Chapitre 4 : Arbres binaires de recherche (ABR)

Un arbre binaire de recherche est un compromis entre la structure dynamique d'arbre et la facilite de recherche par dichotomie dans un tableau trie.

1 Définition

Un arbre binaire de recherche ou ABR est un arbre binaire non vide étiqueté, où les étiquettes appartiennent à un ensemble totalement ordonné et qui vérifient l'une des 2 conditions suivantes :

- La liste des étiquettes en ordre infixe est croissante au sens large,
- Pour tout nœud x d'étiquette e, les étiquettes de la branche gauche de x sont inférieures ou égales à e, et les étiquettes de la branche droite sont supérieures ou égales à e.

Si A est un ABR alors tout sous-arbre non vide de A est un autre ABR.

Voici quatre exemples d'arbres binaires de recherche qui sont définis sur le même ensemble de valeurs :

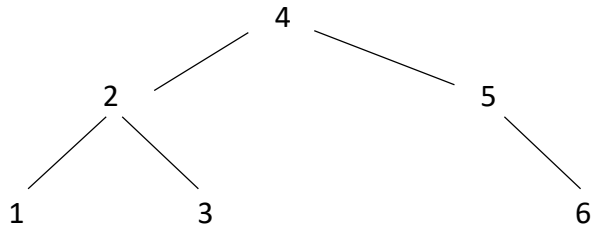


Figure 1 : Arbre binaire de recherche Nº1.

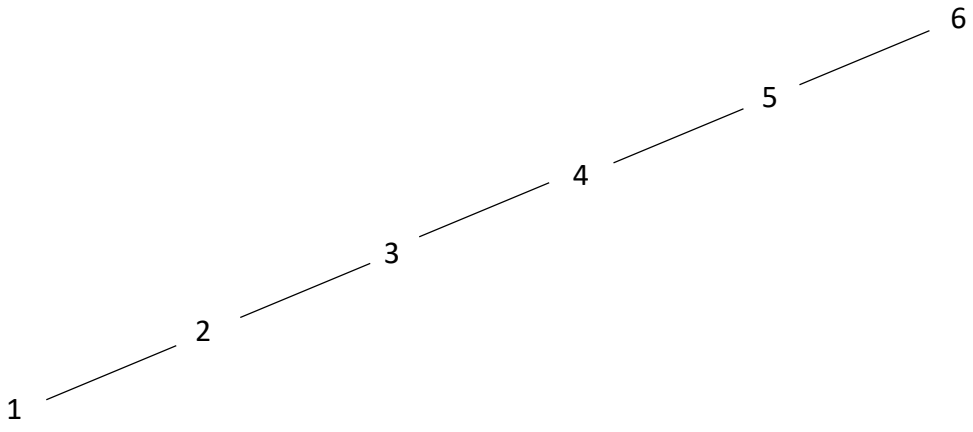


Figure 2 : Arbre binaire de recherche Nº2.

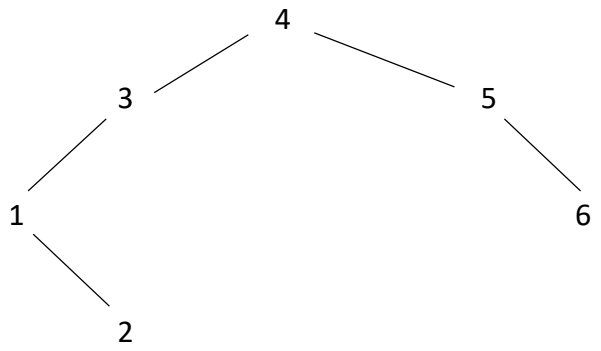


Figure 3 : Arbre binaire de recherche Nº3.

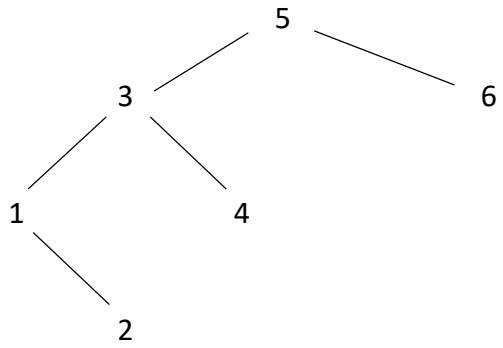


Figure 4 : Arbre binaire de recherche N°4.

