Chapitre 3 – Les arbres binaires

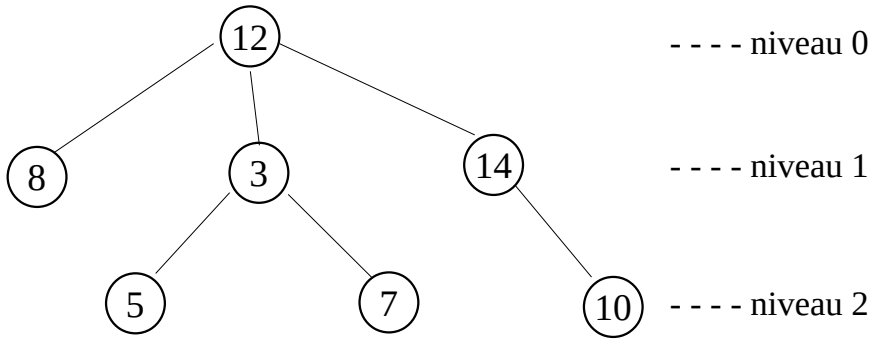
1. Présentation

Un arbre est une structure de données dynamique qui regroupe dans un ordre hiérarchique des éléments de même type appelés noeuds.

Un arbre peut être vide ou bien constitué d'un ou de plusieurs nœuds contenant chacun une valeur.

Chaque nœud de l'arbre est relié à zéro, un ou plusieurs autres nœuds placés au dessous de lui et qui sont appelés ses fils. Les nœuds qui n'ont pas de fils sont appelés nœuds terminaux ou feuilles.

La figure suivante représente un exemple d'arbre contenant des entiers.



2. Terminologie

Chaque nœud se trouve à un niveau de l'arbre. Au niveau 0 (sommet de l'arbre) se trouve un nœud particulier appelé la racine. Les fils de la racine se trouvent au niveau 1, et ainsi de suite.

La **hauteur** d'un arbre correspond au nombre de ses niveaux. La hauteur de l'arbre de l'exemple précédent est 3.

La **taille** d'un arbre correspond au nombre de ses noeuds. La taille de l'arbre de l'exemple précédent est 7.

Un **arbre général** (ou **arbre n-aire**) est un arbre dont les nœuds ont chacun zéro ou plusieurs fils.

Un **arbre binaire** est un arbre dont les nœuds ont chacun au plus 2 fils.

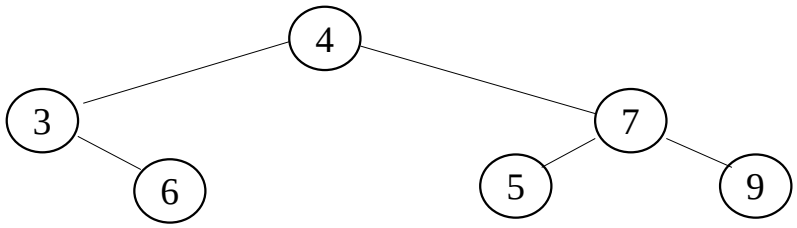
Dans la suite, nous allons nous limiter au cas des arbres binaires mais les algorithmes que nous allons voir pourront être généralisés au cas des arbres généraux .

3. Les arbres binaires

3-1. Présentation

Un arbre binaire est un arbre dont les nœuds ont au plus deux fils. On utilise les termes de fils gauche et de fils droit d'un nœud.

La figure suivante représente un exemple d'arbre binaire contenant des entiers :



Le nœud contenant la valeur 4 est la racine de l'arbre et a pour fils gauche le nœud contenant la valeur 3 et pour fils droit le nœud contenant la valeur 7.

Le nœud contenant la valeur 3 n'a pas de fils gauche. Son fils droit est le nœud terminal (feuille) contenant la valeur 6.

On peut remarquer qu'un arbre a une forme récursive car chaque nœud autre que la racine peut être considéré comme la racine d'un sous-arbre.

