STRUCTURES DE DONNEES : LES ARBRES

I- NOTION D'ARBRE

Nous allons travailler sur la structure de donnée "arbre". Cette structure n'est pas linéaire mais **hiérarchique**.

Un organisateur de tournoi de rugby recherche la meilleure solution pour afficher les potentiels quarts de final, demi-finales et finale :

Au départ nous avons 4 poules de 4 équipes. Les 4 équipes d'une poule s'affrontent dans un mini championnat (3 matchs par équipe). À l'issue de cette phase de poule, les 2 premières équipes de chaque poule sont qualifiées pour les quarts de finale.

Dans ce qui suit, on désigne les 2 qualifiés par poule par :

- Poule 1 => 1er Eq1 ; 2e Eq8
- Poule 2 => 1er Eq2 ; 2e Eq7
- Poule 3 => 1er Eq3 ; 2e Eq6
- Poule 4 => 1er Eq4 ; 2e Eq5

En quart de final on va avoir :

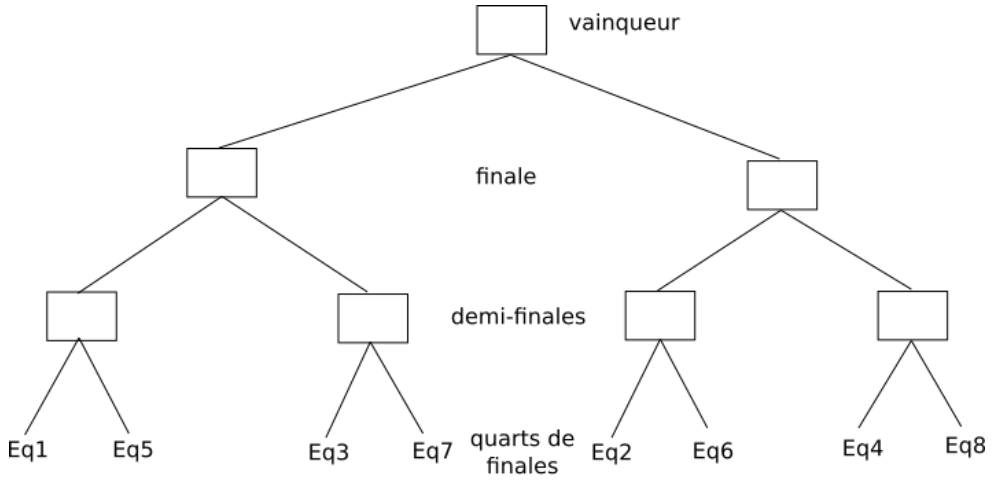
- quart de finale 1 => Eq1 contre Eq5
- quart de finale 2 => Eq2 contre Eq6
- quart de finale 3 => Eq3 contre Eq7
- quart de finale 4 => Eq4 contre Eq8

Pour les demi-finales on aura :

- demi-final 1 => vainqueur quart de finale 1 contre vainqueur quart de finale 3
- demi-final 2 => vainqueur quart de finale 2 contre vainqueur quart de finale 4

L'organisateur du tournoi affiche les informations ci-dessus le jour du tournoi. Malheureusement, la plupart des spectateurs se perdent quand ils cherchent à déterminer les potentielles demi-finales (et ne parlons pas de la finale !)

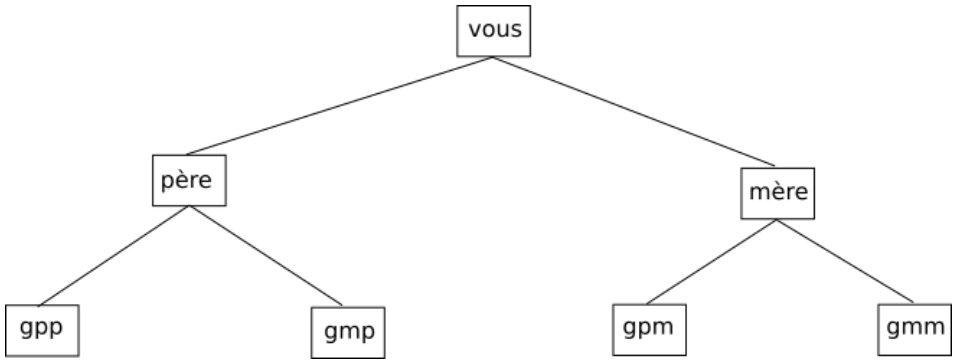
Pourtant, un simple graphique aurait grandement simplifié les choses :