Ces arbres binaires de recherche représentent le même ensemble d'étiquettes {1, 2, 3, 4, 5, 6). On en déduit qu'il peut exister plusieurs arbres binaires de recherche définis sur le même ensemble de valeurs, cependant ils se différencient par certaines caractéristiques.

La définition même d'un arbre binaire de recherche fait que tous les nœuds appartenant à un sous-arbre gauche sont inférieurs ou égaux à la racine de ce sous-arbre, et tous les nœuds appartenant à un sous-arbre droit sont supérieurs ou égaux à la racine de ce sous-arbre.
Remarque : généralement, les valeurs dans un ABR sont uniques. On n'admet pas de répétition de valeur pour éviter la confusion. Par exemple si un arbre contient deux fois la valeur 4, est ce que la deuxième valeur 4 est stockée dans le sous-arbre droit ayant pour racine 4 ou dans le sous-arbre gauche ?
➔ La convention adoptée fait que l'on insère cet élément plutôt dans le sous-arbre gauche.

2 Les primitives ABR

Les arbres binaires de recherche sont des arbres binaires qui définissent une relation d'ordre parmi leurs données. Il est possible d'utiliser quelques-unes des primitives que nous avons déjà définies et implémentées. Il est nécessaire cependant de définir des primitives spécifiques permettant :

- D'ajouter un élément dans un arbre binaire de recherche, il est évident que cet élément ne peut pas être ajouté à n'importe quel endroit. Il faut repérer l'endroit adéquat avant l'ajout.

- De supprimer un élément dans un ABR, la suppression nécessite parfois une

2.1 Insertion dans un ABR

Pour insérer un nouvel élément dans un ABR il faut d'abord repérer sa place dans l'arbre, il faut donc le comparer aux éléments déjà existant dans l'arbre.

```
Fonction Insérer (e : élément, A : ABR) : ABR
Début
   si (A = Nil) alors
        A ← CréerArbreBinaire (e, Nil, Nil)
   sinon
      si (Racine(A) >= e) alors
         A.^FilsGauche ← Insérer (e, FilsGauche(A))
      sinon
         A.^FilsDroit ← Insérer (e, FilsDroit(A))
      finsi
   finsi
Fin
```

On peut aussi écrire Insérer autrement et sous la forme de procédure :

```
Procédure Insérer (e : Elément, VAR A : ABR)
Début
  si (A = Nil) alors
       A ← CréerArbreBinaire (e, Nil, Nil)
  sinon
     si (Racine(A) >= e) alors
          A ← CréerArbreBinaire (Racine(A), Insérer (e, FilsGauche(A)), FilsDroit(A))
     sinon
          A ← CréerArbreBinaire (Racine(A), FilsGauche(A), Inserer (e, Fils Droit(A)))
     finsi
  finsi
Fin
```

Exemple :

Nous voulons insérer la valeur 3 dans l'arbre A suivant :

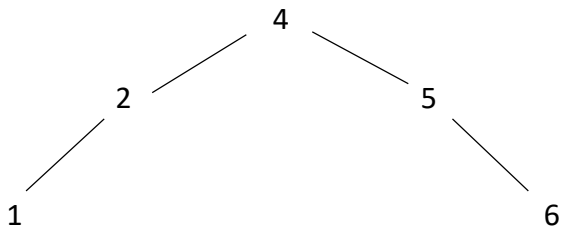
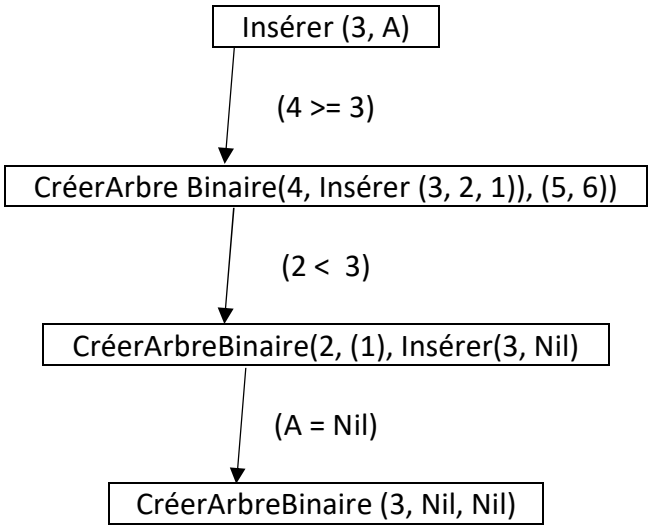
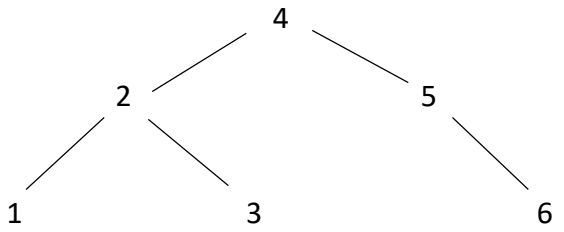