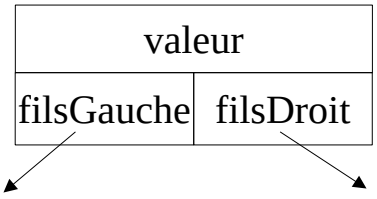
3-2. Implémentation d'un arbre binaire avec des pointeurs

Un arbre est une structure de données dynamique dont les nœuds sont ajoutés au fur et à mesure par allocation dynamique. Pour maintenir la structure d'ensemble, il est nécessaire de garder au niveau de chaque nœud un lien vers ses nœuds fils. Pour cela on peut utiliser des pointeurs pour garder les adresses des nœuds fils.

Un pointeur garde l'adresse du nœud racine et son nom est utilisé pour désigner l'arbre.

Un nœud d'un arbre binaire peut donc être défini comme un enregistrement (ou structure) avec trois champs :

- un champ pour la valeur du nœud.
- deux autres champs qui sont respectivement un pointeur vers le nœud fils gauche et un pointeur vers le nœud fils droit



Par exemple, dans le cas d'un arbre binaire d'entiers, on peut définir un type pour représenter un nœud de la manière suivante :

```
Type
    noeud = Enregistrement
        valeur : entier
        filsGauche: pointeur sur nœud
        filsDroit: pointeur sur noeud
    FinEnregistrement
```

Pour créer un arbre binaire, il faut déclarer une variable pointeur qui va garder l'adresse du nœud racine de l'arbre. Le nom de ce pointeur sera utilisé pour désigner l'arbre binaire.

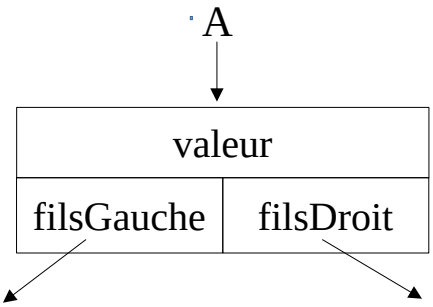
Exemple

```
Variable
    A :  pointeur sur noeud
```

Dans cet exemple, le pointeur A va pointer sur la racine de l'arbre appelé arbre A.

L'arbre A est vide si A contient la valeur NULL.

Pour un arbre non vide, A est un pointeur sur le nœud racine :



Dans ce cas :

*A désigne ce nœud racine

(*A).valeur, ou bien A->valeur, désigne le champ valeur de ce noeud

(*A).filsGauche, ou bien A->filsGauche, désigne le champ filsGauche du nœud qui est un pointeur sur le nœud fils gauche

(*A).filsDroit, ou bien A->filsDroit, désigne le champ filsDroit du nœud qui est un pointeur sur le nœud fils gauche

Remarque

Pour des besoins de simplification, on peut définir un type appelé ***Arbre*** qui est l'équivalent du type ***pointeur sur noeud*** de la manière suivante :

```
Type
    Arbre = pointeur sur noeud
```

Dans ce cas, on pourra créer l'Arbre A avec la déclaration suivante :

```
Variable
    A : Arbre
```

Traduction en langage C

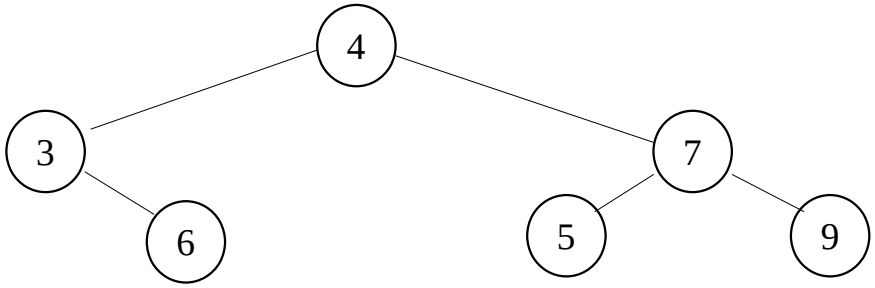
```
struct noeud
{
    int valeur;
    struct noeud* filsGauche ;
    struct noeud* filsDroit ;
} ;
typedef struct nœud nœud ;      //  noeud devient équivalent à struct noeud
typedef noeud* Arbre ;          //  Arbre devient équivalent à  noeud*
int main( )
{
    struct noeud* A ;     / ou bien : Arbre A ;
    A = NULL ;        //  l'arbre est vide
     return 0 ;
}
```