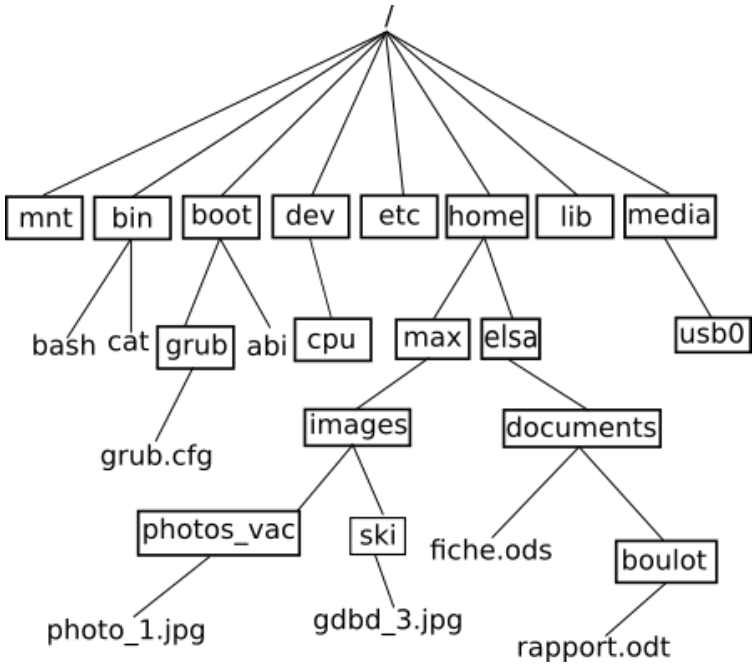
Poule 1 => 1er Eq1 ; 2e Eq8
Poule 2 => 1er Eq2 ; 2e Eq7
Poule 3 => 1er Eq3 ; 2e Eq6
Poule 4 => 1er Eq4 ; 2e Eq5

Les spectateurs peuvent alors recopier sur un bout de papier ce schéma et ensuite se livrer au jeu des pronostics.

Nous avons ci-dessous ce que l'on appelle une structure en arbre. On peut aussi retrouver cette même structure dans un arbre généalogique :



Dernier exemple, les systèmes de fichiers dans les systèmes de type UNIX ont aussi une structure en arbre



Système de fichiers de type UNIX

Définition :

Un arbre est une structure de données constituée de **nœuds**.

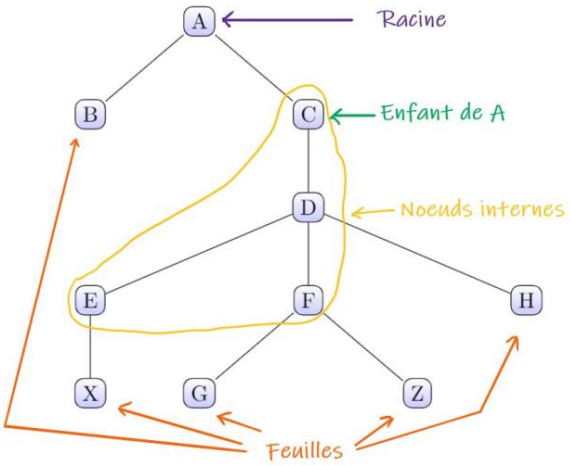
Le sommet de l'arbre s'appelle la **racine**.

Le nœud B situé sous un nœud A est appelé **enfant** du nœud A

Un nœud qui ne possède pas d'enfant est appelé **feuille**.

Les nœuds autre que la racine et les feuilles sont appelés **nœuds internes**.

Une branche est une suite de nœuds consécutifs de la racine vers une feuille.



Dans cet arbre:

- il y a 10 nœuds.
- Le nœud A est la racine.

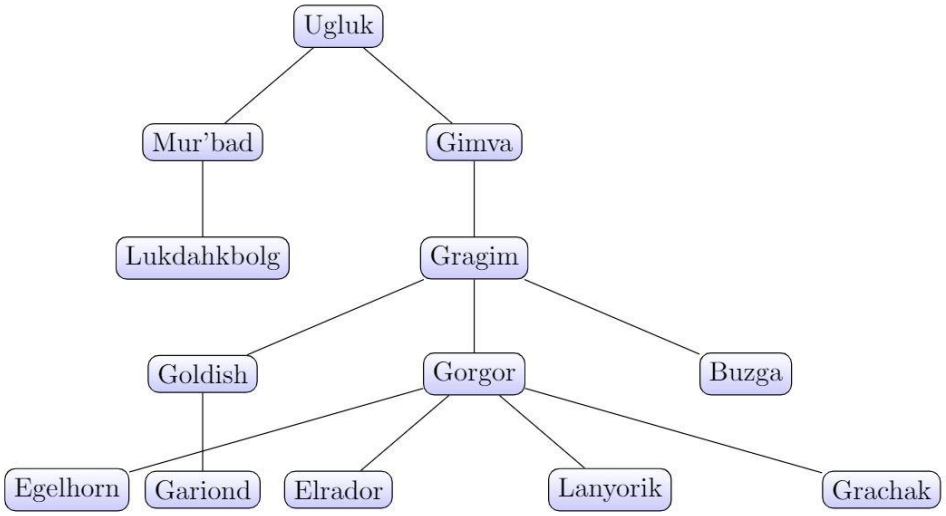
- Il y a 5 feuilles donc 5 branches.
- Le nœud D est l'enfant du nœud C.
- Il y a 4 nœuds internes.

