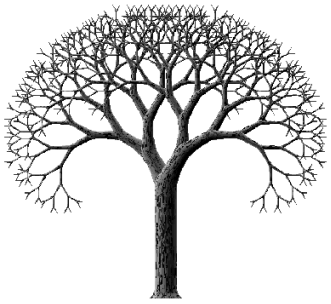
TNSI	Représentation des données Arbres binaires	Cours – **Arbres binaires et arborescences**

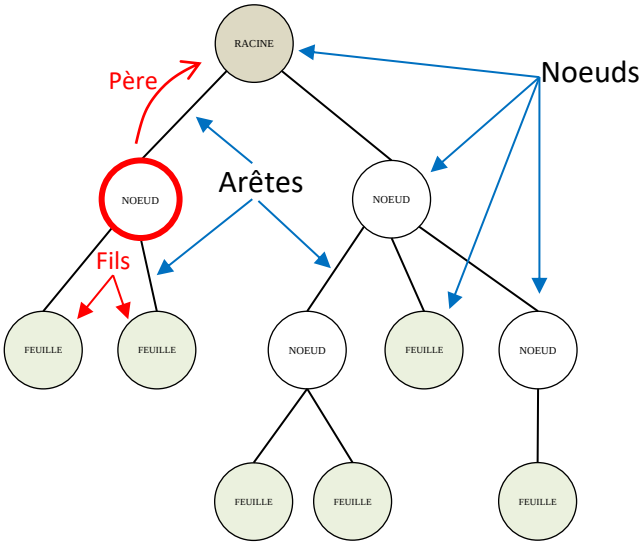
Objectifs :

- ⇨ Découvrir les concepts d'arbre
- ⇨ Décrire une structure d'arbre binaire
- ⇨ Découvrir les arbres binaires de recherche et leur intérêt



I – Les arbres

Les **arbres** sont des structures informatiques permettant de représenter l'information sous forme hiérarchisée.



Un arbre est constitué de nœuds reliés entre eux par des arêtes. Selon une relation père-fils.

Les « arbres » informatiques poussent vers le bas.

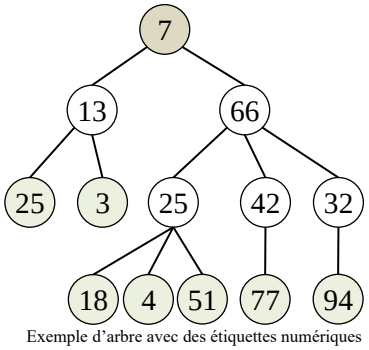
On distingue trois types de nœuds :

- La **racine** de l'arbre est le nœud situé tout en haut et ne possédant pas de parent.
- les **feuilles** (ou nœuds externes), éléments ne possédant pas de fils dans l'arbre ;
- les nœuds internes, éléments possédant des fils (sous-branches).

Chaque nœud (y-compris la racine) peut posséder 0, 1, ou plusieurs **fils** qui lui sont reliés et se situent en-dessous de lui.

Chaque nœud (sauf la racine) est relié en amont a un seul nœud appelé **père** (ou **nœud parent**).

Les nœuds possèdent en général une **étiquette** porteuse d'information (par exemple un nombre, une chaîne de caractère ou n'importe quel objet).



Exemple d'arbre avec des étiquettes numériques

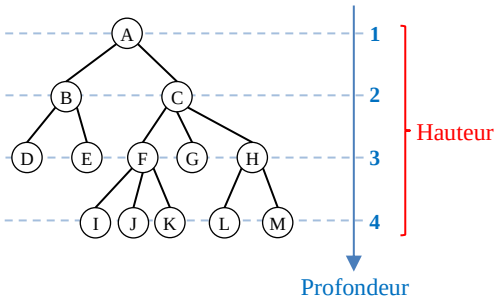
Remarques :

- Les étiquettes portent aussi le nom de **clés** pour certains auteurs.
- Nous ne considérerons ici que des arbres **homogènes**, c'est-à-dire où toutes les étiquettes sont du même type (par exemple que des nombres ou que des lettres, mais pas de mélange).

La **profondeur** d'un nœud est la distance (nombre d'arêtes) de la racine au nœud.

La **hauteur** d'un arbre est la plus grande profondeur d'une feuille de l'arbre.

La **taille** d'un arbre est son nombre de nœuds, la **longueur de cheminement** est la somme des profondeurs de chacune des feuilles.



