6. Les ARBRES BINAIRES DE RECHERCHE

6.1. Définitions

Un arbre binaire de recherche est un arbre binaire dans lequel chaque nœud est muni d'une clé prise dans un ensemble totalement ordonné. En outre, pour chaque nœud x de l'arbre, les clés des nœuds du sous arbre gauche de x sont inférieures à la clé de x, et cette clé est elle même inférieure aux clés des nœuds du sous arbre droit de x. Autrement dit, ∀ nœud x de l'arbre, ∀ y ∈ s-a-g(x), ∀ z ∈ s-a-d(x), on a clé(y) < clé(x) < clé(z) **Exemple :**

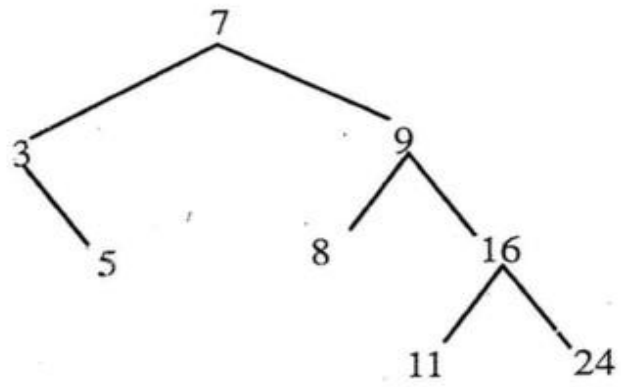


Figure 1. Un exemple d'arbre binaire de recherche.

Remarques :

Un arbre binaire de recherche n'est pas forcément un tas. Par exemple, l'arbre de la figure 1 n'est pas un tas car, d'une part, il n'est pas un arbre parfait et, d'autre part, 7 > 3.

Le **parcours infixe** (s-a-g(x) x s-a-d(x)) de **l'arbre binaire de recherche** donne une liste des clés en ordre croissant. Par exemple, le parcours infixe de l'arbre de la figure 1 produit la liste des clés (3 5 7 8 9 11 16 24) qui est **triée en ordre croissant**.

6.2. Représentation

Un arbre binaire de recherche peut être représenté par un pointeur vers le nœud racine où chaque nœud est un enregistrement qui comprend un champ pour la clé du nœud, suivi

de deux pointeurs, l'un vers le fils gauche (ou sous arbre gauche), l'autre vers le fils droit (ou sous arbre droit).

Par exemple, la figure 2 montre un arbre binaire de recherche A. L'arbre vide (A = nil) est représenté par une case barrée.

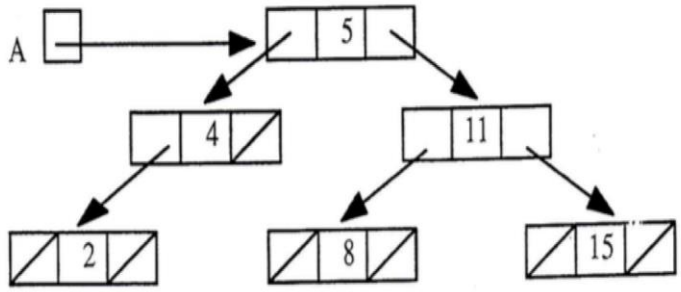


Figure 2. Représentation d'un arbre binaire de recherche à l'aide de pointeurs.

6.3. Recherche d'une clé dans un arbre binaire de recherche

Soient A un arbre binaire de recherche, et x une clé donnée. Le principe général de la recherche dans un arbre binaire de recherche est le suivant.

- On compare la clé cherchée x à la clé de la racine de l'arbre binaire de recherche A, soit à clé(A).

  - Si x= clé(A), l'algorithme est terminé.
  - Si x < clé(A), on cherche x dans le sous arbre gauche de A.
  - Six> clé(A), on cherche x dansk sous arbre droit de A.

L'algorithme récursif qui en résulte est alors le suivant.

Fonction chercher (d x: t; d A: Arbre) : booléen;

{Cette fonction utilise en entrée A et x. A est un arbre binaire de recherche, et x est une clé donnée. La fonction chercher consiste à tester si x appartient ou non à A}

Début

Fin ;

L'algorithme itératif est le suivant.