Les arbres sont des types abstraits très utilisés en informatique. On les utilise notamment quand on a besoin d'une structure hiérarchique des données : dans l'exemple ci-dessous le fichier grub.cfg ne se trouve pas au même niveau que le fichier rapport.odt (le fichier grub.cfg se trouve "plus proche" de la racine / que le fichier rapport.odt). On ne pourrait pas avec une simple liste qui contiendrait les noms des fichiers et des répertoires, rendre compte de cette hiérarchie (plus ou moins "proche" de la racine).

On trouve souvent dans cette hiérarchie une notion de temps (les quarts de finale sont avant les demi-finales ou encore votre grand-mère paternelle est née avant votre père), mais ce n'est pas une obligation (voir l'arborescence du système de fichiers).

Activité 1

On donne l'arbre suivant :



1. Quelle est la racine de cet arbre ?
2. Combien y-a-t-il de nœuds ?
3. Combien y-a-t-il de feuilles ?
4. Combien y-a-t-il de branches ?
5. L'ensemble des nœuds internes compte combien d'éléments ?
6. Quels sont les enfants de Gragim ?

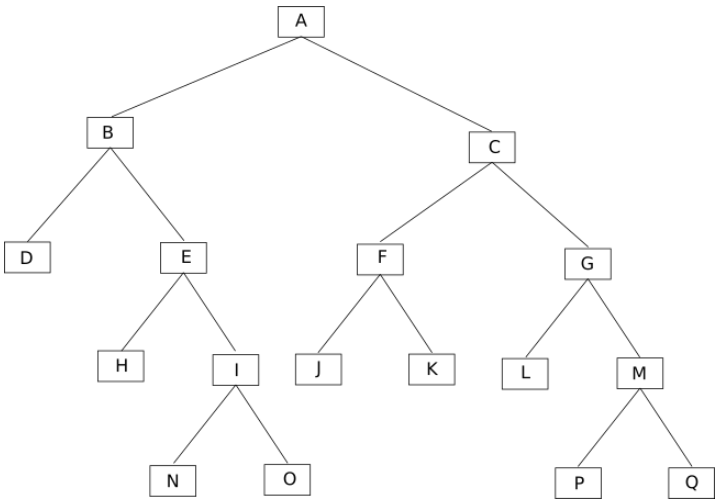
II- LES ARBRES BINAIRES

a) INTRODUCTION

Les arbres binaires sont des cas particuliers d'arbre : l'arbre du tournoi de rugby et l'arbre "père, mère..." sont des arbres binaires, en revanche, l'arbre représentant la structure du système de fichier n'est pas un arbre binaire. Dans un arbre binaire, on a au maximum 2 **arête** qui partent d'un élément (pour le système de fichiers, on a 7 branches qui partent de la racine : ce n'est donc pas un arbre binaire).

Dans la suite nous allons uniquement travailler sur les arbres binaires.

Soit l'arbre binaire suivant :



b) UN PEU DE VOCABULAIRE

- chaque élément de l'arbre est appelé **nœud** (par exemple : A, B, C, D,...,P et Q sont des nœuds)
- le nœud initial (A) est appelé **nœud racine** ou plus simplement **racine**
- On dira que le nœud E et le nœud D sont **les fils** du nœud B. On dira que le nœud B est le père des nœuds E et D
- Dans un arbre binaire, un nœud possède au plus 2 **fils**
- Un nœud n'ayant aucun **fils** est appelé **feuille** (exemples : D, H, N, O, J, K, L, P et Q sont des feuilles)
- À partir d'un nœud (qui n'est pas une feuille), on peut définir un **sous-arbre gauche** et un **sous-arbre droite** (exemple : à partir de C on va trouver un sous-arbre gauche composé des nœuds F, J et K et un sous-arbre droit composé des nœuds G, L, M, P et Q)
- On appelle **arête** le segment qui relie 2 nœuds.
- On appelle **taille** d'un arbre le nombre de nœuds présents dans cet arbre

- On appelle **profondeur** d'un nœud ou d'une feuille dans un arbre binaire le nombre de nœuds du chemin qui va de la racine à ce nœud. La racine d'un arbre est à une profondeur 1, et la profondeur d'un nœud est égale à la profondeur de son prédécesseur plus 1. Si un nœud est à une profondeur p, tous ses successeurs sont à une profondeur p+1.

