Chapitre 6

Arbres binaires de recherche

6.1 Introduction

On a étudié le problème de la recherche dans une collection d'éléments ordonnés entre eux : on a montré que

- Pour une liste contiguë, la recherche se fait en $\Theta(log\ n)$ comparaisons ;
- Pour la même représentation l'adjonction ou la suppression peut nécessiter $\Theta(n)$ opérations.

Si la collection n'est pas figée, il paraît nécessaire d'utiliser une représentation chaînée (même si la représentation prend plus de place mémoire). Pour cette représentation :

- L'adjonction d'un élément se fait en un temps constant d'opérations ;
- La suppression demande trop de temps ($\Theta(n)$ comparaisons au pire).

L'utilisation de certaines structures arborescentes permet d'obtenir de meilleures complexités.

Les méthodes de recherche dans une arborescence reposent toutes sur le même principe : la comparaison avec le (ou les) élément(s) d'un noeud de l'arbre, permet d'orienter la suite de la recherche dans l'arbre.

On peut aussi voir un lien avec les arbres de décision : l'arbre de décision de l'algorithme de recherche dichotomique peut être vu comme un cas particulier d'arbre binaire de recherche.

6.2 Définitions

Définition 6.2.1 *Un* **arbre binaire de recherche** *est un arbre binaire étiqueté tel que pour tout nœud x de l'arbre :*

- *Les éléments de tous les nœuds du* **sous-arbre gauche** *de x sont* **inférieurs ou égaux** *à l'élément contenu dans x ;*
- *Les éléments de tous les nœuds du* **sous-arbre droit** *de x sont* **supérieurs** *à l'élément contenu dans x.*

Dans la suite, on travaillera avec un arbre binaire représenté de façon chaînée par la représentation fils gauche, fils droit. Pour un noeud N :

- N.G (resp. N.D) représente la racine du sous-arbre gauche (resp. droit) de N (ou l'arbre vide si ce sous-arbre gauche (resp. droit) est vide).
- N.Valeur représente la valeur contenue dans le nœud N.

Comme on utilise les listes chaînées, on définit le **Type** ABR suivant :

Type ABR = **Vide Ou Enregistrement**
Valeur : <type de base>
G : ABR {sous-arbre gauche}
D : ABR {sous-arbre droit}
Fin Enregistrement

On utilisera la fonction Créer_ABR(), pour créer un ABR indéterminé :

Fonction Créer_ABR(): ABR
{ **Rôle** : elle crée un ABR avec une Valeur et des fils G et D indéterminés
}

6.3 Recherche d'un élément

Une version récursive de la recherche dans un ABR est donnée par :

Fonction Chercher(X, A) : Booléen
{ **Rôle** : la fonction recherche si un élément X appartient à un ABR A : Le résultat est **Vrai** si X appartient, **Faux** sinon }
Paramètres Donnée
A : ABR {on travaille sur l'arbre A}
X : <type de base> {la valeur à rechercher}
Début
Si A = **Vide**
Alors Retourner (**Faux**) {Recherche négative}
Sinon
Si X = A.Valeur
Alors Retourner (**Vrai**) {Recherche positive}
Sinon
Si X < A.Valeur
Alors Retourner (Chercher(X,A.G))
Sinon Retourner (Chercher(X,A.D))
Fin Si
Fin Si
Fin Si
Fin

Remarque. Cette recherche est en $O(n)$ et en $\Omega(1)$ où n est le nombre de noeuds dans l'arbre.

6.4 Adjonction d'un élément

L'adjonction d'un élément dans une liste (ou arbre) se fait selon le principe suivant :

1. Déterminer la place où l'on doit faire l'adjonction ;
2. Réaliser l'adjonction à cette place.

Dans un ABR, l'opération de l'adjonction se décompose en deux étapes :

(1) En une étape de **recherche**, qui renvoie des informations sur la place où doit être inséré le nouvel élément ;

(2) Une deuxième étape de l'**adjonction proprement dite**.

6.4.1 Insertion aux feuilles

Dans cette méthode, on rattache un nouveau nœud à une feuille de l'arbre de recherche. L'ajout est précédé d'une phase de recherche de la place d'insertion pour qu'après insertion la propriété d'arbre binaire de recherche soit respectée.
La procédure récursive est donnée par :

```
Procédure Ajoutefeuille(X, A)
{ Rôle : Adjonction d'un élément X aux feuilles d'un arbre binaire de recherche A }
Paramètre Donnée
    X : <type de base> { la valeur à rechercher}
Paramètre Donnée/Résultat
    A : ABR { on travaille sur l'arbre A à modifier}
Début
    Si A = Vide
    Alors
        A⟵ Créer_ABR()
        A.Valeur ⟵ X
        A.G ⟵ Vide
        A.D ⟵ Vide
    Sinon
            Si X ≤ A.Valeur
            Alors Ajoutefeuille(X,A.G)
            Sinon Ajoutefeuille(X,A.D)
            Fin Si
    Fin Si
Fin
```

6.4.2 Adjonction à la racine

Cette méthode peut être rapprochée des méthodes auto-adaptatives sur les listes.

Pour pouvoir mettre une valeur X à la racine de l'arbre de recherche, il faut que toutes les valeurs précédemment contenues dans l'arbre de recherche soient séparées en deux arbres de recherche : (i) un arbre qui sera le sous-arbre gauche de la racine et qui contiendra des valeurs inférieures ou égales à X, et (ii) l'autre arbre qui sera le sous-arbre droit de la racine et qui contiendra des valeurs strictement supérieures à X.