Figure 5: ABR A.

Nous allons appliquer la deuxième version d'insertion sur l'arbre A.



Par l'application de l'instruction **CréerArbreBinaire** (4. **CréerArbreBinaire**(2, (1), **CréerArbreBinaire**(3, Nil, Nil)), (5, 6)), nous obtenons le nouvel arbre A.



Remarque : L'arbre A doit être passé par adresse dans la procédure Insérer.

2.2 Suppression dans un ABR

La suppression dans un ABR est assez compliquée, c'est pour cela que nous allons détailler tous les cas possibles par l'intermédiaire d'exemples.

Pour supprimer un nœud dans un arbre binaire de recherche plusieurs cas de figure peuvent se présenter. Il est toutefois nécessaire d'obtenir un arbre binaire de recherche à l'issue de la suppression.

D'abord il faut chercher l'élément à supprimer, une fois trouvé on se retrouve dans l'une des situations suivantes, soit le nœud à supprimer :

- **le nœud x a un seul fils** : on supprime **x** et on le remplace par ce fils,
- **x est une feuille** : on le supprime et on le remplace par Nil,
- **x a deux fils** : on le supprime et on le remplace par l'élément minimum se trouvant dans son fils droit, (ou par l'élément maximum se trouvant dans son fils gauche).

2.2.1 Illustration

L'arbre A suivant servira pour l'illustration de tous les cas de figure de la suppression.

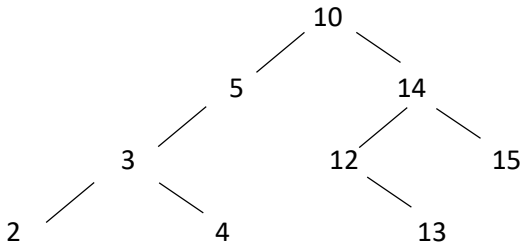


Figure 6 : Arbre A.

Cas de la suppression du noud dont la valeur est 12 : 12 possède un seul fils, un fils droit, une feuille dont la valeur est 13. Il suffit donc de supprimer 12 et de le remplacer par 13. On obtient :

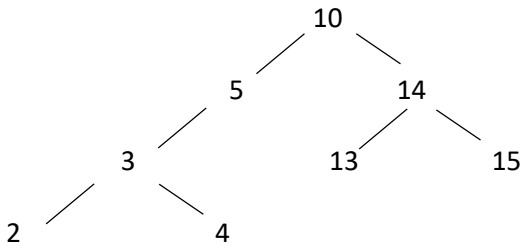


Figure 7 : Arbre A après suppression de 12.

Le nœud ayant pour valeur 13 prend la place de son père et devient fils gauche de 14, alors qu'il était fils droit de 12. On garde la structure d'arbre binaire de recherche intacte, 13 étant inférieur à 14, donc il ne peut être que son fils gauche.

Cas de la suppression du nœud dont la valeur est 4 : 4 n'a pas de fils, on le remplace par Nil. On obtient :

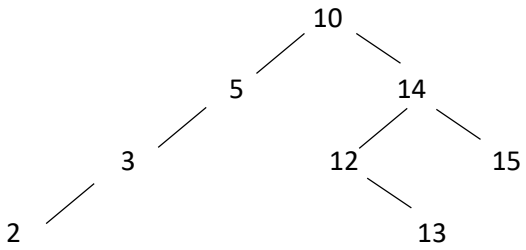


Figure 8 : Arbre A après suppression de 4.