  Exemples : profondeur de B = 2 ; profondeur de I = 4 ; profondeur de P = 5

  ATTENTION : on trouve aussi dans certains livres la profondeur de la racine égale à 0 (on trouve alors : profondeur de B = 1 ; profondeur de I = 3 ; profondeur de P = 4). Les 2 définitions sont valables, il faut juste préciser si vous considérez que la profondeur de la racine est de 1 ou de 0.

- On appelle **hauteur** d'un arbre la profondeur maximale des nœuds de l'arbre.

  Exemple : la profondeur de P = 5, c'est un des nœuds les plus profonds, donc la hauteur de l'arbre est de 5.

ATTENTION : comme on trouve 2 définitions pour la profondeur, on peut trouver 2 résultats différents pour la hauteur : si on considère la profondeur de la racine égale à 1, on aura bien une hauteur de 5, mais si l'on considère que la profondeur de la racine est de 0, on aura alors une hauteur de 4.

!!! Ainsi un *arbre vide* a pour hauteur 0.

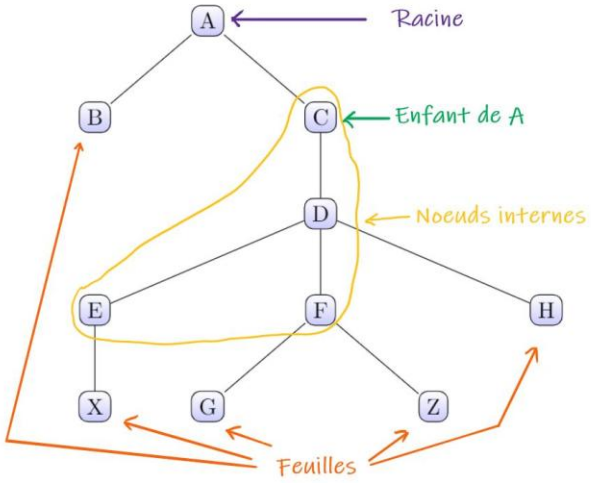
Définition	Hauteur de l'arbre avec un nœud	Hauteur de l'arbre vide
Par le nombre d'arêtes	0	−1
Par le nombre de nœuds	1	0

On peut caractériser un arbre par différentes caractéristiques :

- Son **arité** : le nombre maximal d'enfants qu'un nœud peut avoir.

Exemple :

On reprend l'arbre de la définition d'un arbre :



- L'arité de cet arbre est 3.
- La taille de cet arbre est 10.
- La hauteur de cet arbre est 4.

Remarque : On appelle **arbre binaire** un arbre d'arité inférieure ou égale à 2.

c) STRUCTURE RECURSIVE

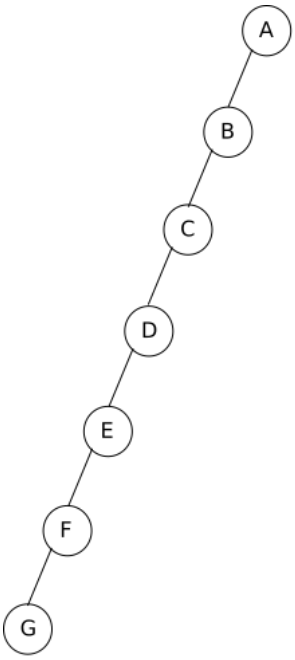
Il est aussi important de bien noter que l'on peut aussi voir les arbres comme des structures récursives : les fils d'un nœud sont des arbres (sous-arbre gauche et un sous-arbre droite dans le cas d'un arbre binaire), ces arbres sont eux-mêmes constitués d'arbres.

On peut alors définir un **arbre binaire** de façon *récursive* :

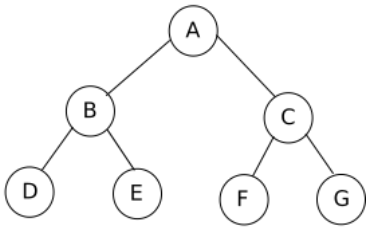
- soit c'est **l'arbre vide** s'il ne contient aucun noeud
- soit il est constitué d'un noeud spécial appelé **racine** de l'arbre dont le *fils droit* et le *fils gauche* sont des **arbres binaires**

d) ENCADREMENT DE LA HAUTEUR D'UN ARBRE

Il est possible d'avoir des arbres binaires de même taille mais de "forme" très différente :



Arbre 1



Arbre 2

Sur le schéma ci-dessus l'arbre 1 est dit " **filiforme**" alors que l'arbre 2 est dit "**complet**" (on dira qu'un arbre binaire est complet si tous les nœuds possèdent 2 fils et que toutes les feuilles se situent à la même profondeur). On pourra aussi dire que l'arbre 1 est **déséquilibré** alors que l'arbre 2 est équilibré.