Dans l'exemple ci-contre, l'arbre a une hauteur de 4 et une taille de 13.
La profondeur du nœud E est 3 et celle du nœud K est 4.

Remarque importante :
La définition de la profondeur et de la hauteur ne fait pas l'unanimité, et certains auteurs comptent la profondeur à partir de 0 (la racine a une profondeur de 0, ses fils directs de 1, …). Selon cette définition l'arbre de l'exemple précédent aurait une hauteur de 3 et la profondeur du nœud E serait de 2.

On peut aussi définir un arbre de la façon suivante :

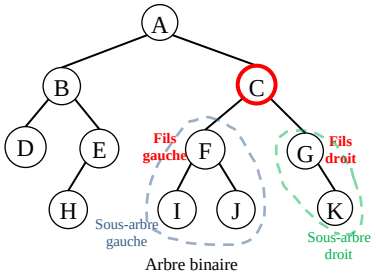
Ou bien l'arbre est vide, ou bien il est constitué d'un nœud racine qui possède éventuellement un ou plusieurs fils, qui sont eux-mêmes des arbres.

Une telle définition montre que les arbres, comme les nœuds des listes chaînées, sont intrinsèquement récursifs. On utilisera donc souvent des algorithmes récursifs lorsqu'on manipule des arbres.

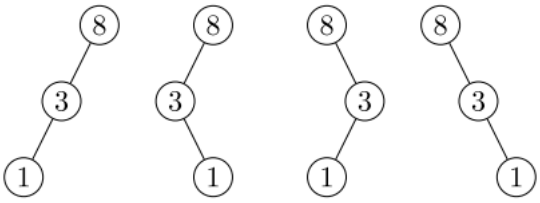
II - Arbres binaires

1) Définition

Les **arbres binaires** sont des arbres dont les nœuds possèdent au plus deux fils ordonnés : un **fils gauche** et un **fils droit**.

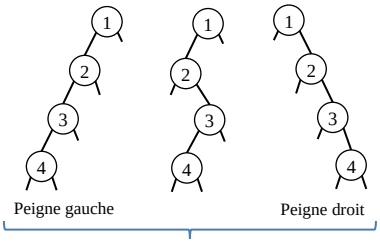


Il ne faut pas confondre fils gauche et fils droit, ainsi les arbres suivants ne sont pas les mêmes :

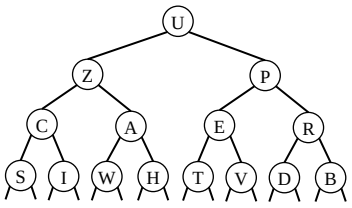


2) Forme

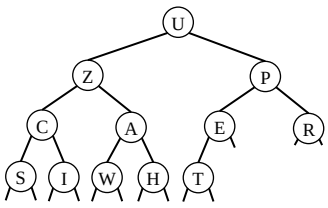
Il est possible d'avoir des arbres de formes variées :



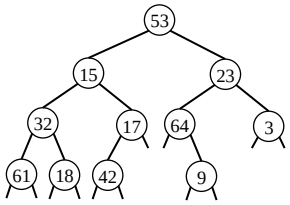
Arbres dégénérés ou filiforme :
Tous les nœuds ne possèdent qu'un seul fils (sauf les feuilles)
Un tel arbre est équivalent à une liste chaînée.



Arbres parfaits :
Tous les nœuds possèdent 2 fils, sauf les feuilles qui sont toutes à la même profondeur



Arbres complets ou presque complets :
Tous les niveaux sont complètement remplis, sauf éventuellement le dernier.
Les feuilles du dernier niveau sont le plus à gauche possible.



Arbres équilibrés :
Pour chaque nœud les hauteurs des sous arbres gauches et droit différent au plus de 1.

3) Encadrement de la taille

Pour un arbre binaire de hauteur h, la taille n de l'arbre est toujours dans l'intervalle :

$$h \leq n < 2^h$$

h = n dans le cas particulier d'un arbre filiforme et $n = 2^h - 1$ pour un arbre parfait

On peut aussi écrire $\log_2(n) < h \leq n$

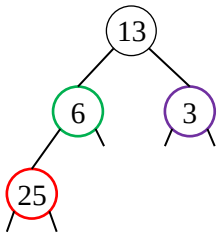
4) Implémentation en python

Noeud
Attributs `valeur` (objet de type quelconque) `fils_gauche` : Nœud `fils_droit` : Noeud
Méthodes `__init__(fils_g,valeur,fils_d)`