3-3. Ajout d’un nœud à un arbre binaire

On peut définir une fonction AjouterNoeud qui ajoute un nœud dans un arbre binaire. Cette fonction prend en paramètres la valeur du nœud à ajouter, l'adresse de son fils gauche et l'adresse de son fils droit. En cas de succès, elle retourne l'adresse du nœud ajouté. En cas d’échec, elle retourne NULL.

```
AjouterNoeud( v : entier, Ag:pointeur sur noeud ,  Ad:pointeur sur noeud ) : pointeur sur noeud
Début
    Variable
        A : pointeur sur noeud
    A = Allouer(1, nœud) )
    Si (A!= NULL)
        A->valeur = v
        A->filsGauche = Ag
        A->filsDroit = Ad
    FinSi
    retourrner A
FinFonction
```

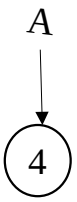
Cette fonction peut être appelée plusieurs fois pour construire l’arbre binaire suivant :



Variable

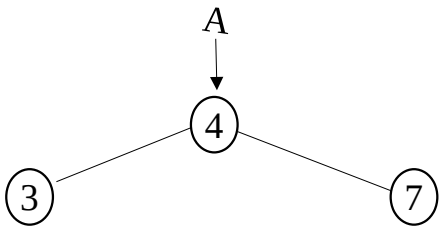
A : pointeur sur noeud

A = AjouterNoeud(4, NULL, NULL)

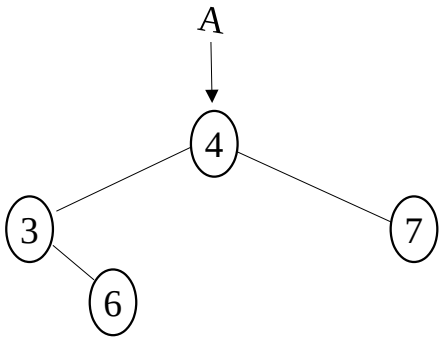


A->filsGauche = AjouterNoeud(3, NULL, NULL)

A->filsDroit = AjouterNoeud(7, NULL, NULL)

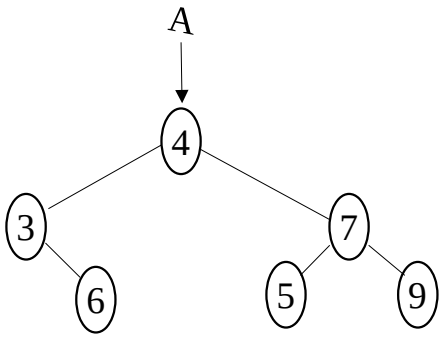


A->filsGauche->filsDroit = AjouterNoeud(6, NULL, NULL)



A->filsDroit->filsGauche = AjouterNoeud(5, NULL, NULL)

A->filsDroit->filsDroit = AjouterNoeud(9, NULL, NULL)



A la fin de l'exécution des instructions, le pointeur A pointe sur la racine de l'arbre construit et permet de désigner cet arbre comme étant l'arbre A.

3-4. Parcours d’un arbre binaire

Parcourir un arbre, c’est visiter chacun de ses nœuds pour y effectuer un certain traitement, par exemple pour afficher sa valeur. On peut distinguer le parcours préfixe, le parcours infixe et le parcours postfixe.

3-4-1. Parcours préfixe

Avec le parcours préfixe (appelé aussi parcours en pré-ordre ou parcours RGD), la racine de l'arbre est d'abord traitée. Ensuite le sous-arbre gauche est parcouru pour le traitement de ses nœuds. Enfin le sous-arbre droit est parcouru pour le traitement de ses nœuds.