Si on prend un arbre filiforme de taille n, on peut dire que la hauteur de cet arbre est égale à $n-1$ (si on prend la définition de la hauteur d'un arbre où la racine a une profondeur 0)
Si on prend un arbre complet de taille n, on peut démontrer que la hauteur de cet arbre est égale à la partie entière de $log_2(n)$ (on arrondit à l'entier immédiatement inférieur le $log_2(n)$)). Dans le cas de l'arbre 2,
Nous avons $log_2(7) = 2{,}8$ donc en prenant la partie entière on a bien la hauteur de l'arbre 2 égale à 2.

Un arbre filiforme et un arbre complet étant deux cas extrêmes, on peut affirmer que pour un arbre binaire quelconque

$$\lfloor log_2(n) \rfloor \leq h \leq n-1$$

Avec n la taille de l'arbre et h la hauteur de l'arbre $\lfloor log_2(n) \rfloor$ permet de prendre la partie entière du logarithme base 2 de n)

e) LES ARBRES BINAIRES EN PYTHON

Python ne propose pas de façon native l'implémentation des arbres binaires. Mais nous aurons, plus tard dans l'année, l'occasion d'implémenter des arbres binaires en Python en utilisant la programmation orientée objet.

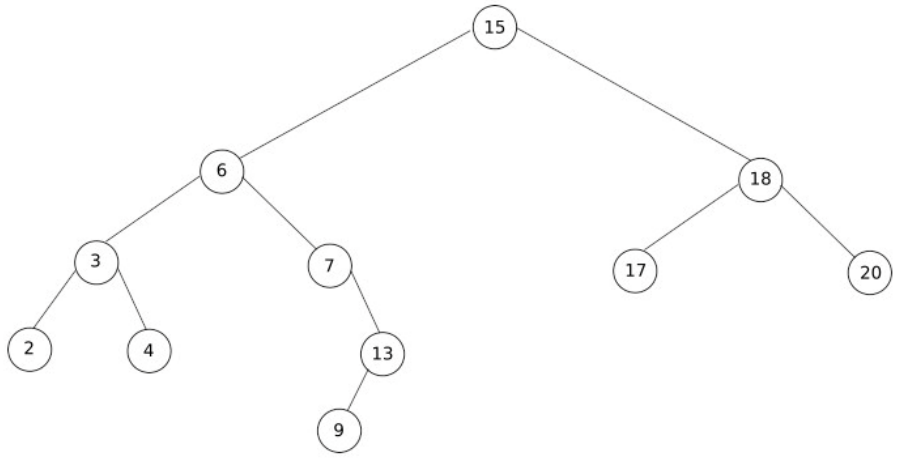
III- les arbres binaires de recherche

Un arbre binaire de recherche est un cas particulier d'arbre binaire.

Un **arbre binaire de recherche** est un arbre binaire dans lequel l'étiquette d'un nœud est appelé **clé** et est un entier. L'arbre binaire vérifie des propriétés suivantes :

- Les clés de tous les nœuds du sous-arbre gauche d'un nœud x sont inférieures ou égales à la clé de x.
- il faut que les clés de nœuds composant l'arbre soient ordonnables (on doit pouvoir classer les nœuds, par exemple, de la plus petite clé à la plus grande)
- soit x un nœud d'un arbre binaire de recherche. Si y est un nœud du sous-arbre gauche de x, alors il faut que y.clé $\leqslant$ x.clé. Si y est un nœud du sous-arbre droit de x, il faut alors que x.clé $\leqslant$ y.clé
- Les clés de tous les nœuds du sous-arbre droit d'un nœud x sont strictement supérieures à la clé de x.

Exemple d'arbre binaire de recherche :



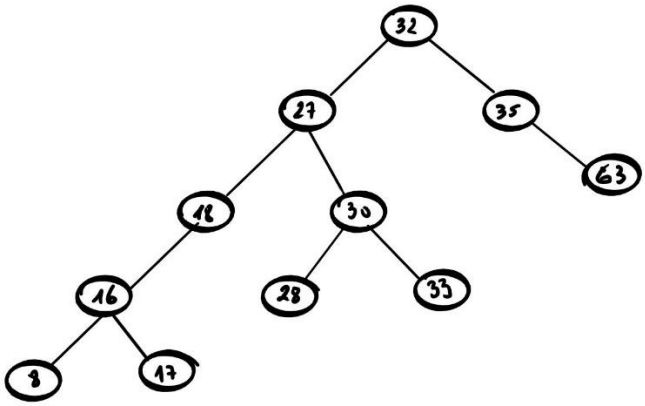
Vous pouvez vérifier que le fils gauche d'un nœud a une valeur plus petite que son père

(par exemple 3 < 6) et que le fils droit d'un nœud a une valeur plus grande que son père (par exemple 7 > 6)

