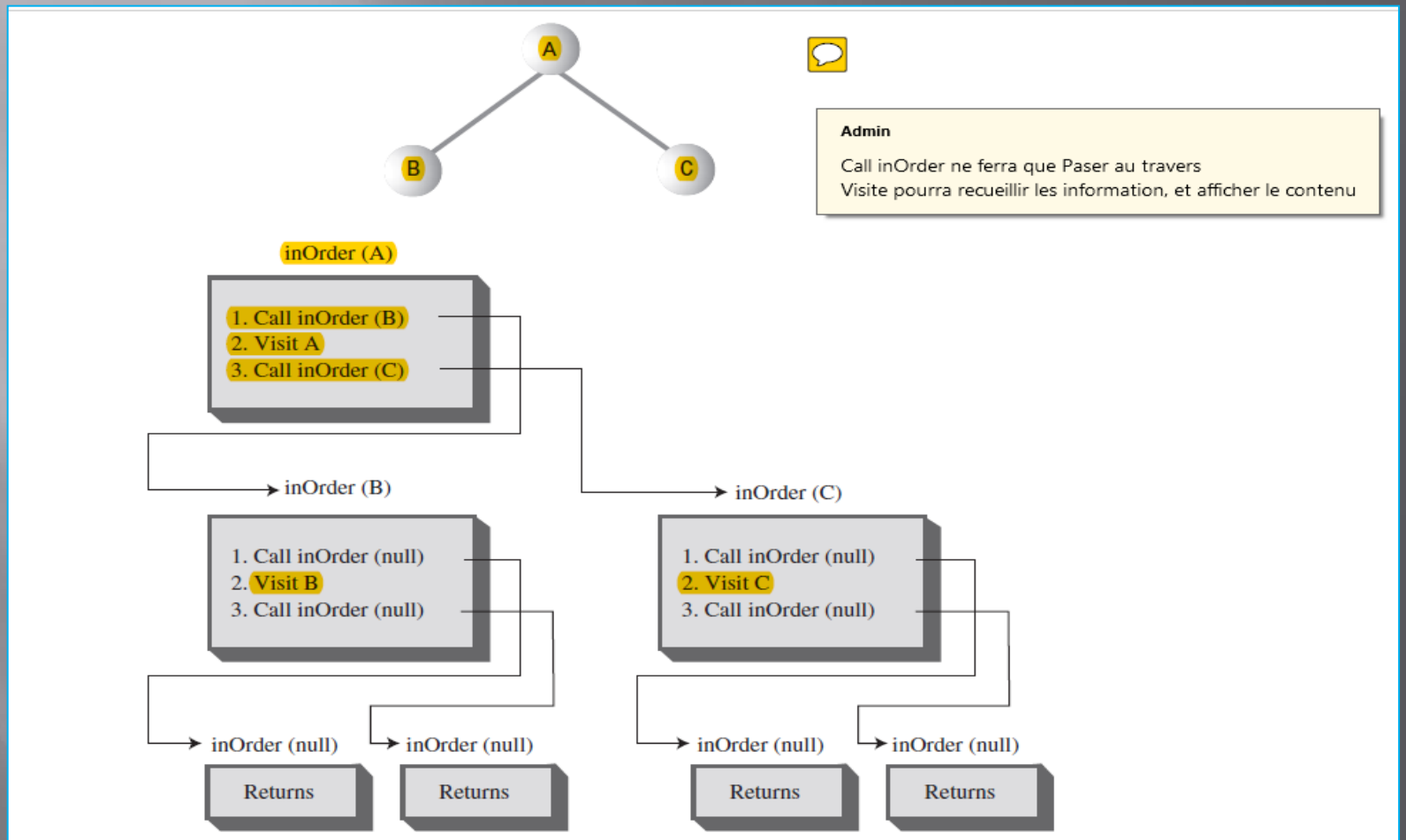

FIGURE 8.9 The `inOrder()` method applied to a three-node tree.

