

CHAPITRE
<h1>5</h1>
<h1>ARBRES BINAIRES</h1>


Table des matières

1 Définitions et vocabulaire :	2
2 Relation entre taille et hauteur	3
3 Interface et implémentation possible	4
3.1 Taille et profondeur d'un arbre	5
3.2 Parcours d'un arbre	6
3.2.1 Parcours en profondeur(DFS)	6
3.2.2 Parcours en largeur(BFS)	7
4 Les arbres binaires de recherche	7
4.1 Recherche d'une valeur dans un ABR	8
4.2 Complexité de l'algorithme de recherche	8
4.3 Insertion dans un arbre binaire	9

Les listes, comme les *pires* et les *files* sont des données de stockage linéaires. Nous présentons ici une structure *hiérarchique* : les *arbres binaires*.

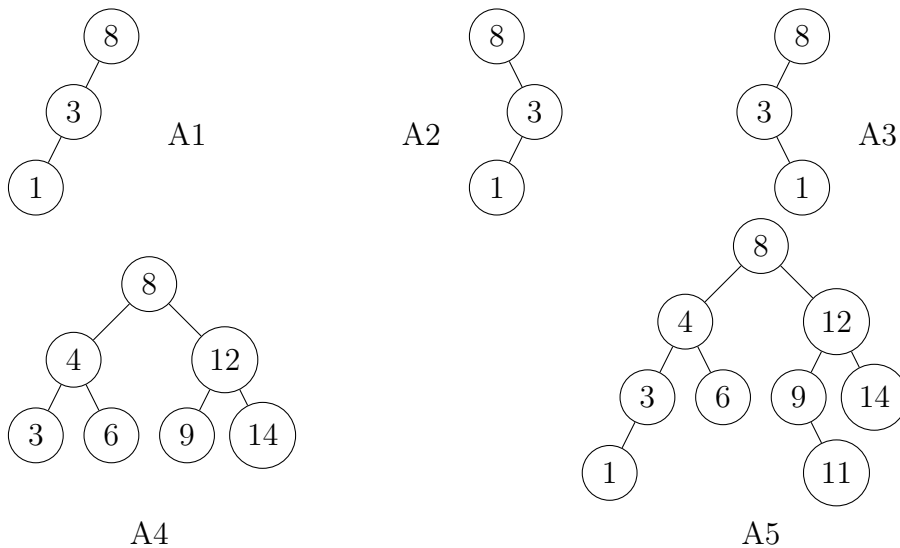
1 Définitions et vocabulaire :

Un arbre binaire est défini de la façon suivante : ou bien il est vide, ou bien il est constitué d'une *noeud* racine qui possède deux fils, un fils gauche et un fils droit, qui sont tous les deux des arbres binaires.



Il s'agit donc d'une définition récursive.

Ci-dessous, quelques exemples d'arbres :



Les trois premiers arbres sont distincts et mettent en avant la nécessité de distinguer fils droit et fils gauche.



Le nombre de noeuds d'un arbre binaire s'appelle sa *taille*.

Ainsi la taille des arbres ci-dessus est-elle de 3, 7 et 9 pour le dernier. On dit que la racine est à profondeur 0, ses fils à profondeur 1, etc...



La *profondeur maximale* d'un noeud d'un arbre s'appelle sa *hauteur*.

Les arbres ci-dessus sont de hauteur 2 ou 3. Par convention, la hauteur d'un arbre vide est -1 et celle d'un arbre réduit à sa racine, égale à 0.

On définit de plus la notion pertinente de *feuille* d'un arbre :



Un noeud qui ne possède pas d'enfant est appelé *feuille* de l'arbre.

Exercice n°1

Compléter le tableau ci-dessous qui donne le nombre de feuilles pour les arbres précédents.

Arbre	A_1	A_2	A_3	A_4	A_5
Nombre de feuilles					

Un arbre *complet* est un arbre qui à chaque profondeur, possède tous ses noeuds.

Exercice n°2

Déterminer la taille d'un arbre complet de hauteur h .

2 Relation entre taille et hauteur

Il s'agit à présent d'établir un encadrement de la hauteur h d'un arbre en fonction de sa taille n .

Exercice n°3

On s'appuiera sur des constructions d'arbres pour répondre aux questions suivantes :

- ① Pour une taille n donnée, quelle est la hauteur maximale que l'arbre peut atteindre ?
- ② Compléter à l'aide de dessins s'il le faut, le tableau suivant donnant la hauteur minimale d'un arbre binaire en fonction de sa taille :
- ③ D'après le tableau, comment évolue h à chaque fois qu'on double n ?
- ④ En déduire un encadrement de h en fonction de n .

n	1	2	3	4	5	8	9	15	20
h									

3 Interface et implémentation possible

Un arbre binaire c'est :

- ① soit un arbre vide Δ ;
- ② soit un triplet (e, g, d) , appelé noeud, avec e son étiquette, g son sous arbre binaire gauche et d son sous arbre binaire droit.

