

Arbres

Fiche de synthèse

① VOCABULAIRE

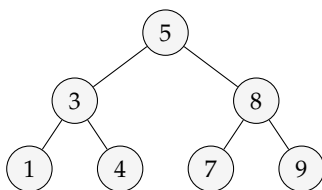
Définition 1 (Arbre) Un **arbre** est une structure hiérarchique formée de **nœuds** reliés par des **arêtes**, sans cycle : entre deux nœuds, il existe un seul chemin.

Exemples : l'arborescence des fichiers, un arbre généalogique, la structure d'une page HTML.

- La **racine** est l'unique nœud sans père ; une **feuille** est un nœud sans fils ; un **nœud interne** a au moins un fils.
- Le **père** d'un nœud est le nœud situé juste au-dessus ; ses **fils** sont les nœuds situés juste en dessous.
- Le **degré** d'un nœud est son nombre de fils.
- La **profondeur** d'un nœud est le nombre d'arêtes entre la racine et ce nœud ; la racine est à la profondeur 0.
- La **hauteur** d'un arbre est la plus grande profondeur de ses feuilles : 0 pour un arbre réduit à sa racine, -1 pour l'arbre vide.
- La **taille** d'un arbre est son nombre de nœuds.

② ARBRES BINAIRES

Définition 2 (Arbre binaire) Un **arbre binaire** est soit vide, soit formé d'une racine et de deux arbres binaires, son **sous-arbre gauche** et son **sous-arbre droit**. Chaque nœud a donc au plus deux fils.



Cet arbre a pour taille 7 et pour hauteur 2. Son interface comprend les opérations `est_vide`, `racine`, `gauche`, `droit` et `est_feuille`, qui ne s'appliquent qu'à un arbre non vide, sauf la première.

```

class Arbre:
    def __init__(self, valeur, gauche=None,
                 droit=None):
        self.valeur = valeur
        self.gauche = gauche
        self.droit = droit

    def est_feuille(self):
        return (self.gauche == None
                and self.droit == None)
  
```

```

a = Arbre(5, Arbre(3, Arbre(1), Arbre(4)),
          Arbre(8, Arbre(7), Arbre(9)))
  
```

L'arbre vide est représenté par `None`. Ainsi, `a.droit.gauche.valeur` vaut 7.

③ ALGORITHMES RÉCURSIFS

Un arbre binaire est défini à partir de sous-arbres : on traite l'arbre vide en cas de base, puis on combine les résultats des deux sous-arbres.

```

def taille(arbre):
    if arbre == None:
        return 0
    return (1 + taille(arbre.gauche)
            + taille(arbre.droit))

def hauteur(arbre):
    if arbre == None:
        return -1
    return 1 + max(hauteur(arbre.gauche),
                   hauteur(arbre.droit))
  
```

④ PARCOURS EN PROFONDEUR

Parcours	Ordre	Sur l'arbre ci-dessus
préfixe	nœud, gauche, droit	5, 3, 1, 4, 8, 7, 9
infixe	gauche, nœud, droit	1, 3, 4, 5, 7, 8, 9
suffixe	gauche, droit, nœud	1, 4, 3, 7, 9, 8, 5

```

def prefixe(arbre):
    if arbre != None:
        print(arbre.valeur)
        prefixe(arbre.gauche)
        prefixe(arbre.droit)
  
```

Pour les parcours infixe et suffixe, on déplace l'instruction `print` entre les deux appels récursifs, ou après eux.

⑤ PARCOURS EN LARGEUR

Le parcours en largeur visite les nœuds niveau par niveau, de gauche à droite : 5, 3, 8, 1, 4, 7, 9. Il utilise une **file** : premier entré, premier sorti.

```

def largeur(arbre):
    file = [arbre]
    while file != []:
        noeud = file.pop(0)
  
```

```

if noeud != None:
    print(noeud.valeur)
    file.append(noeud.gauche)
    file.append(noeud.droit)

```

⑥ ARBRES BINAIRES DE RECHERCHE

Définition 3 (Arbre binaire de recherche) Un **arbre binaire de recherche** (ABR) est un arbre binaire dans lequel, pour chaque nœud, les valeurs du sous-arbre gauche sont strictement inférieures à la valeur du nœud, et celles du sous-arbre droit strictement supérieures.

L'arbre de cette fiche est un ABR : son parcours infixe donne les valeurs dans l'ordre croissant, ce qui est vrai pour tout ABR.

```

def rechercher(arbre, x):
    if arbre == None:
        return False
    if x == arbre.valeur:
        return True
    elif x < arbre.valeur:
        return rechercher(arbre.gauche, x)
    else:
        return rechercher(arbre.droit, x)

```

La recherche suit un seul chemin de la racine vers une feuille : son coût dépend de la hauteur de l'arbre.

- Dans un arbre **équilibré**, les deux sous-arbres de chaque nœud ont des hauteurs proches ; la hauteur est de l'ordre de $\log_2 n$ et la recherche est en $O(\log n)$.
- Dans un arbre **dégénéré**, chaque nœud a au plus un fils, comme après l'insertion de valeurs déjà triées ; l'arbre se comporte comme une liste et la recherche est en $O(n)$.

📌 À retenir.

- La taille compte les nœuds ; la hauteur est la plus grande profondeur d'une feuille.
- Les algorithmes sur un arbre binaire sont récurifs : cas de l'arbre vide, puis les deux sous-arbres.
- Préfixe, infixe et suffixe sont des parcours en profondeur ; le parcours en largeur utilise une file.
- Le parcours infixe d'un ABR donne ses valeurs dans l'ordre croissant.
- La recherche dans un ABR est en $O(\log n)$ s'il est équilibré, en $O(n)$ s'il est dégénéré.