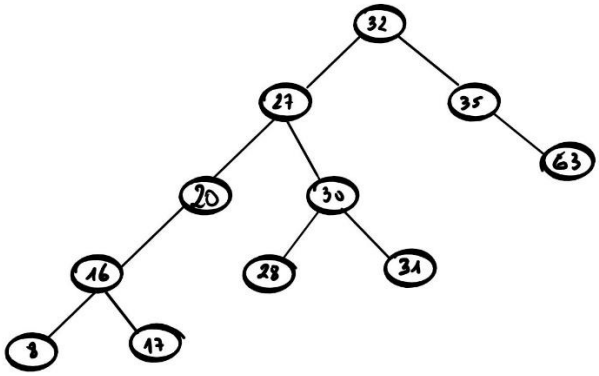
Attention : pour un nœud donné A, tous les nœuds de l'arbre gauche de A auront des valeurs plus petites que la valeur du nœud A et tous les nœuds de l'arbre droit de A auront des valeurs plus grandes que la valeur du nœud A.

Exemple :

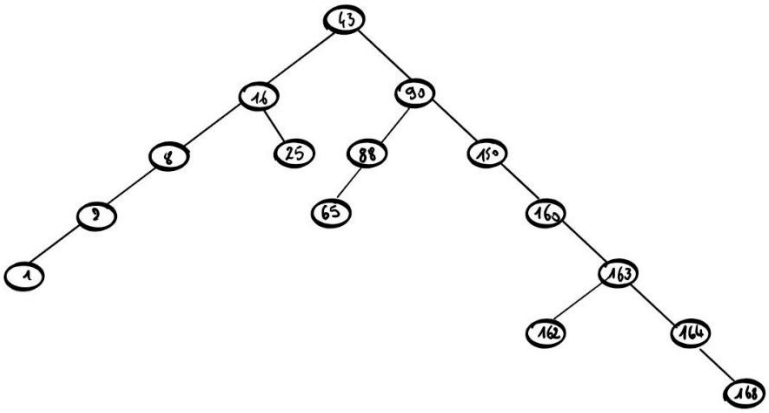
Quels sont parmi les arbres proposés ci-dessous les arbres binaires de recherche?



1



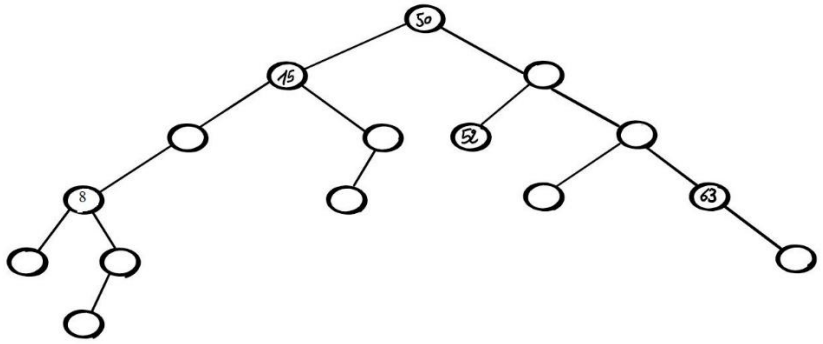
2



3

Exercice

Compléter cet arbre pour que ce soit un arbre binaire de recherche :

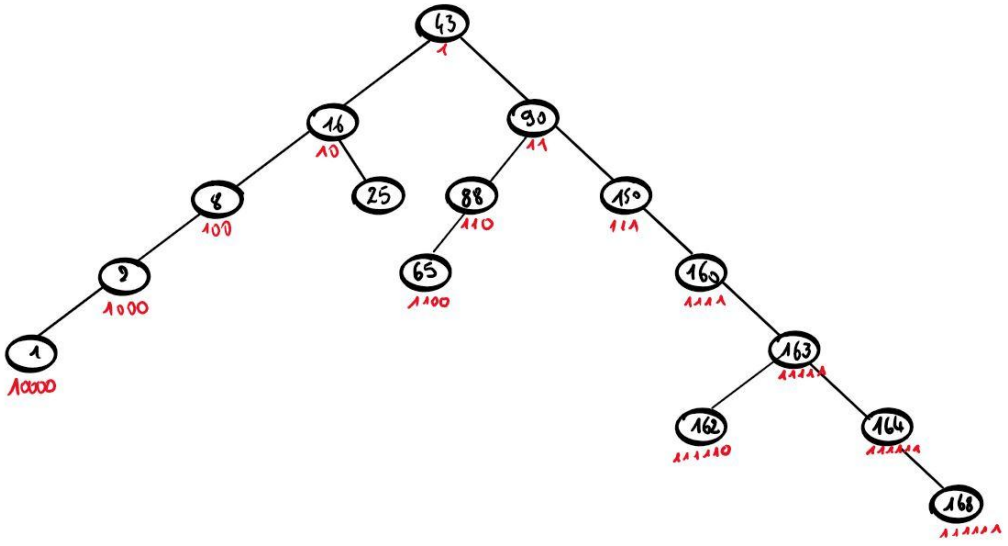


Propriété :

Un **étiquetage intéressant** d'un arbre de recherche est le suivant :

1. La racine est étiquetée 1.
2. Le premier nœud du sous arbre gauche prend l'étiquette de son père auquel on ajoute un 0.
3. Le premier nœud du sous arbre droit prend l'étiquette de son père auquel on ajoute un 1.