Cas de la suppression du nœud 14 : 14 possède 2 fils : un fils droit ayant pour valeur 15 et un fils gauche ayant pour valeur 12.

- On peut le remplacer par l'élément minimum se trouvant dans son sous-arbre droit : 15 étant le seul nœud dans le sous-arbre droit de 14, il est le seul à pouvoir le remplacer.

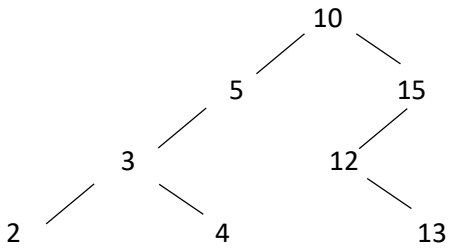


Figure 9 : A après suppression 14.

- On peut aussi le remplacer par l'élément maximum se trouvant dans son sous-arbre gauche. Après une rapide recherche nous trouvons l'élément 13.

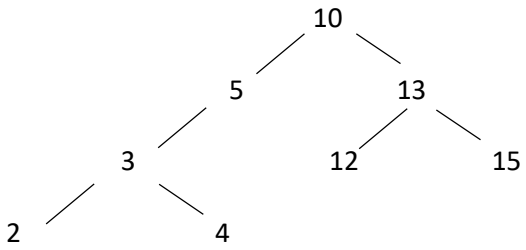


Figure 10 : A après la suppression de 14.

Pour mieux assimiler le fonctionnement de la suppression d'un nœud ayant 2 fils dans un ABR, reprenons l'exemple avec le nœud racine 10.

Pour supprimer le nœud racine, on remplace sa valeur par celle de l'élément qui est minimum dans son sous-arbre droit, et on supprime ce dernier de l'arbre. (L'élément

minimum dans un sous-arbre droit est en fait le plus petit élément parmi les plus grands éléments de l'arbre A.)

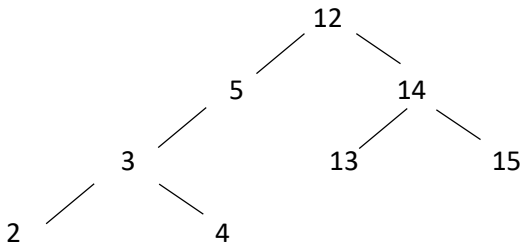


Figure 11 : Arbre A après suppression de 10.

Dans cet exemple on remarque mieux ce qui se passe lors de la suppression d'un nœud possédant 2 fils. Pour supprimer la racine 10, un certain nombre d'étapes a été suivi :

- Recherche de l'élément à supprimer,
- Localisation de cet élément et constatation qu'il possède 2 fils,
- Recherche de l'élément minimum de son sous-arbre droit, cet élément est 12
- Le nœud qui contient la valeur 12 est supprimé de l'arbre, en étant remplacé par son fils droit 13,
- La valeur 12 est remontée pour remplacer la valeur de la racine, le nœud racine 10 est par ce fait supprimé.

D'où pour supprimer un nœud ayant 2 fils il faut d'abord supprimer le minimum de son sous-arbre droit (ou le maximum de son sous-arbre gauche).

L'arbre ci-dessous est obtenu si on remplace la racine par l'élément maximum de sons sous-arbre gauche.

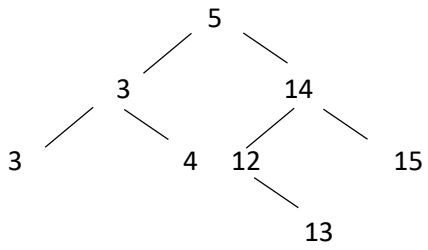


Figure 12: A après suppression de 10 et remplacement par 5.

Il est très simple de rechercher le minimum et le maximum dans un arbre binaire de recherche. On parcourt l'arbre :

- En choisissant toujours la branche gauche, le minimum est le dernier nœud n'ayant pas de fils gauche,
- En choisissant la branche droite, le maximum est le dernier nœud n'ayant pas de fils droit.