D’où la fonction récursive suivante :

```
ParcoursPrefixe( A : pointeur sur noeud )
Début
    Si (A!= NULL)
        Ecrire( A->valeur )
        ParcoursPrefixe( A->filsGauche )
        ParcoursPrefixe( A->filsDroit )
    FinSi
FinFonction
```

Si on affiche l’arbre de l'exemple précédent avec ce type de parcours, on aura :

4 3 6 7 5 9

3-4-2. Parcours infixe

Avec le parcours infixe (appelé aussi parcours en ordre symétrique ou parcours GRD), le sous-arbre gauche est d'abord parcouru pour le traitement de ses nœuds. Ensuite la racine est traitée. Enfin le sous-arbre droit est parcouru pour le traitement de ses nœuds. D'où la fonction récursive suivante :

```
ParcoursInfixe( A : pointeur sur noeud )
Début
    Si (A!= NULL)
        ParcoursInfixe( A->filsGauche )
        Ecrire( A->valeur )
        ParcoursInfixe( A->filsDroit )
    FinSi
FinFonction
```

Si on affiche l'arbre de l'exemple précédent avec ce type de parcours, on aura :

3 6 4 5 7 9

3-4-3. Parcours postfixe

Avec le parcours postfixe (appelé aussi parcours en post-ordre ou parcours GDR), le sous-arbre gauche est d'abord parcouru pour le traitement de ses nœuds. Ensuite le sous-arbre gauche est parcouru pour le traitement de ses nœuds. Enfin la racine est traitée. D'où la fonction récursive suivante :

```
ParcoursPostfixe( A : pointeur sur noeud )
Début
    Si (A!= NULL)
        ParcoursPostfixe( A->filsGauche )
        ParcoursPostfixe( A->filsDroit )
        Ecrire( A->valeur )
    FinSi
FinFonction
```

Si on affiche l'arbre de l'exemple précédent avec ce type de parcours, on aura :

6 3 5 9 7 4

3-5. Hauteur d’un arbre binaire

La fonction récursive suivante retourne (donne) la hauteur d’un arbre binaire qui est égale au nombre de niveaux dans l’arbre.

La hauteur d’un arbre vide est égale à 0.

```
Hauteur( A : pointeur sur noeud ) : entier
Début
    Si (A == NULL)
        Retourner 0
    Sinon
        Retourner 1 + Maximum( Hauteur( A->filsGauche), Hauteur( A->filsDroit))
    FinSi
FinFonction
```

La fonction Maximum qui donne le maximum de deux entiers peut être définie comme suit :

```
Maximum( a : entier, b:entier) : entier
Début
    Si (a > b)
        Retourner a
    Sinon
        Retourner b
    FinSi
FinFonction
```

3-6. Taille d’un arbre binaire

La fonction récursive suivante retourne (donne) la taille d’un arbre binaire qui est égale au nombre de nœuds (éléments) de l’arbre.

La taille d’un arbre vide est égale à 0.

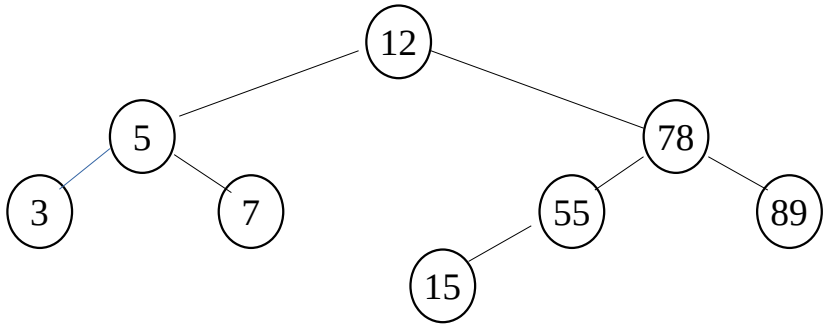
```
Taille( A : pointeur sur noeud ) : entier
Début
    Si (A == NULL)
        Retourner 0
    Sinon
        Retourner 1 +  Taille( A->filsGauche) + Taille( A->filsDroit)
    FinSi
FinFonction
```

4. Arbre binaire de recherche

4-1. Définition

Un arbre binaire de recherche (ABR) est un arbre binaire vérifiant la propriété suivante : pour tout nœud ayant une valeur v, tous ses descendants à gauche ont des valeurs inférieures ou égales à v et tous ses descendants à droite ont des valeurs strictement supérieures à v.

