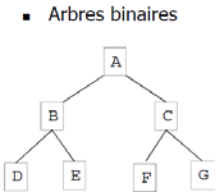
Chapitre 07 : Initiation aux algorithmes des arbres

1. Définitions

- Un arbre binaire est une structure de données de type hiérarchique.
- Les éléments constituant un arbre ont pour nom : racine, nœuds, feuilles fils gauche, fils droit et père.
- Le nœud initial est nommé racine.
- Les éléments terminaux sont des feuilles.
- Dans un arbre binaire, chaque élément possède au plus deux éléments fils au niveau inférieur, habituellement appelés fils gauche et fils droit.
- Le parcours d'un arbre peut se faire en profondeur ou en largeur de la gauche vers la droite (resp : de la droite vers la gauche)
- Un arbre binaire est un graphe (un ensemble de sommets) connexe (il n'y a pas de sommet isolé) sans cycle (il n'y a pas de boucle) dont la représentation graphique est la suivante :

  - A,B,C,D,E,F,G : sont des nœuds
  - A : est une racine
  - D,E,F,G : sont des feuilles



- A l'exception du nœud qui est au sommet de l'arbre et qui est un nœud privilégié appelé "racine", tous les autres nœuds possèdent un père.
- Un nœud qui n'a pas de fils est appelé une feuille
- Un arbre binaire est **un arbre de recherche** ssi les nœuds sont énumérés lors d'un parcours infixe en ordre croissant
- Arbre **parfait complet** c'est un arbre binaire dont le nombre de nœuds de chaque niveau N est égale $\mathbf{2^N}$
- Arbre **parfait** c'est un arbre binaire dont tous les nœuds de chaque niveau sont présents sauf éventuellement au dernier niveau où il peut manquer des nœuds (nœuds terminaux = feuilles), dans ce cas l'arbre parfait est un arbre binaire incomplet et les feuilles du dernier niveau doivent être regroupées à partir de la gauche de l'arbre.

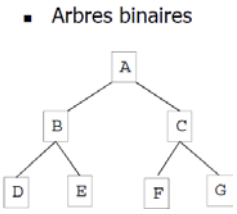
1. Représentation d'un arbre

Il existe différentes manières de représenter un arbre en machine. A l'aide :

- d'une classe
- d'une liste de listes
- d'une matrice
- d'un dictionnaire

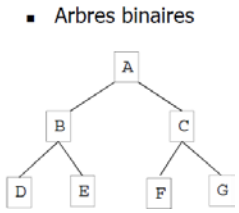
Exemple01(représentation sous forme d'une classe)

```
class arbre:
   def __init__( self,info=None ,g = None,d = None):
       self.info = info
       self.g  = g
       self.d  = d
def inserer(A,info,pos):
     if pos=='r': A.info=info
     elif pos=='g': A.g=arbre(info)
     else:A.d=arbre(info)
A=arbre()
inserer(A,'A','r')
inserer(A,'B','g')
inserer(A,'C','d')
inserer(A.g ,'D','g')
inserer(A.g,'E','d')
inserer(A.d,'F','g')
inserer(A.d,'G','d')
```



Exemple02(représentation sous forme de listes)

- chaque sous-arbre sera représenté à l'aide d'une liste :
  - Nœud = [*nomDuNoeud, [liste sous arbre]*]
- Chaque feuille sera représentée à l'aide d'une liste
  - Feuille =[*nomDeLaFeuille , [None,None]*]

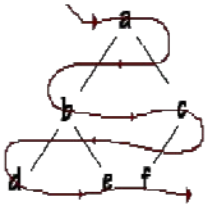


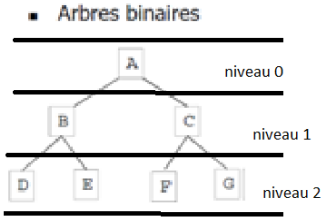
Arbre1=['A', [['B', [['D', [None,None]], ['E', [None,None]]]], ['C', [['F', [None,None]], ['G', [None,None]]]]]]

Exemple03(La représentation sous forme de dictionnaire)

A={'A':['B','C'],'B':['D','E'],'C':['F','G'], 'D' :[None, None], 'E' :[None, None], 'F' :[None, None], 'G' :[None, None]}