Comme nous l'avons fait pour les listes chaînées, on peut partir d'une classe `Noeud` qui représentera un nœud d'arbre binaire :

```
class Noeud():
    """Classe représentant le noeud d'un arbre binaire"""
    def __init__(self, fils_gauche, valeur, fils_droit):
        self.fils_gauche = fils_gauche
        self.valeur = valeur
        self.fils_droit = fils_droit
```

On peut alors créer un arbre comme une série de nœuds. Par exemple pour créer l'arbre ci-contre, on utilisera l'instruction :




```
a1 = Noeud(Noeud(Noeud(None,25,None),6,None),13,Noeud(None,3,None))
```

Par la suite, pour simplifier les tests, on pourra récupérer le fichier « affiche_arbre.py » et l'importer avec :

```
from affiche_arbre import affiche_arbre, chaine_representation_arbre
```

On peut alors utiliser la procédure `affiche_arbre(n)` qui affiche l'arbre à partir du nœud `n` ou `chaine_représentation_arbre(n)` qui renvoie la chaine représentant l'arbre.

Application 1 :

1) Ecrire une fonction `taille` qui prend en argument un nœud et renvoie la taille de l'arbre dont ce nœud est la racine.
2) Ecrire une fonction `hauteur` qui prend en argument un nœud et renvoie la hauteur de l'arbre dont ce nœud est la racine.

Pour tester vos fonctions, vous pouvez vérifier que l'arbre

```
a2 = Noeud(Noeud(Noeud(Noeud(Noeud(Noeud(None,"A",None),"H",Noeud(None,"U",
     Noeud(None,"R",None))),"D",Noeud(Noeud(None,"O",None),"Q",None)),"Q",None),
     "L",Noeud(Noeud(None,"Y",Noeud(None,"B",None)),"I",None)),"G",
     Noeud(Noeud(Noeud(None,"E",Noeud(None,"V",None)),"W",Noeud(Noeud(None,"N",
     None),"O",Noeud(None,"F",None))),"S",Noeud(Noeud(None,"C",None),"X",None)))
```

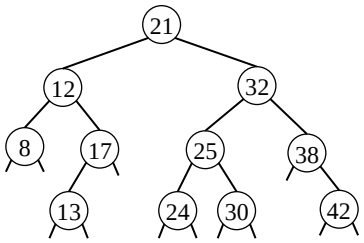
a une taille de 22 et une hauteur de 7

III – Les arbres binaires de recherche (ABR)

Les arbres binaires permettent de faire des recherches très rapides (en O(log(n))) tout en garantissant une insertion et une suppression des éléments en temps logarithmique également.

Un **arbre binaire de recherche** (ABR) est un arbre binaire pour lequel la condition suivante est respectée pour tous les nœuds :

La valeur du nœud est supérieure (ou égale) à toutes celles de son sous-arbre gauche et inférieure (ou égale) à toutes celles de son sous-arbre droit.



Arbre binaire de recherche

C'est cette propriété qui rend la recherche et l'insertion dans un ABR particulièrement efficace car pour chercher une valeur on n'a jamais plus d'étapes que la hauteur de l'arbre.

IV – Arborescences

Dans de nombreux cas, on a besoin d'une structure arborescente où chaque nœud peut avoir plus de deux fils comme par exemple pour l'arborescences des fichiers sur un disque dur.

Les fichiers XML (comme le HTML) ou JSON sont d'autres exemples d'arborescences.

Application 2 :

1) Quelle structure de donnée peut-on utiliser pour représenter une arborescence ?
2) Ecrire une classe Noeud implémentant l'interface ci-dessous pour représenter une arborescence.

Fonction	Description
constructeur(*val*, *fils*) → Noeud	Créer un arbre dont la racine a pour étiquette *val* et dont les fils sont dans la liste *fils* (on peut donner la valeur par défaut None à ces arguments)
fils() → liste de Nœud	Renvoie un tableau de tous les fils du nœud
est_feuille() → Booléen	Renvoie Vrai si le nœud est une feuille
ajoute_fils(*f*)	Ajoute le nœud *f* comme fils du nœud

3) Quel attribut rajouter à la classe Noeud pour qu'on puisse facilement remonter d'un cran dans l'arborescence (comme la commande `cd ..` en invite de commande) ? Ajouter cet attribut à la classe et modifier ses méthodes en conséquence.