Fonction chercheriter (d x: t; d A: Arbre) : booléen;

{Cette fonction utilise en entrée A et x. A est un arbre binaire de recherche, et x est une clé donnée. La fonction chercheriter consiste à tester si x appartient ou non à A} var trouve : booléen, var B : Arbre;

Début

Fin ;

6.4. Insertion aux feuilles d'une clé dans un arbre binaire de recherche

Soient A un arbre binaire de recherche, et x une clé donnée. On désire ajouter la clé x à 1'arbre A de sorte que l'arbre résultant soit encore un arbre binaire de recherche. Pour résoudre ce problème, on commence par chercher la place de la clé à insérer x pour ensuite effectuer l'insertion proprement dite de x. Vu la structure de l'arbre binaire de recherche,

pour ajouter x à l'arbre binaire de recherche A, on commence par vérifier si A est un arbre vide. Si tel est le cas, on créé un arbre réduit à un seul nœud de clé x. Sinon, Si A n'est pas l'arbre vide, on compare x à la clé(A). Si x = clé(A), on ne fait rien. Sinon, si x < clé(A), on insère x dans le sous arbre gauche. Sinon, on insère x dans le sous arbre droit.

Exemple :

On souhaite ajouter la clé 6 à l'arbre ABR de la Figure 1.

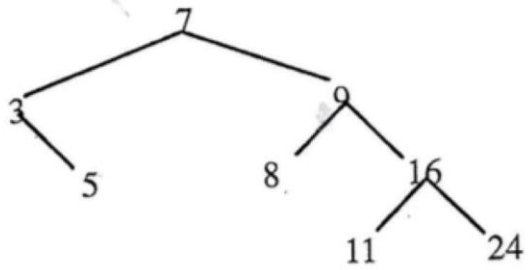


Figure 1. Un exemple d'arbre binaire de recherche.

Comme 6 < clé(A), on appel insérer sur le sous arbre gauche de 7. Or, comme 6 > 3, on appel insérer sur le sous arbre droit de 3. Comme 6 > 5, on appel insérer sur le sous arbre droit de 5. Arrivée à cette étape, comme le sous arbre droit de 5 est vide, on crée un nœud de clé 6, et l'arbre binaire résultant est alors celui de la figure 2.

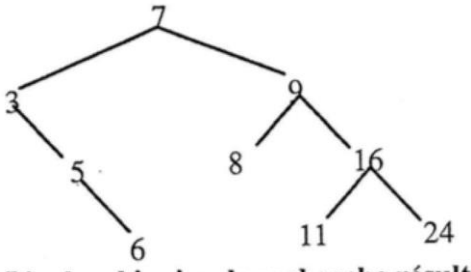


Figure 2. L'arbre binaire de recherche résultant après l'insertion de 6 à l 'arbre binaire de recherche de la figure 1.

L'algorithme récursif qui en résulte est alors le suivant.

Procédure insérer (d x : t ; dr A : Arbre) ;
{Cette procédure utilise en entrée A et x. A est un arbre binaire de recherche, et x est une clé donnée. La procédure insérer consiste à ajouter x à A si x ne figure pas dans A}
Début

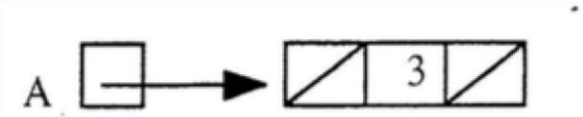
Fin ;

Exemple :

Supposons que l'on veuille construire un arbre binaire de recherche par adjonctions successives de 3, 1, 6.

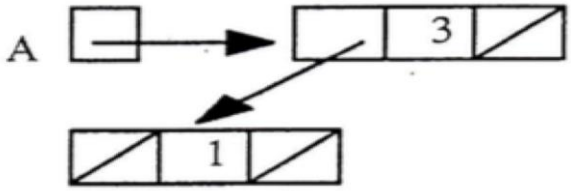
1ère étape : insérer (3, A)

Comme A= nil, l'appel de insérer (3, A) a pour effet



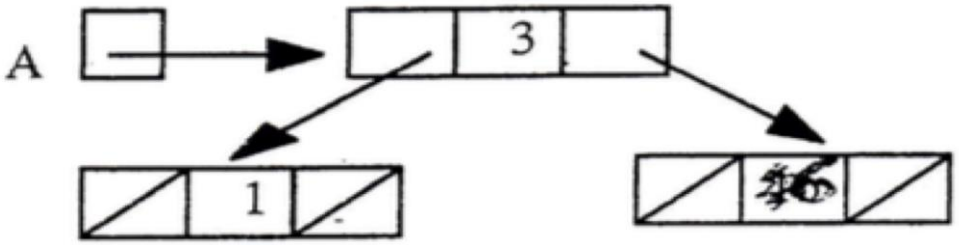
2ème étape : insérer (1, A)

Comme A ≠ nil et comme 1 < 3, l'appel de. insérer (1, A) conduit à insérer (1, A^.fg). Comme A^.fg = nil, on crée le nœud d'étiquette 1.