Concentrons nous sur l'appel réccurssif de la méthode inOrder

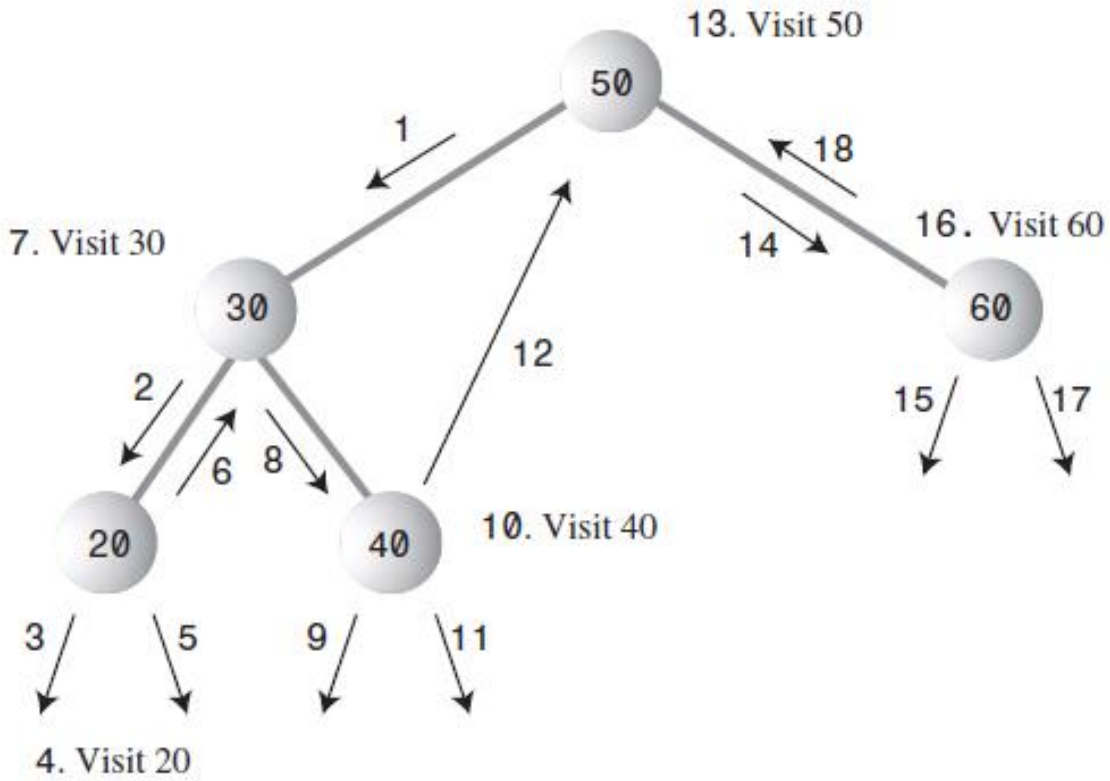

FIGURE 8.10 Traversing a tree inorder.

Valeur minimum d'un arbre

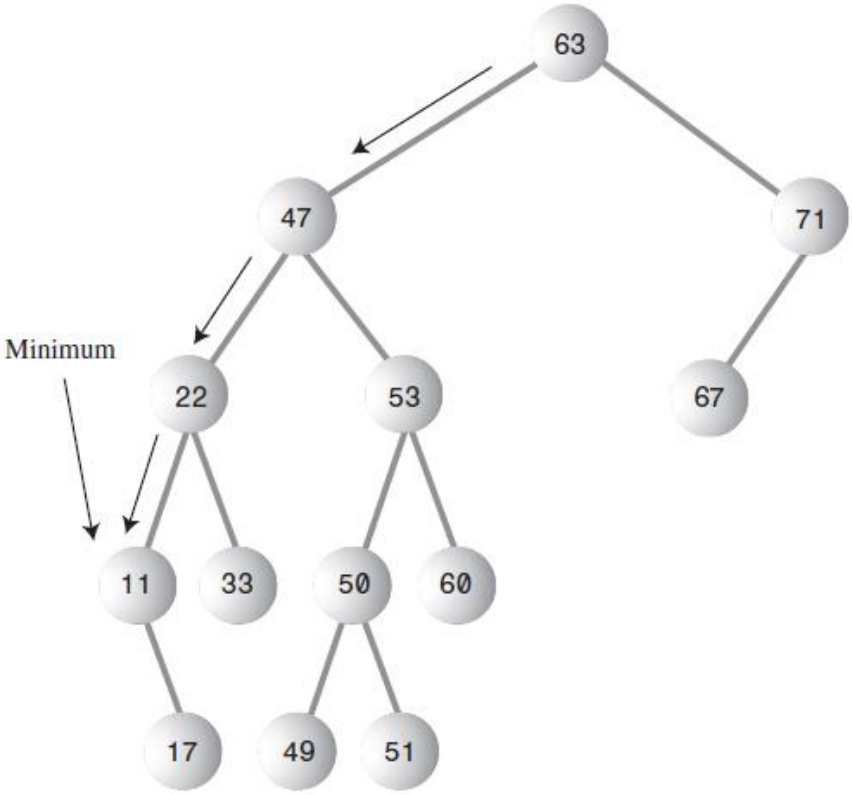

FIGURE 8.12 Minimum value of a tree.

Implicitement on peut déduire la procédure pour trouver la valeur maximale d'un arbre.