Références :

Arbres binaires et algorithmes : https://www.lri.fr/~fiorenzi/Teaching/AL/C4.pdf, https://pixees.fr/informatiquelycee/n_site/nsi_term_algo_arbre.html
Différentes façons de représenter les arbres binaires : http://math.univ-lyon1.fr/irem/Formation_ISN/formation_recursivite/arborescence/arbres.html

Arbres et arborescences : http://si.nsi.free.fr/TNSI/Cours/Chapitre9.php
Arbres et arbres binaires : https://info.blaisepascal.fr/nsi-arbres
Application des arbres binaires à la création de labyrinthes : https://hurna.io/fr/academy/algorithms/maze_generator/binary.html

Exercices – **Arbres binaires et arborescences**

Exercice 1 :

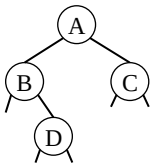
Dessiner tous les arbres binaires ayant respectivement 3 et 4 nœuds.

Exercice 2 :

Sachant qu'il y a 1 arbre binaire vide, 1 arbre binaire contenant 1 nœud, 2 arbres contenant 2 nœuds, 5 arbres contenant 3 nœuds et 14 arbres contenant 4 nœuds, calculer le nombre d'arbres binaires contenant 5 nœuds. On ne cherchera pas à les construire tous, mais seulement à les dénombrer.

Exercice 3 : affichage

Ecrire une fonction `affiche(n)` qui affiche un arbre dont on donne le nœud racine `n` sous la forme suivante : pour un arbre vide, on n'affiche rien ; pour un nœud, on affiche une parenthèse ouvrante, son sous-arbre gauche (récursivement), sa valeur, son sous-arbre droit (récursivement), puis enfin une parenthèse fermante.
Ainsi l'arbre représenté ci-contre s'écrira : `((B(D)A(C))`



Exercice 4 :

Dessiner l'arbre binaire dont la représentation par la fonction `affichage` de l'exercice précédent donne : `(1((2)3))`.

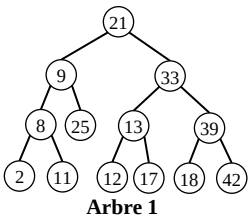
Exercice 5 : classe `Noeud`

On fait le choix d'utiliser la classe `Noeud` pour représenter un arbre (ou sous-arbre) et on va l'étendre afin de lui ajouter un certain nombre de méthodes utiles. On déclarera un arbre vide par `Noeud(None, None, None)` (la valeur et les fils gauche et droit sont à `None`).

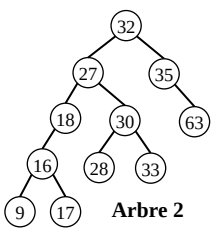
1) Ajouter une méthode `est_vide()` qui renvoie `True` si l'arbre est vide et `False` sinon.
2) Ajouter les méthodes `taille()` et `hauteur()` qui renvoient respectivement la taille et la hauteur de l'arbre.
3) Ajouter une méthode `__str__` pour renvoyer une chaîne représentant l'arbre en utilisant les fonctions du fichier « affiche_arbre.py », puis une méthode `__repr__` qui renvoie la représentation plus compacte faite à l'exercice 3.
4) Ajouter une méthode `greffe_gauche(n)` qui teste si le fils gauche de l'arbre existe et si ce n'est pas le cas y attache le nœud `n` passé en argument (ne fait rien s'il y a déjà un sous-arbre gauche). Même question avec une méthode `greffe_droit(n)`.
5) Ajouter une méthode `__eq__` qui teste l'égalité de deux arbres. Deux arbres sont égaux s'ils ont exactement les mêmes valeurs situées dans les mêmes branches (bien faire la distinction entre fils gauche et fils droit).

Exercice 6 :

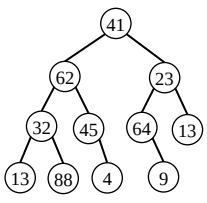
Parmi les arbres suivants, lesquels sont des arbres binaires de recherche ?