La figure suivante représente un exemple d'arbre binaire de recherche.



4-2. Ajout d’un nœud dans un arbre binaire de recherche

Pour ajouter un nœud avec une valeur dans un arbre binaire de recherche, on peut appeler la fonction récursive suivante qui alloue de l’espace pour le noeud qui va contenir la valeur puis accroche ce nœud en tenant compte de la propriété de l’arbre. Cette fonction retourne 1 en cas de succès et 0 en cas d’échec.

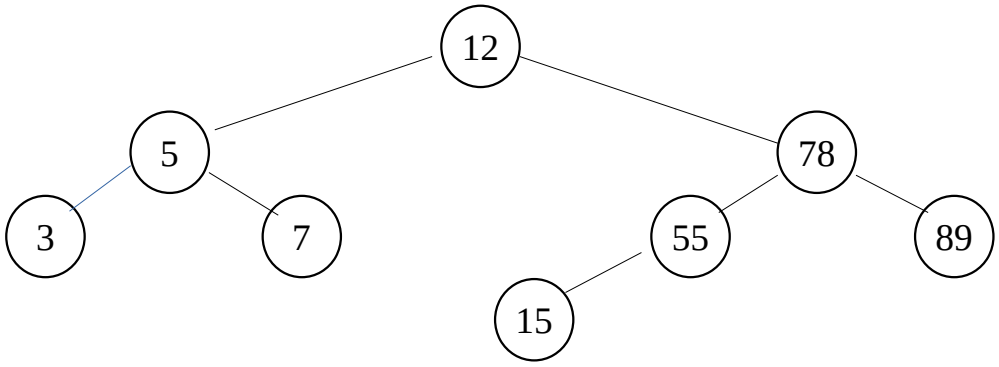
```
AjouterNoeud( E-S A:pointeur sur noeud , int v : Entier) : entier
Début
    Si (A == NULL)
        A = Allouer(1, nœud)
        Si (A==NULL)   // Si l'allocation échoue
            Retourner 0
        Sinon
            A->valeur = v
            A->filsGauche = NULL
            A->filsDroit = NULL
            Retourner 1
        FinSi
    Sinon Si ( v ≤  A->valeur)
        AjouterNoeud(A->filsGauche , v )
    Sinon
        AjouterNoeud( A->filsDroit , v )
    FinSi
FinFonction
```

Cette fonction peut être appelée plusieurs fois (dans une boucle par exemple) pour construire un arbre binaire de recherche avec des valeurs données par l'utilisateur.

Exercice

Dessiner l'arbre binaire de recherche obtenu en ajoutant successivement des nœuds avec les valeurs suivantes: 12, 78, 55, 89, 5, 15, 7, 3.

Réponse



4-3. Recherche d’une valeur dans un arbre binaire de recherche

Pour rechercher une valeur dans un arbre binaire de recherche, on peut appeler la fonction récursive suivante qui prend en paramètre l’arbre et la valeur à rechercher et retourne l’adresse du nœud contenant la valeur ou NULL en cas d’absence :

```
 Rechercher( A : pointeur sur noeud , v : entier ): pointeur sur noeud
Début
    Si (A == NULL)
        retourner NULL
    Sinon Si ( v == A->valeur)
        retourner A
    Sinon si ( v <  A->valeur)
        retourner Rechercher(A->filsGauche , v)
    Sinon
        retourner Rechercher(A->filsDroit , v) ;
FinFonction
```