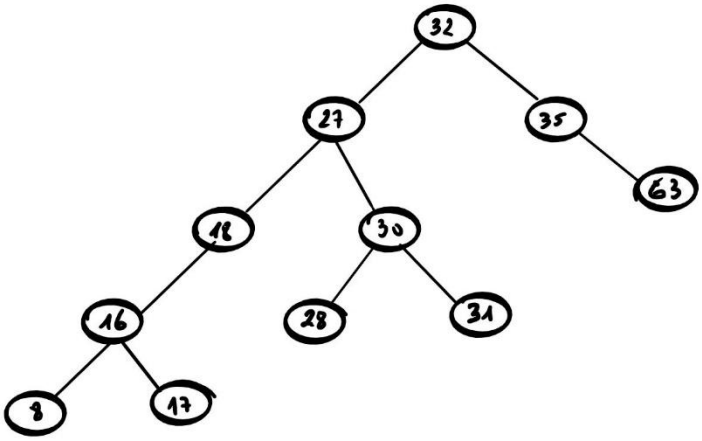
Exemple :



Quelle est l'étiquette de 25 dans l'arbre précédent ?

Exercice :

Etiqueter comme l'exemple précédent l'arbre suivant :



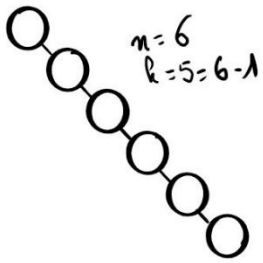
Propriété :

Pour un arbre binaire de recherche de hauteur h et de taille n les inégalités : $\lfloor log_2(n) \rfloor \leq h \leq n-1$

Pour démontrer ce résultat nous allons observer le cas où l'arbre est le plus profond et le cas où l'arbre est le moins profond.

- Cas où l'arbre est le plus profond :

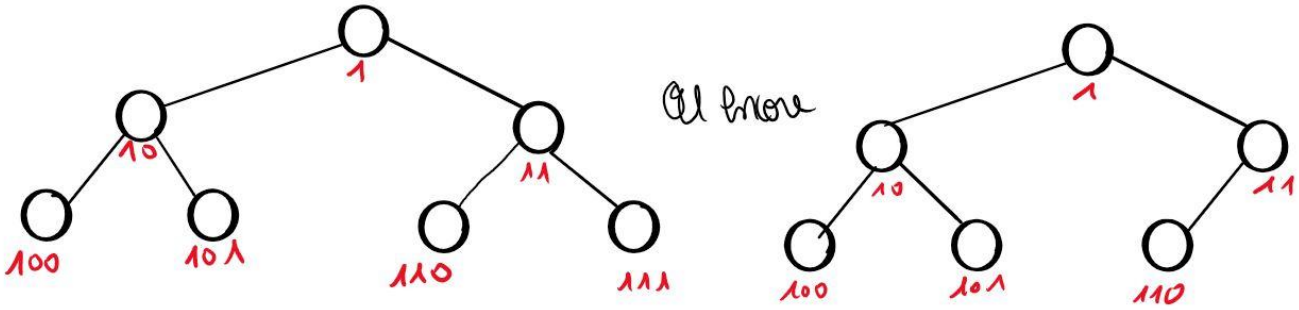
Pour une taille n fixée, la hauteur maximale d'un arbre binaire est $h = n - 1$, qu'on obtient avec des arbres « filiformes » comme cet arbre :



Ainsi $h \leq n - 1$

- Cas où l'arbre est le moins profond

Les arbres de taille n de hauteur minimale sont les arbres comme ces arbres :



Les clés des feuilles sont supérieures ou égales en binaire à 100...0100...0 où 0 est répété p fois

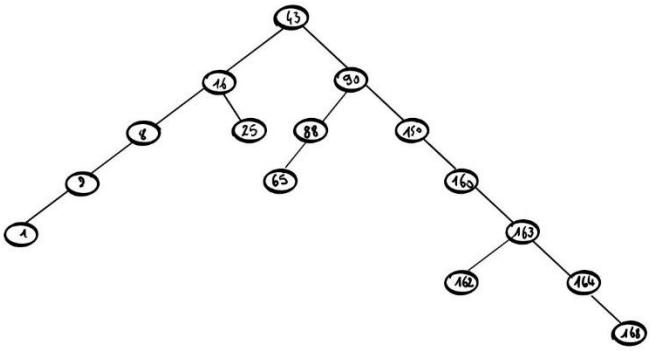
Dans ce cas la hauteur h est égale à p, il y a au moins n=2^p nœuds

Or $log_2(2^p) = p$

Ainsi $log_2(n) \leq p$

Exemple :

Vérifions cette inégalité sur cet arbre :



La taille de cet arbre est 15.