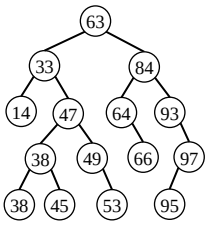
Arbre 1



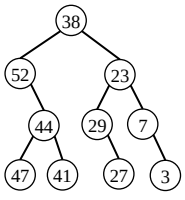
Arbre 2



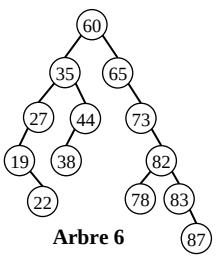
Arbre 3



Arbre 4



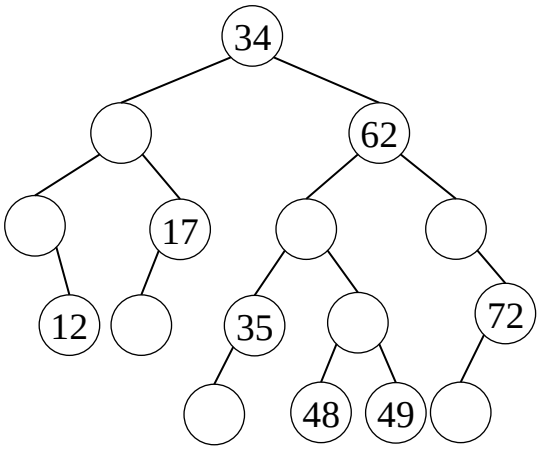
Arbre 5



Arbre 6

Exercice 7 :

Compléter l'arbre ci-contre pour que ce soit un ABR.



Exercice 8 :

Dessiner tous les arbres binaires de recherche formés de 3 nœuds et comportant les nombres 1, 2 et 3.

Exercice 9 : un étiquetage intéressant

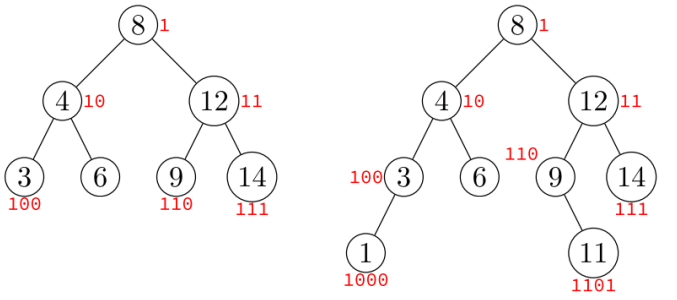
On peut étiqueter les nœuds d'un arbre de la façon suivante :

- la racine porte l'étiquette 1;
- si un nœud quelconque porte l'étiquette x...yz, la racine de son éventuel fils gauche porte l'étiquette x...yz**0** et la racine de son éventuel fils droit l'étiquette x...yz**1**.

On observe que les étiquettes des nœuds de profondeur p sont constituées de p + 1 bits, et sont deux à deux distinctes. On en déduit que toutes les étiquettes des nœuds d'un même arbre sont deux à deux distinctes.

Les étiquettes peuvent être vues comme les écritures de numéros en base 2 : on a ainsi numéroté les différents nœuds de l'arbre.

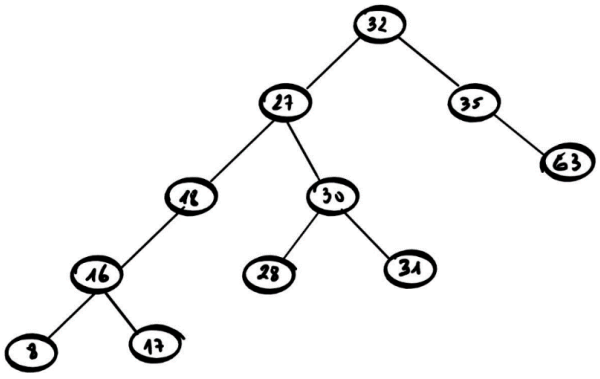
Voici le résultat de cet étiquetage sur les deux arbres :



1) Quelle est l'étiquette du nœud 6 ?

Dans le cas de l'arbre de gauche, si on traduit en décimal les étiquettes des nœuds, on constate qu'on les a simplement numérotés de 1 à 7, de haut en bas et de gauche à droite.

2) Etiqueter suivant la même méthode l'arbre ci-contre.