L2 Informatique 2024 – 2025

TD 3 – Les arbres binaires

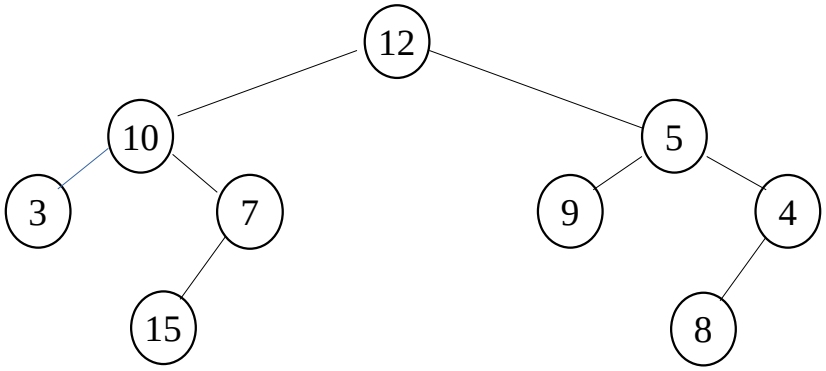
Exercice 1. Arbre binaire

Dans un programme appelé arbre.c :

1-1) Ajouter la définition suivante (vue en cours en pseudo-code) du type noeud représentant un éléments d'arbre binaire d'entiers :

1-2) Donnez la définition de la fonction AjouterNoeud (vue en cours en pseudo-code) qui ajoute un nœud dans un arbre binaire quelconque. Cette fonction prend en paramètre la valeur du nœud à ajouter, l'adresse de son fils gauche et l'adresse de son fils droit.

Appeler cette fonction pour construire l'arbre binaire suivant:



1-3) Définir les fonctions d'affichage d'un arbre binaire en modes préfixe, infixe et postfixe. Appeler ces fonctions pour afficher l'arbre précédemment construit.

1-4) Définir la fonction récursive Hauteur qui retourne la hauteur d'un arbre binaire donné en paramètre. Appeler cette fonction pour trouver la hauteur de l'arbre précédemment construit.

1-5) Définir la fonction récursive Taille qui retourne la taille (le nombre de noeuds) d'un arbre binaire donné en paramètre. Appeler cette fonction pour trouver la taille de l'arbre précédemment construit.

1-6) Définir la fonction récursive AfficherFeuilles qui affiche les valeurs des feuiiles (nœuds terminaux) de l'arbre. Appeler cette fonction pour afficher les feuilles taille de l'arbre précédemment construit.

Exercice 2. Arbre binaire de recherche

Dans un algorithme

2-1. Donner la définition du type Arbre représentant un arbre binaire dont les valeurs sont de type entier.

2-2. Définir une fonction itérative (non récursive) qui ajoute dans un arbre binaire de recherche un nœud avec une valeur donnée en paramètre.

Appeler cette fonction pour construire un arbre binaire de recherche avec des valeurs données par l'utilisateur.

2-3. Définir les fonctions d'affichage d'un arbre binaire en modes préfixe, infixe et postfixe.

Appeler ces fonctions pour afficher l'arbre binaire de recherche précédemment construit.

2-4. Définir une fonction itérative (non récursive) qui recherche une valeur dans un arbre binaire de recherche et qui retourne l'adresse du nœud qui la contient ou NULL en cas d'absence.

Appeler cette fonction pour rechercher l'arbre binaire de recherche précédemment construit une valeur donnée par l'utilisateur.

2-5) Définir les fonctions MinArbre et MaxArbre qui retournent respectivement le minimum et le maximum des valeurs d'un arbre binaire de recherche non vide donné en paramètre.

Appeler ces fonctions pour trouver puis afficher le minimum et le maximum l'arbre binaire de recherche précédemment construit.

Remarque :

Le minimum d'un arbre binaire de recherche non vide se trouve dans le noeud qui est le plus à gauche.

Le maximum d'un arbre binaire de recherche non vide se trouve dans le noeud qui est le plus à droite.

2-6. Définir une fonction récursive qui supprime d'un arbre binaire de recherche donné en paramètre le nœud contenant une valeur donnée en paramètre de telle sorte que l'arbre reste un arbre binaire de recherche après la suppression.

Traduire cet algorithme en programme en langage C.