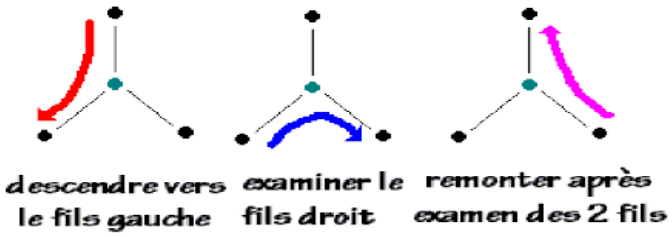
2. **Parcours en largeur :**





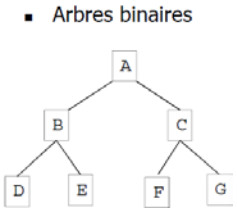
Hauteur (arbre)=3

3. **Parcours en profondeur :**



4. **Le tri Maximier (tri en utilisant un arbre binaire) :voir l'exercice 04**

Exercice 01(La représentation d'un arbre sous forme de classe)



- Soit l'arbre binaire suivant :
- Mettez en œuvre les opérations suivantes :

1. Définir une classe Arbre permettant de déclarer des objets de type Arbre
2. **Insertion dans un arbre**
   Définir la fonction suivante:
   a) **def inserer(A,info,pos):** qui permet d'insérer l'information **info** dans l'arbre **A** à la position **pos**.
3. **Affichage d'un arbre (parcours en profondeur)**
   Définir les fonctions suivantes :
   a) **def inordre(A)**
   b) **def préordre(A)**
   c) **def postordre(A)**
4. **Affichage d'un arbre (parcours en largeur)**

Définir les fonctions suivantes :

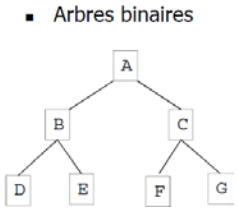
a) **def niveau (A,N)** :qui permet d'afficher le niveau **N**
b) **def nombreDeNœudsParNiveau (A,n)**: qui retourne le nombre de Nœuds d'un niveau **N**
c) **def nombreDeNiveaux (A):** qui retourne le nombre de niveaux
d) **def nombreDeNœuds (A)**: qui retourne le nombre de Nœuds dans un arbre
e) **def affichageEnLargeur(A):**qui affiche un arbre en largeur

5. **def chercher(x,A)** :qui retourne **True** si **x** existe dans **A** ,**False** sinon
6. **def listeDenoeudsParNiveau(A,N) :** qui retourne une liste de Nœuds d'un niveau donnée.
7. **def parfaitComplet(A):** qui retourne **True** si l'arbre **A** est parfait complet, **False** sinon
8. **def parfait(A):** qui retourne **True** si l'arbre **A** est parfait ,**False** sinon
9. **def arbreDerecherche(A):** qui retourne **True** si l'arbre **A** est un arbre de recherche, **False** sinon
10. **def transformerListe(L,A,i=0) :**qui permet de transformer une liste simple vers un arbre binaire parfait .
11. **def transformerArbre(A,L) :**qui permet de transformer un arbre binaire parfait vers une liste simple .

Prof :M.El Ghmary

Exercice 02(La représentation d'un arbre sous forme de liste)

- Soit l'arbre binaire suivant :
- Mettez en œuvre les opérations suivantes :



1. **Insertion dans un arbre**
   Définir la fonction suivante:
   a) def inserer(A,info,pos): qui permet d'insérer l'information **info** dans l'arbre **A** à la position **pos**.
1. **Représenter l'arbre ci-dessus sous forme de listes**
2. **Affichage d'un arbre (parcours en profondeur)**
   Définir les fonctions suivantes :
   a) def inordre(A)
   b) def préordre(A)
   c) def postordre(A)
3. **Affichage d'un arbre (parcours en largeur)**

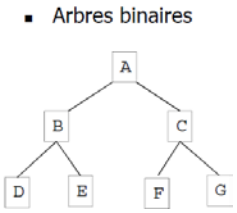
Définir les fonctions suivantes :

a) def niveau (A,N) :qui permet d'afficher le niveau **N**
b) def nombreDeNœudsParNiveau (A,n): qui retourne le nombre de Nœuds d'un niveau **N**
c) def nombreDeNiveaux (A): qui retourne le nombre de niveaux
d) def nombreDeNœuds (A): qui retourne le nombre de Nœuds dans un arbre
e) def affichageEnLargeur(A): qui affiche un arbre en largeur

4. **Recherche dans un arbre**
   def chercher(x,A) :qui retourne True si x existe dans A ,False sinon
5. **def transformerListe(L,A,i=0) :**qui permet de transformer une liste simple vers un arbre binaire parfait .