Valeur minimum d'un arbre

```
public Node minimum()      // returns node with minimum key value
   {
   Node current, last;
   current = root;                      // start at root
   while(current != null)               // until the bottom,
```

```
   {
   last = current;                      // remember node
   current = current.leftChild;         // go to left child
   }
return last;
}
```

Supprimer un Noeud

- Supprimer un probablement l'action la plus plus difficile à oppérer dans un arbre.

- Il y a 3 cas de figures :
  - Le noeud à supprimer est une feuille (il n'a pas d'enfants). Facile à faire.
  - Le noeud à supprimer a 1 enfant (modérement facile).
  - Le noeud à supprimer a 2 enfants (compliqué).

Supprimer un noeud feuille. Le Noeud 7

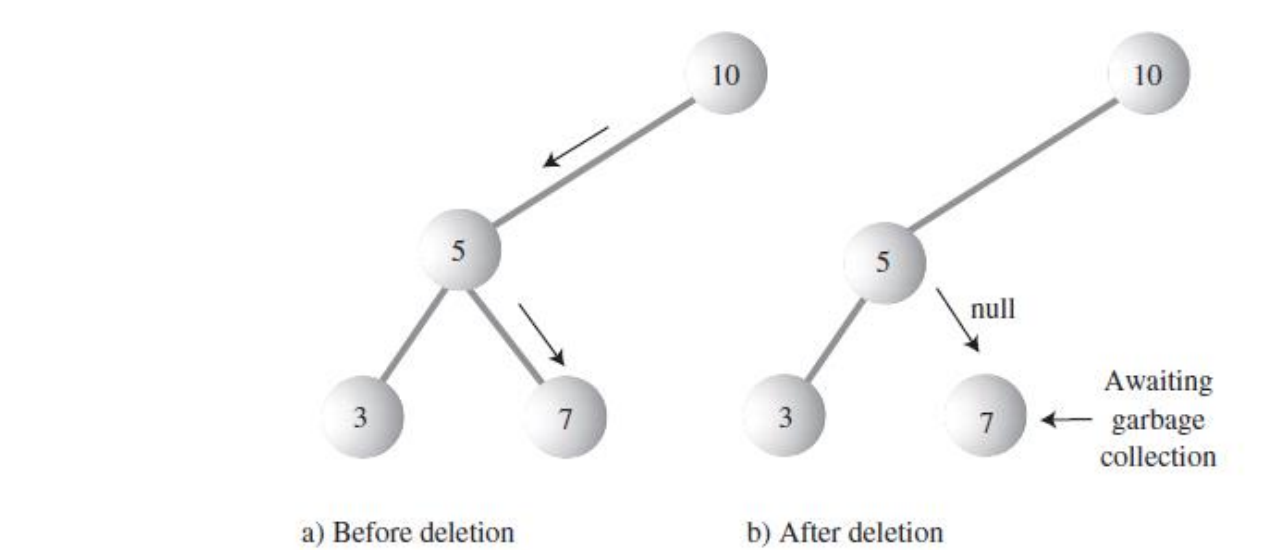

FIGURE 8.13 Deleting a node with no children.

On modifie l'attribut enfant_gauche ou enfant_droite de SON parent par null.

Supprimer un noeud ayant un enfant

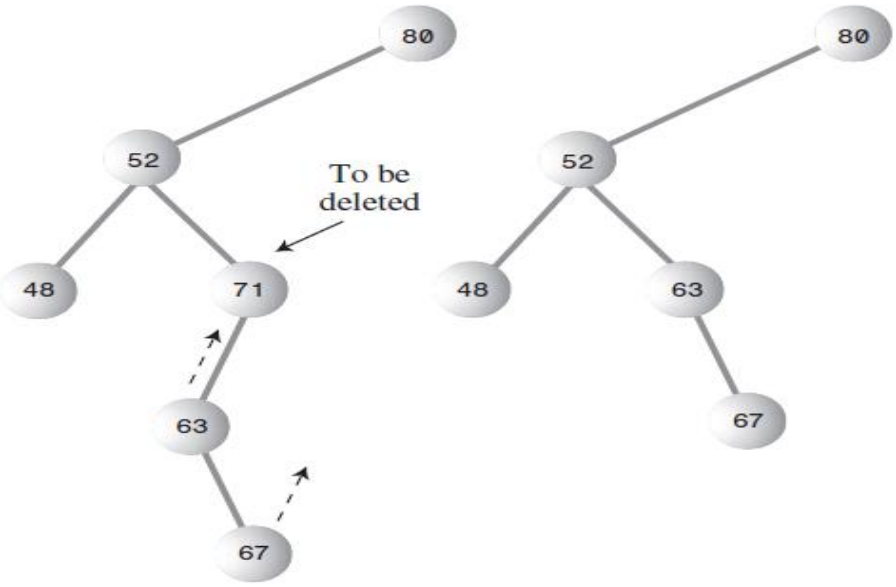

FIGURE 8.14 Deleting a node with one child.