Exercice n°4

Construire les arbres binaires suivants :

- ① $(1, \Delta, \Delta)$
- ② $(3, (1, \Delta, (4, (1, \Delta, (5, \Delta, \Delta)), \Delta)), \Delta)$
- ③ $(3, (1, (1, \Delta, \Delta), \Delta), (4, (5, \Delta, \Delta), (9, \Delta, \Delta)))$
- ④ $(3, (1, (1, \Delta, \Delta), (5, \Delta, \Delta)), (4, (9, \Delta, \Delta), (2, \Delta, \Delta)))$

On définit le type abstrait **Arbre binaire** par ses interfaces (ceci est une possibilité et on en découvrira plusieurs dans ce cours) :

```
1 Constructeurs:
2 arbre_vide():()-> arbre binaire
3 noeud:(etiquette*arbre binaire*arbre binaire)-> arbre binaire
4 Sélecteurs:
5 etiquette: arbre binaire->etiquette
6 gauche: arbre binaire->arbre binaire
7 droite: arbre binaire->arbre binaire
8 Prédicat:
9 est_vide():arbre binaire->booléen
```

Ces fonctions abstraites définissent *l'interface* des arbres binaires et peuvent être implémentées de diverses façon en Python : en classe objet, par des listes ou encore des tuples. Nous le découvrirons dans les nombreux exercices proposés.

Exemple n°1

On choisit d'implémenter des arbres binaires à partir de listes python dans le paradigme fonctionnel.

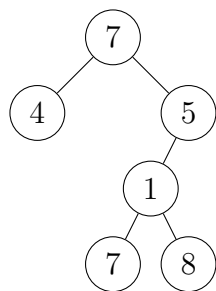
- ① Implémenter les primitives précédentes dans le langage Python.
- ② Dessiner l'arbre :
`noeud(7, noeud(4), noeud(5, noeud(1, noeud(7), noeud(8))))`.
- ③ Créer un arbre binaire complet de hauteur 2.

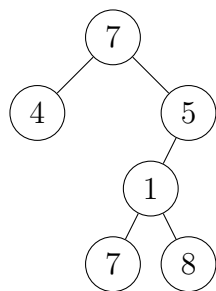
L'implémentation fonctionnelle pourrait donner ceci :

```

1 def arbre_vide():
2     return []
3 def noeud(etiquette, gauche = arbre_vide(), droit = arbre_vide()):
4     return [etiquette, gauche, droit]
5 def etiquette(arbre):
6     return arbre[0]
7 def gauche(arbre):
8     return arbre[1]
9 def droit(arbre):
10    return arbre[2]
11 def est_vide(arbre):
12    return arbre == arbre_vide()

```



L'arbre demandé est :  et la construction « à la main » de l'arbre complet de hauteur 2 *donc* de taille 7 serait :

```

1 a = noeud(7)
2 b = noeud(6)
3 c = noeud(5)
4 d = noeud(4)
5 arbre = noeud(1, noeud(2,d,c), noeud(3,b,a))

```

Exercice n°5

Proposez un code Python qui permet de construire un arbre complet de taille n.

Exercice n°6

Coder les fonctions `est_feuille` qui prend en paramètre un arbre binaire qui retourne un booléen indiquant si c'est arbre est une feuille puis la fonction `compte_feuilles` qui compte les feuilles d'un arbre binaire.

3.1 Taille et profondeur d'un arbre

Pour un arbre binaire, le calcul de sa taille doit être envisagé de façon récursive.

Exercice n°7

Pour un arbre binaire non vide, quelle relation existe t-il entre sa taille et la taille de ses fils?

On construit alors une fonction récursive python `taille` qui prend en paramètres un arbre binaire a et qui retourne sa taille.

Exercice n°8

Compléter la fonction suivante où a est un arbre binaire.

```
1 def taille(a):
2     if .....:
3         return 0
4     else:
5         return 1 + .....
```

De la même façon, vous pouvez tester la fonction `profondeur` qui détermine la profondeur d'un arbre binaire :

```
1 def profondeur(a):
2     if est_vide(a):
3         return -1
4     else:
5         return 1 + max(profondeur(gauche(a)),profondeur(droit(a)))
```

Exercice n°9

Enrichissez votre interface des arbres binaires en liste avec les deux nouvelles primitives précédentes et testez-les sur vos deux arbres de l'exemple ci-dessus.

3.2 Parcours d'un arbre

On distingue deux façons de parcourir un arbre.