Pour supprimer un nœud minimum il faut le remplacer par son fils unique, autrement dit son fils droit même s'il est égal à Nil. C'est le cas du nœud 13 qui remplace son père 12 dans la Figure 11.

Pour supprimer un nœud maximum, on le remplace par son fils gauche même s'il est égal à Nil. C'est le cas du sous-arbre dont la racine est 3 et qui remplace le nœud 5 dans la Figure 12.

2.2.2 Algorithme

La suppression dans un ABR nécessite deux modules : un module appelé **Supprimer** qui cherche le nœud à supprimer et qui fait appel à un module secondaire **SuppMin**.

Le module **SuppMin** assure trois fonctions :

1. Trouver le nœud dont la valeur représente le minimum dans un arbre binaire de recherche, (le sous-arbre droit du nœud à supprimer)
2. Retourner la valeur de ce nœud, qui sert pour effectuer le remplacement de la valeur du nœud à supprimer,
3. Supprimer ce nœud et ajuster l'arbre.

```
Fonction SuppMin (A : ArbreBinaire) : ArbreBinaire
   var
      T: ^Noeud
   Début
      si (A # Nil) alors
         si (A.^FilsGauche # Nil) alors
               Retourner SuppMin (A.^FilsGauche)
         sinon
               T ← A
               A ← A.^FilsDroit
               Retourner (T)
         finsi
      finsi
Fin
```

L'algorithme **Supprimer** permet de supprimer le noud dont la valeur est égale à ***e*** dans l'arbre. Cet algorithme assure les fonctions suivantes :

- Il compare l'élément ***e*** à supprimer avec la racine courante de l'arbre A
- Si la racine est supérieure à **e** alors, il appelle récursivement **Supprimer** pour le sous-arbre gauche de A,
- Si la racine est inférieure alors, il appelle récursivement **Supprimer** pour le sous-arbre droit de A,
- S'il y a égalité, **Supprimer** distingue les différents cas possibles :
-
    - S'il existe un seul fils, (respectivement aucun fils), alors nœud est remplacé par le fils existant, (respectivement par Nil),

- S'il existe deux fils, le module **SuppMin** est appelé, la valeur du nœud est remplacée par le minimum du sous-arbre droit, et ce minimum est supprimé de l'arbre.

```
Fonction Supprimer ( A ArbreBinaire, e Element) : ArbreBinaire
Var
   T : ^ Noeud
Debut
   si (A = Nil ) alors
         Retourner ( A )
   sinon
       si (Racine (A) > e) alors
             Retourner Supprimer (A.^FilsGauche, e)
      sinon
          si (Racine(A) <e) alors
             Retourner Supprimer (A.^FilsDroit, e)
          sinon
          // Cas de e = Racine (A)
          // On considère les cas, e a un seul fils ou 2 fils
               si (A.^FilsGauche = Nil) alors
                     A ← A.^FilsDroit
               sinon
                    si (A.^FilsDroit = Nil) alors
                       A ← A.^FilsGauche
                    sinon
                    // Cas ou 2 fils existent
                    T ← SuppMin (A.^FilsDroit)
                    A.^Valeur ← T .^Valeur
                    Retourner (A)
                   finsi
              finsi
          finsi
      finsi
  finsi
Fin
```

3 Recherche dans un ABR

Pour chercher une valeur *v* dans un arbre binaire de recherche le processus est le suivant :

- Si l'arbre est vide alors la recherche a échoué,
- Si la racine de l'arbre parcouru est égale à la valeur **v** alors la recherche a réussi,
- Si la racine de l'arbre parcouru est supérieure (ou égale) à **v** alors il faut continuer la recherche dans le sous-arbre gauche,
- Si la racine de l'arbre parcouru est inférieure à **v** alors il faut continuer la recherche dans le sous-arbre droit.

```
Fonction Recherche (A : ArbreBinaire, e : Elément) : Booléen
Début
    si (A = Nil) alors
       Retourner Faux
    sinon
       si (A = Racine (A)) alors
          Retourner Vrai
       sinon
          si (e > Racine (A)) alors
             Retourner Recherche (A.^FilsDroit, e)
          sinon
             Retourner Recherche(A.^FilsGauche, e)
          finsi
       finsi
    finsi
Fin
```