3ème étape : insérer (6, A)

Comme A ≠ nil et comme 6 > 3, l'appel de. insérer (6, A) conduit à insérer (6, A^.fd). Comme A^.fd = nil, on crée le nœud d'étiquette 6.



6.5. Suppression d'une clé dans un arbre binaire de recherche

Soient A un arbre binaire de recherche, et x une clé donnée. On désire supprimer la clé x de 1'arbre A de sorte que l'arbre résultant soit encore un arbre binaire de recherche. Pour ce faire, on commence par chercher la place de la clé à supprimer x pour ensuite effectuer la suppression proprement dite de x. Or, comme l'arbre résultant doit être un arbre binaire de recherche, la suppression de x va alors dépendre de la place de x dans l'arbre binaire de recherche initial. Soit n le nœud dont x est le contenu. Si n est une feuille (c.-à-d., n est sans fils), on supprime la feuille. Sinon, si n n'a qu'un seul fils, on remplace n par son fils unique. Sinon, si n a deux fils, on trouve deux solutions qui nous permettent de préserver la structure d'arbre binaire de recherche, tout en modifiant le moins possible l'arbre binaire de recherche initial.

- ✓ Première solution : on remplace n par le nœud le plus à droite dans son sous arbre gauche, c.-à-d., par le nœud de plus grande clé dans son sous arbre gauche.
- ✓ Deuxième solution : on remplace n par le nœud le plus à gauche dans son sous arbre droit, c.-à-d., par le nœud de plus petite clé dans son sous arbre droit.

Exemple :

Reprenons l'ABR de la Figure 1.

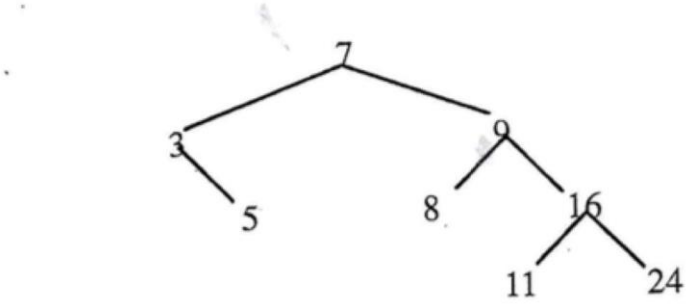
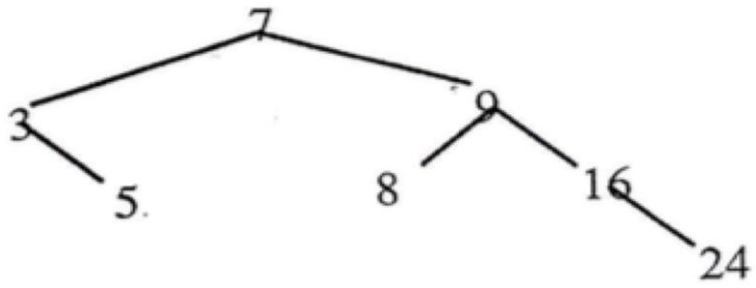


Figure 1. Un exemple d'arbre binaire de recherche.

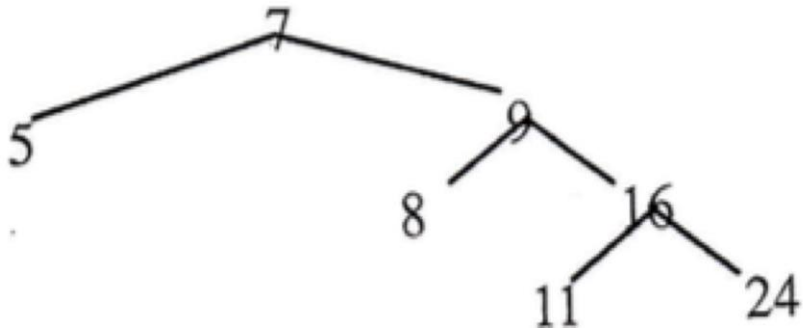
1er cas: supprimer (11, A)

Comme le nœud dont 11 est le contenu est une feuille, on supprime la feuille, et l'arbre résultant A' reste bel et bien un arbre binaire de recherche.