Ce Noeud n'a que 2 connections : une avec son partent, et une autre avec son unique enfant. Nous allons donc connecter SON PARENT directement avec SON UNIQUE enfant.

Supprimer un noeud ayant un enfant

- Ceci demande de changer la référence chez le parent : enfant_gauche ou enfant_droite à pointer sur l'enfant du Noeud supprimé.

Supprimer un noeud ayant 2 enfants

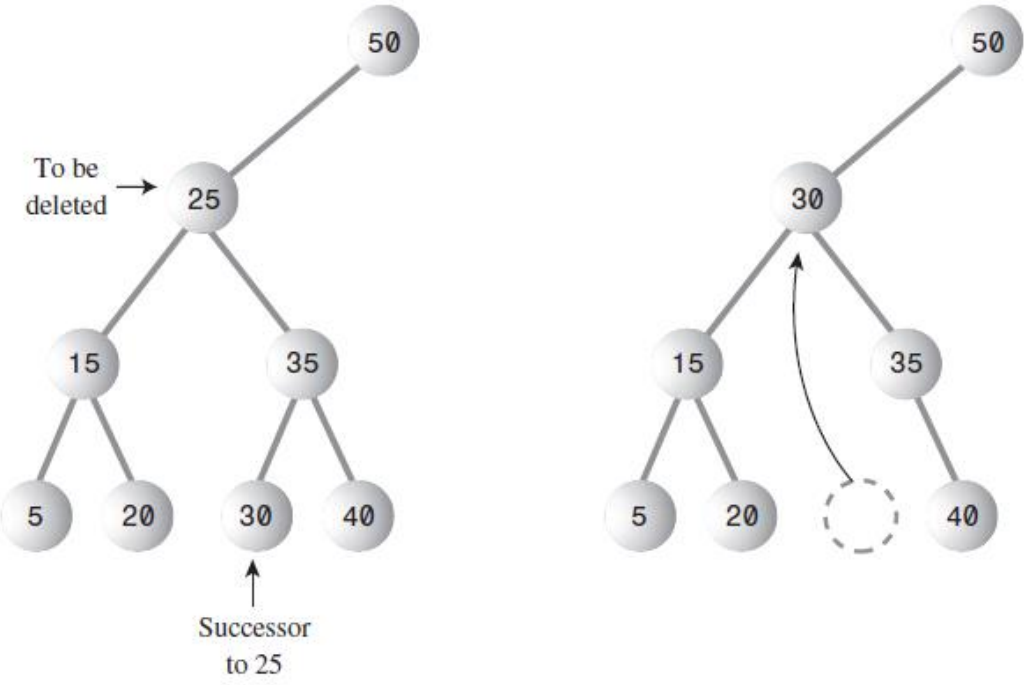

FIGURE 8.16 Node replaced by its successor.

Par quoi je le remplace mon Noeud ? Remarque, si je considère le sous-arbre de droite, c'est le plus petit élément (30) qui peut le remplacer, et si je considère le sous-arbre de gauche, c'est le plus gros élément (20) qui peut le remplacer.

Idées de Gabarit

- Class Noeud <E> {
- int cle; // la clee
- E donnee; // la valeur
- Noeud enfantGauche; // son enfant de gauche
- Noeud enfantDroite; // son enfant de droite
-
- public Trouver() // trouver la valeur d'un Noeud via sa clee
- public Ajouter() // ajouter un Noeud
- public Supprimer() // supprimer un Noeud
- }

Idées de Gabarit

```
Class Arbre_Binaire {
        private Noeud racine;
        public void add (key, value)
        public Noeud find (key)
        public void traverse ()    inOrder
        public int min ()
        public int max ()
//      …..
        public delete (key)
        public tab[E] Afficher_Arbre()
}
```

- Lister les entrées en ordre
- Peut retourner un tableau des éléments triés
- Utiliser la récursivité :
  - S'appeler pour lire son enfant de gauche
  - Visiter le noeud (par exemple afficher sa valeur)
  - S'appeler pour lire son enfant de droite

Implémentations

- Je passerai au travers mon code pour vous faire part des grandes lignes.
- Nous parcourons l'ensemble des methodes requises.
- Je laisserai ensuite le gabarits sur Classroom pour la rescousse des dernieres soumissions.