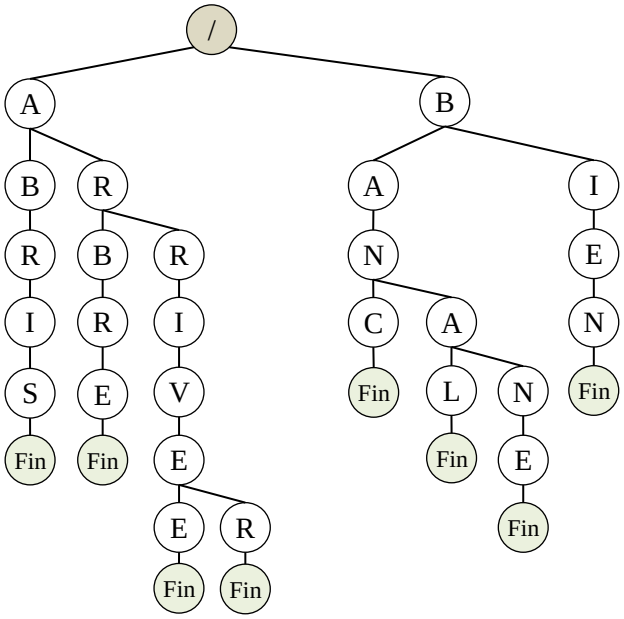
Exercice 10 : minimum d'un ABR

1) Dans un ABR, où se trouve le plus petit élément ?
2) Ecrire une fonction `minimum(n)` qui renvoie la valeur minimale des clés contenus dans l'arbre dont on donne le nœud racine `n`.

Exercice 11 : estABR

1) Ecrire une fonction `min_ss_arbre(n)` qui renvoie la plus petite valeur du sous-arbre quelconque (pas forcément ABR) dont on donne la racine `n`.
2) Même question pour une fonction `max_ss_arbre(n)` qui renvoie la plus grande valeur.
3) Ecrire une fonction `estABR(n)` qui renvoie `True` si l'arbre de racine `n` est un ABR. On pourra se servir des fonctions précédentes.

Exercice 12 : Arbre lexical



On utilise un arbre pour stocker des mots d'un lexique. La racine est le point d'entrée du lexique mais ne contient aucune lettre, puis ses fils sont la première lettre du mot, les fils de chaque fils de la racine, la deuxième lettre, etc…
La fin d'un mot est indiquée par une feuille ayant l'étiquette « Fin ».
Ainsi l'arbre ci-contre stocke les mots ARBRE, ABRIS, ARRIVEE, ARRIVER, BANC, BANAL, BANANE et BIEN.

1) Représenter l'insertion dans l'arbre des mots « BANCAL » et « BISE ».

Pour les fonctions suivantes, on utilisera la classe Noeud des arborescences programmée à la fin du cours. Afin de réaliser des tests, on donne la définition de l'arbre donné ci-contre :

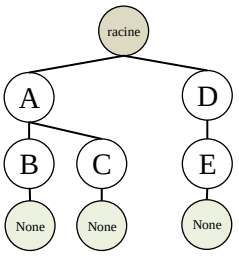
```
a = Noeud("/",[Noeud("A",[Noeud("B",[Noeud("R",[Noeud("I",[Noeud("S",[Noeud(None)])])])]),
    Noeud("R", [Noeud("B",[Noeud("R",[Noeud("E",   [Noeud(None)])])]),Noeud("R",[Noeud("I",[Noeud("V",[Noeud("E",
    [Noeud("E",[Noeud(None)]),Noeud("R",[Noeud(None)])])])])])]),Noeud("B",[Noeud("A",[Noeud("N",[Noeud("C",
    [Noeud(None)]),Noeud("A",[Noeud("L",[Noeud(None)]),Noeud("N",[Noeud("E",[Noeud(None)])])])])]),Noeud("I",
    [Noeud("E",[Noeud("N",[Noeud(None)])])])]))
```

L'étiquette de fin sera représentée par une valeur de `None` pour l'implémentation en python.

2) Ecrire une fonction `rechercher_mot(n, mot)` qui recherche la chaîne `mot` dans l'arbre dont on donne la racine `n` et renvoie `True` si le mot est présent dans l'arbre et `False` sinon.
3) Ecrire une procédure `affiche_arborescence(n)` qui affiche un arbre en parcourant ses nœuds et en affichant pour chaque nœud sa valeur précédée d'un nombre d'espace égal à sa profondeur.

L'arbre représenté ci-contre s'afficherait donc :

```
racine
 A
  B
   None
  C
   None
 D
  E
   None
```



Vous pouvez tester votre fonction en utilisant la définition de l'arbre précédent donnée ci-dessous :

```
b = Noeud("racine",[Noeud("A",[Noeud("B",[Noeud(None)]),Noeud("C",[Noeud(None)])]),Noeud("D",[Noeud("E",[Noeud(None)])])])
```

4) Ecrire une fonction `insertion(n, mot)` qui insère la chaine `mot` dans l'arbre dont on fournit la racine `n`.
5) Quel est l'intérêt d'une telle structure ?