2ème cas: supprimer (3, A)

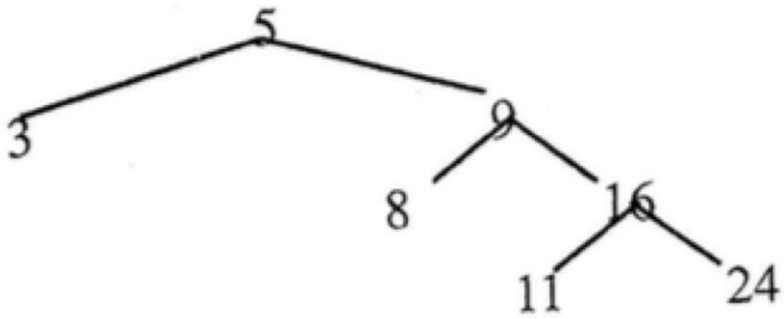
Comme le nœud dont 3 est le contenu a un seul fils, on remplace ce nœud par son fils unique et l'arbre résultant reste bel et bien un arbre binaire de recherche.



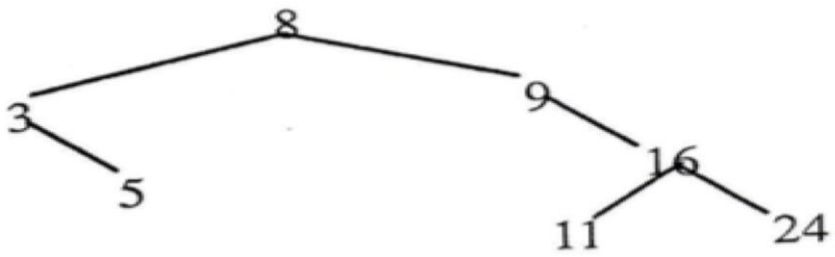
3ème cas: supprimer (7, A)

Comme le nœud dont 7 est le contenu a deux fils, on peut remplacer ce nœud soit par le nœud de plus grande clé dans son sous arbre gauche, soit par le nœud de plus petite clé dans son sous arbre droit.

Si, on remplace le nœud de clé 7 par le nœud de plus grande clé dans son sous arbre gauche, l'arbre binaire de recherche résultant est le suivant.



Si, on remplace le nœud de clé 7 par le nœud de plus petite clé dans son sous arbre droit, l'arbre binaire de recherche résultant est le suivant.



L'algorithme de suppression d'une clé dans un arbre binaire de recherche est le suivant.

Procédure supprimer (d x : t ; dr A : Arbre) ;
{Cette procédure utilise en entrée A et x. A est un arbre binaire de recherche, et x est une clé donnée. La procédure supprimer consiste à supprimer x de A. Si x ne figure pas dans A, elle renvoie A, sinon, elle renvoie A privé de x} var

max : t

Début

```
Si (arbrevide (A) = faux) alors
        Si (x < A^. etiquette_nœud) alors  supprimer
                (x, A^.fg)
        Sinon
                Si (x > A^. etiquette_nœud) alors  supprimer
                        (x, A^.fd)
                Sinon {x=clé(A)}
                        Si arbrevide (A^.fg) alors
```

```
                    A :=A^.fd
                Sinon
                    Si arbrevide (A^.fd) alors
                        A :=A^.fg
                    Sinon {A^.fg n'est pas vide et A^.fd n'est pas vide}
                        Supmax(max, A^.fg)
                        A^. etiquette_nœud :=max
                    Fin si
                Fin si
                Fin si
            Fin si
        Fin si
Fin ;
```

où supmax est une procédure de suppression du maximum dans un arbre binaire de recherche.

```
Procédure supmax (r max : t ; dr A : Arbre) ;
{Cette procédure utilise en entrée A qui est un arbre biliaire de recherche non vide. Elle supprime le nœud de plus grande clé dans A, et retourne la clé de ce nœud dans max}
    Début
        Si arbrevide (A^.fd) alors
            max := A^. etiquette_nœud A
            :=A^.fg
        Sinon
            supmax (max, A^.fd)
        Fin si
    Fin ;
```

Pour les algorithmes de recherche, d'insertion et de suppression dans un arbre binaire de recherche, l'opération fondamentale est la comparaison ente deux clés. Pour ces algorithmes, la complexité au pire est en $\Theta(n)$, et la complexité en moyenne est en $\Theta(\log_2(n))$.