On considère deux étapes :

1. Une première étape qui consiste à couper l'arbre binaire de recherche en deux sous-arbres binaires de recherche.

2. Une deuxième étape qui consiste à rassembler les nouveaux sous-arbres créés autour de la nouvelle racine X.

◇ Première étape

La procédure de coupure de l'arbre binaire de recherche initial en deux sous-arbres binaires de recherche est donnée par :

Procédure Couper($X,\ A,\ G,\ D$)
{ **Rôle** : coupure de l'ABR A, selon l'élément X, en deux ABR G et D;
G contient tous les éléments de A inférieurs ou égaux à X, et D contient tous les éléments de A strictement supérieurs à X. }
Paramètre Donnée
X : <type de base> {la valeur autour coupe}
Paramètre Donnée/Résultat
A : ABR {ABR à couper}
Paramètres Résultat
$G,\ D$: ABR {deux nouveaux ABRs}
Début
Si A = **Vide**
Alors
$G \longleftarrow$ **Vide**
$D \longleftarrow$ **Vide**
Sinon
Si $X <$ A.Valeur
Alors
$D \longleftarrow A$
Couper($X,\ A.G,\ G,\ D.G$)
Sinon
$G \longleftarrow A$
Couper($X,\ A.D,\ G.D,\ D$)
Fin Si
Fin Si
Fin

◇ Deuxième étape

La procédure de rassemblement des deux ABRs autour de la racine X est donnée par :

Procédure Ajouter_Racine($X,\ A,\ G,\ D$)
{ **Rôle** : la procédure réalise l'adjonction de l'élément X à la racine de l'arbre binaire de recherche A; elle utilise la procédure de coupure d'un arbre binaire de recherche }
Paramètre Donnée
X : <type de base> {la valeur à insérer}
Paramètre Donnée/Résultat
A : ABR {ABR dans lequel on insère X}
Variable
R : ABR {de type ABR}
Début
$R \longleftarrow$ Créer_ABR()
R.Valeur $\longleftarrow X$
Coupure($X,\ A,\ R.G,\ R.D$)
$A \longleftarrow R$
Fin

6.5 Suppression d'un élément

La suppression d'un élément dans une liste (ou arbre) se fait selon le principe suivant :

1. Déterminer la place où l'on doit faire la suppression ;
2. Réaliser la suppression à cette place ;
3. Accompagné éventuellement d'une réorganisation des éléments.

Dans un arbre binaire de recherche, l'opération de la suppression se décompose en deux étapes :

(1) En une étape de **recherche** ;

(2) Une deuxième étape de l'**suppression** qui dépend de la place de l'élément X dans l'arbre binaire de recherche.

Pour cette suppression, plusieurs cas sont possibles :

1. Si le noeud contenant la valeur est sans fils, on peut supprimer le noeud directement.
2. Si le noeud contenant la valeur possède un fils, on peut remplacer le noeud par son fils, et on obtient directement un arbre binaire de recherche.
3. Si le noeud contenant la valeur possède deux fils, plusieurs solutions sont possibles. On choisit ici de remplacer la valeur supprimée par celle qui lui est immédiatement inférieure dans l'arbre de recherche de racine le noeud contenant la valeur à supprimer : cette valeur est la plus grande dans le sous-arbre gauche du noeud contenant la valeur à supprimer.

• Procédure (fonction) de suppression de l'élément Max dans arbre binaire de recherche

Fonction Sup_Max(A) : <type de base>
{ **Rôle** : cette fonction supprime le nœud contenant l'élément maximum dans un ABR A non vide ;
En résultat, elle renvoie l'élément Max de A, qui est privé de ce maximum. }
Paramètre Donnée/Résultat
A : ABR {ABR : on cherche (privé) de so Max}
Variable
Max : <type de base> {Max élément à chercher}
Début
Si $A.D =$ **Vide**
Alors
$Max \longleftarrow A.Valeur$
$A \longleftarrow A.G$
Retourner (Max)
Sinon
Retourner (Sup_Max($A.D$))
Fin

• Procédure de suppression de l'élément X dans un arbre binaire de recherche

Procédure Supprime_ABR(X, A)
{ **Rôle** : cette procédure supprime le nœud contenant l'élément X dans un ABR A ; En résultat, elle donne A si X n'est pas dans A, et sinon A privé de X ; Elle utilise la fonction Sup_Max dans ABR. }
Paramètre Donnée/Résultat
A : ABR {ABR : A privé (ou non) de X}
Paramètre Résultat
X : <type de base> {X élément à supprimer}
Début
Si $A \neq$ **Vide**
Alors
Si $X < A.Valeur$
Alors Supprime_ABR(X, $A.G$)
Sinon
Si $X > A.Valeur$
Alors Supprime_ABR(X, $A.D$)
Sinon {$\underline{A.Valeur = X}$}
Si $A.G =$ **Vide**
Alors $A \longleftarrow A.D$
Sinon
Si $A.D =$ **Vide**
Alors $A \longleftarrow A.G$
Sinon { $\underline{2 \text{ fils}}$}
$A.Valeur \longleftarrow$ Sup_Max($A.G$)
Fin Si
Fin Si
Fin Si
Fin Si
Fin Si
Fin