Exercice 03(La représentation d'un arbre sous forme de dictionnaire)

- Soit l'arbre binaire suivant :
- Mettez en œuvre les opérations suivantes :



1. **Insertion dans un arbre**
   Définir la fonction suivante:
   a) def inserer(A,info,g,d) : qui permet d'insérer un **Nœud** dans l'arbre **A**.
2. **Affichage d'un arbre (parcours en profondeur)**
   Définir les fonctions suivantes :
   a) def inordre(A,K)
   b) def préordre(A,K)
   c) def postordre(A,K)
3. **Affichage d'un arbre (parcours en largeur)**

Définir les fonctions suivantes :

a) def niveau (A,K,N) :qui permet d'afficher le niveau **N**
b) def nombreDeNœudsParNiveau (A,K,n): qui retourne le nombre de Nœuds d'un niveau **N**
c) def nombreDeNiveaux (A,K): qui retourne le nombre de niveaux
d) def nombreDeNœuds (A,K): qui retourne le nombre de Nœuds dans un arbre
e) def affichageEnLargeur(A,K): qui affiche un arbre en largeur

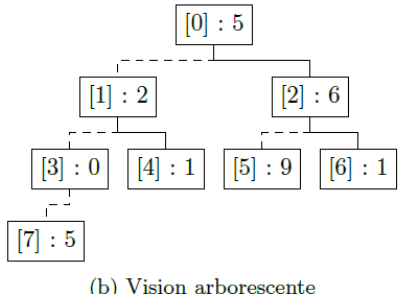
4. **def chercher(x,A,K) :**qui retourne True si x existe dans A ,False sinon
5. **def transformer(T,A) :**qui permet de transformer une liste simple vers un arbre binaire parfait

Exercice_04(tri par tas(maximier))

Le but de l'exercice est l'écriture d'un programme de tri de tableaux basé sur la notion de tas.
Les questions se suivent logiquement, mais beaucoup sont indépendantes.

indice	0	1	2	3	4	5	6	7
T[indice]	5	2	6	0	1	9	1	5

(a) Vision tabulaire



(b) Vision arborescente

Figure 1 - Deux vues différentes d'un même tableau

La **figure 1** montre qu'un même tableau peut être dessiné avec des cases contigües, ou bien avec des cases dispersées dans une arborescence. Avec la vue contigüe, on utilise généralement une variable i qui parcourt les indices du tableau. Avec la vue arborescente, on peut évidemment utiliser une variable i qui parcourt les indices du tableau, mais on utilise également trois fonctions qui permettent de suivre les liens bidirectionnels (réels ou virtuels) de l'arborescence :

- **gauche(indice)** représente les liens pointillé du haut vers le bas de l'arborescence.
  Par exemple, dans la figure 1b, gauche(1)=3, gauche(4)=9 et gauche(2)=5.
- **droite(indice)** représente les liens en trait plein du haut vers le bas de l'arborescence.
  Par exemple, dans la figure 1b, droite(1)=4, droite(3)=8 et droite(0)=2.
- **pere(indice)** représente les liens du bas vers le haut de l'arborescence.
  Par exemple, dans la figure 1b, pere(4)=1, pere(7)=3 et pere(2)=0.

1) Des fonctions élémentaires pour se déplacer dans un tableau.

- **a) def gauche (i) :** retourne un entier **g** tel qu'il existe un lien du haut vers le bas reliant les indices i à **g**.
- **b) def droite(i) :** retourne un entier d tel qu'il existe un lien du haut vers le bas reliant les indices i à d.
- **c) def pere(i):** retourne un entier **p** tel qu'il existe un lien du bas vers le haut reliant les indices i à p.

2) Construction d'un tas à partir d'un tableau.

- **a) def maximum(T,i,limite):** retourne l'indice (inferieur à limite) de la plus grande des trois valeurs T[i], T[gauche(i)] et T[droite(i)].
- **b) def entasser(T,i,limite):** échange des valeurs du tableau de haut en bas, en suivant une branche de l'arborescence. Cela a pour effet de faire descendre des petites valeurs, et de faire monter les grandes valeurs. Il est donc possible de construire un tas, en itérant cet algorithme sur les indices décroissants du tableau.
- **c) def construireTas(T):** transforme un tableau en un tas(En utilisant entasser(T,i,limite)).
- **d) def estunTas(T)** :qui retourne **True** si le tableau T est un tas, **False** sinon.
- **e) def trierTas(T):** permet de trier un tas
- **f) def triParTas(T):** permet de trier le tableau

Remarque :

- Un tas est un tableau d'entiers tel que pour tous les indices **i>0**, la valeur de **T[i] <= à celle de T[pere(i)]**
- Dans un tas, la valeur maximale est à la racine de l'arborescence, donc en T[0].
  Dans le tableau trié, cette valeur doit être en **T[len(T)-1]**. Il suffit donc d'échanger ces deux valeurs pour progresser vers la solution.
  Une fois cet échange fait, si l'on exclut la dernière valeur du tableau, le tas est peu changé.
  En fait **entasser(T,0,len(T)-1)** va créer un nouveau tas pour les valeurs du tableau dont les indices sont inferieurs à limite = len(T)-1. Il suffit donc d'itérer ces deux étapes **(échange, entasser)** pour trier un tas.