Les arbres binaires

1. Encadrement de la hauteur d'un arbre binaire

La hauteur d'un arbre binaire dépend du choix des racines des arbres/sous arbres. Selon ce choix un même arbre peut être complet ou filiforme (voir figure 1)

La hauteur d'un arbre filiforme de taille **n** est égale à **n−1.**
La hauteur d'un arbre complet de taille **n** est égale à $[\log_2(n)]$ (arrondi à l'entier inférieur)

Un arbre filiforme et un arbre complet étant deux cas extrêmes, on peut encadrer la hauteur h d'un arbre binaire quelconque de taille n par :

$$[\log_2(n)] \leq h \leq n-1$$

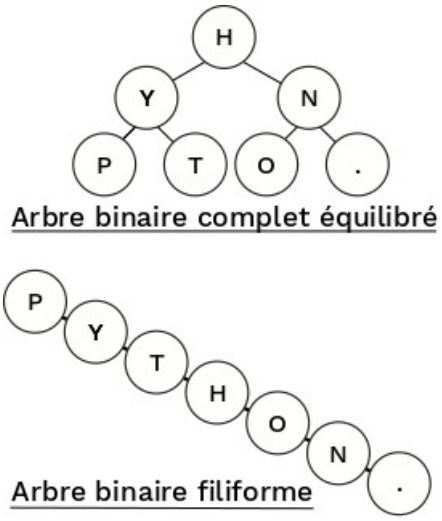

Figure 1: Arbre binaire

2.Arbre Binaire de Recherche

Un arbre binaire de recherche (ABR) (*binary search tree* ou *BST*) est un arbre binaire tel que :

- ➔ les clefs des nœuds doivent être ordonnables (il doit exister une relation d'ordre)
- ➔ pour chacun de ses nœuds:
  - ✗ chaque nœud du sous-arbre gauche a une clé inférieure ou égale à celle du nœud considéré.
  - ✗ chaque nœud du sous-arbre droit possède une clé supérieure à celle-ci

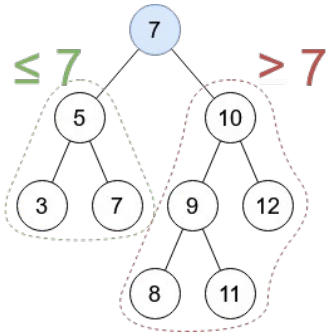

Figure 2: Arbre binaire de recherche

Un ABR permet de rechercher rapidement (faible coût) des enregistrements dont les clés sont données par les nœuds de l'arbre.

Complexité d'un algorithme de recherche dans un ABR

La complexité de l'algorithme de recherche dans un ABR dépend de la forme de notre arbre.

- ➔ La complexité est **O(n)** si l'arbre est filiforme
- ➔ La complexité est **$O(\log_2(n))$** si l'arbre est équilibré

Principaux algorithmes

```
class Abr :
    def __init__(self,d, gauche=None, droit=None) :
        '''Construit un arbre binaire de recherche, constituite d'un noeud
        d'étiquette d, d'un sous arbre gauche et un sous arbre droit.
        gauche = None et droit = None par defaut -> construction d'une
        feuille)'''
        self.data = d
        self.sag = gauche
        self.sad = droit

    def est_feuille(self) :
        '''Retourne vrai si le noeud est une feuille'''
        return self.sag==None and self.sad==None

    def nbre_feuilles(self) :
        ''' retourne le nombre de feuilles de l'arbre'''
        if self.est_feuille() : return 1
        elif self.sad == None : return self.sag.nbre_feuilles()
        elif self.sag == None : return self.sad.nbre_feuilles()
        else : return self.sag.nbre_feuilles() + self.sad.nbre_feuilles()

    def nbre_noeuds(self, debug = False) :
        ''' retourne le nombre de noeuds de l'arbre'''
        if self.est_feuille() : return 0
        elif self.sad == None : return self.sag.nbre_noeuds() + 1
        elif self.sag == None : return self.sad.nbre_noeuds() + 1
        else : return 1 + self.sag.nbre_noeuds() + self.sad.nbre_noeuds()

    def hauteur(self) :
        '''Renvoie la hauteur de l'arbre'''
        if self.est_feuille() : return 0
        elif self.sad == None : return self.sag.hauteur() + 1
        elif self.sag == None : return self.sad.hauteur() + 1
        else : return 1+ max(self.sag.hauteur(), self.sad.hauteur())

    def rechercher(self, n : int) :
        '''Recherche la presence de n dans l'arbre'''
        if self.data == n : return True
        elif self.est_feuille() : return False
        elif self.data > n : return self.sag.rechercher(n)
        elif self.data <= n : return self.sad.rechercher(n)

    def inserer(self, n :int) :
        '''Insere n dans l'arbre de recherche'''
        if self.data > n :
            if self.sag == None : self.sag = Abr(n)
            else : self.sag.inserer(n)
        else :
            if self.sad == None : self.sad = Abr(n)
            else : self.sad.inserer(n)
```