3.2.1 Parcours en profondeur(DFS)

Dans le parcours en profondeur, on parcourt d'abord la racine de l'arbre, puis récursivement les fils gauche et droit. L'ordre dans lequel est fait ce traitement donne les trois parcours possibles :

préfixe	infixe	suffixe
racine -gauche-droit	gauche- racine -droit	gauche-droit- racine

Exercice n°10

Donner l'ordre des noeuds pour les trois parcours en profondeur des arbres A4 et A5 ci-dessus.

3.2.2 Parcours en largeur(BFS)

On parcourt de gauche à droite tous les noeuds de hauteur 0 , puis de hauteur 1, ... On stocke en général les noeuds dans une file. Le parcours en largeur d'abord est un parcours étage par étage (de haut en bas) et de gauche à droite.

Exercice n°11

Donner le parcours en largeur pour les arbres A4 et A5

4 Les arbres binaires de recherche

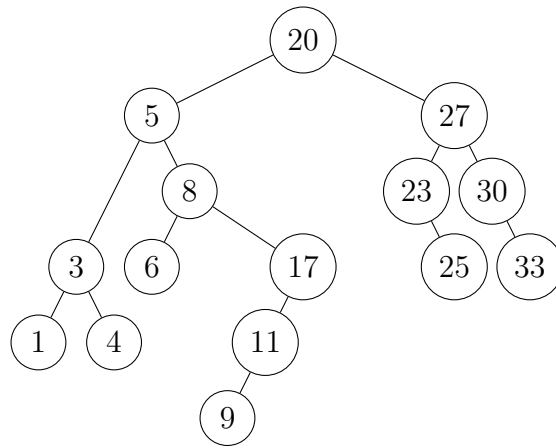
Un arbre binaire de recherche (ABR) est un cas particulier d'arbre binaire : c'est une structure de donnée qui permet de représenter un ensemble de valeurs qui dispose d'une relation d'ordre.

Dans un ABR, l'étiquette d'un noeud est appelée **clé** puis :

- les clés de tous les noeuds du sous arbre gauche d'un noeud X sont inférieures ou égales à la clé de X
- les clés de tous les noeuds du sous arbre droit d'un noeud X sont supérieures ou égales à la clé de X

Les opérations caractéristiques sur les arbres binaires de recherche sont la **recherche** d'une valeur, l'**insertion** d'une valeur ou sa **suppression**.

Exemple de ABR



4.1 Recherche d'une valeur dans un ABR

La recherche d'une valeur dans un ABR n'est pas très difficile : cela consiste à parcourir une branche en partant de la racine, en descendant sur le fils gauche ou sur le fils droit selon que la clé portée par le noeud est plus petite ou plus grande que la valeur recherchée. La recherche s'arrête dès la valeur trouvée ou que l'on atteint l'extrémité de la branche (la valeur n'est donc pas dans l'arbre...). L'algorithme s'écrit ainsi :

Entrée(s) a est un ABR et v une clé
Sortie(s) un booléen Vrai si $v \in a$ et Faux sinon
tant que NON estVide(a) ET $v \neq val(a)$ **faire**
 si $v < val(a)$ **alors**
 $a = \text{fils.gauche}(a)$
 sinon
 $a = \text{fils.droit}(a)$
 fin du si
fin du tant que
si estVide(a) **alors**
 retourner FAUX
sinon
 retourner VRAI
fin du si

Algorithme 1 : RECHERCHE DANS UN ARBRE BINAIRE DE RECHERCHE a D'UNE VALEUR v

4.2 Complexité de l'algorithme de recherche

Un arbre binaire est **complètement équilibré** si tous ses niveaux sont complets sauf éventuellement le dernier.

La recherche d'une clé dans un ABR équilibré est alors semblable à la recherche dans un tableau trié car on divise la zone de recherche par deux à chaque itération. La complexité est alors en $O(\log_2(n))$ où n est la taille de l'arbre. Si l'arbre n'est pas équilibré, la recherche sera moins efficace avec une complexité en $O(n)$ dans le pire des cas.

4.3 Insertion dans un arbre binaire

L'ajout d'un élément est un plus compliqué à programmer. L'idée est de commencer comme par la recherche et d'effectuer un ajout lorsqu'on arrive à un arbre vide (sauf si on tombe sur un doublon...). On propose ici une version récursive de l'algorithme.

Entrée(s) a est un ABR et v une clé

Sortie(s) v est insérée dans a

si $est_vide(a)$ **alors**

$a = creer_arbre(v, creer_arbre_vide(), creer_arbre_vide())$

sinon

si $v < val(a)$ **alors**

Insertion($v.fils.gauche(a)$)

sinon

si $v > val(a)$ **alors**

Insertion($v.fils.droit(a)$)

fin du si

fin du si

fin du si

Algorithme 2 : INSERTION DANS UN ARBRE BINAIRE DE RECHERCHE a D'UNE NOUVELLE CLÉ v