La hauteur est 6. $log_2(15) = 3{,}9$ à 10^{-1} $près$

⌊3,9⌋=3

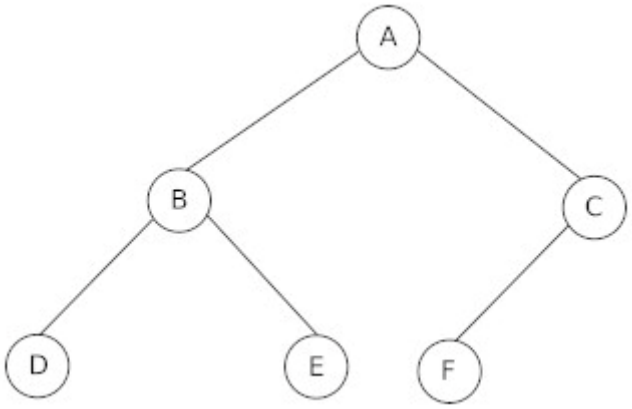
On a bien : 3 ≤ 6 ≤14

Activité 1

Trouvez un autre exemple de données qui peuvent être représentées par un arbre binaire (dans le domaine de votre choix). Dessinez au moins une partie de cet arbre binaire. Déterminez la hauteur et la taille de l'arbre que vous aurez dessiné.

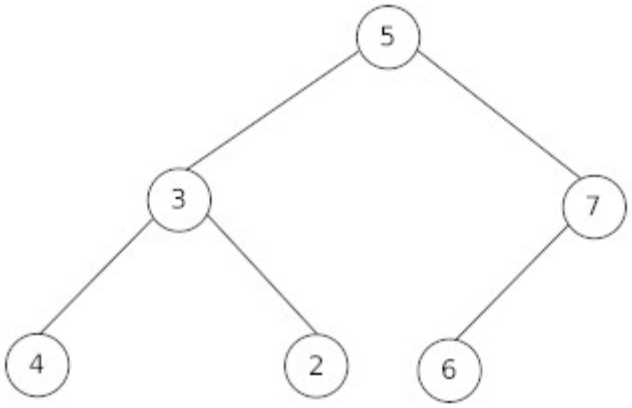
Activité 2

Soit l'arbre binaire suivant :



1. Cet arbre est-il un arbre binaire ? Justifiez votre réponse.
2. Donnez la clé (valeur) de la racine de cet arbre.
3. Quels sont les fils du nœud B.
4. Donnez l'arbre droit du nœud A.
5. Le nœud C est-il une feuille ? Justifiez votre réponse.
6. Donnez la taille de cet arbre.
7. Donnez la profondeur du nœud B (on prendra la profondeur de la racine égale à 0).
8. Donnez la hauteur de cet arbre (on prendra la profondeur de la racine égale à 0).

Activité 3



1. Expliquez pourquoi cet arbre binaire n'est pas un arbre binaire de recherche.
2. Modifiez cet arbre pour le transformer en arbre binaire de recherche.

Activité 4

Soit les valeurs suivantes : 14, 22, 8, 47, 42, 13, 1, 24, 33, 74.

Construisez un arbre binaire de recherche à partir de ces valeurs.

Les Arbres Binaires de Recherche en Python

Un **arbre binaire de recherche (ABR)** est un **arbre binaire ordonné** dans lequel :

- Les **valeurs du sous-arbre gauche** sont **inférieures** à la racine ;
- Les **valeurs du sous-arbre droit** sont **supérieures** à la racine ;
- Chaque sous-arbre est lui-même un arbre binaire de recherche.

Cette structure est très utilisée pour :

- La **recherche rapide** d'une donnée ;
- Le **tri dynamique** ;
- Les **bases de données** et **compilateurs**.

Représentation d'un arbre en Python

En Python, un arbre n'est **pas un type de donnée intégré** (comme une liste ou un dictionnaire).
On le construit donc **soi-même**, souvent à l'aide d'une **classe `Noeud`**.

Exemple :

```
class Noeud:
    def __init__(self, valeur):
        self.valeur = valeur
        self.gauche = None
        self.droite = None
```

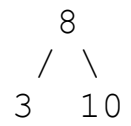
Chaque nœud contient :

- une **valeur** (entier, chaîne, etc.)
- un **fils gauche**
- un **fils droit**.

Visuellement, si on écrit :

```
racine = Noeud(8)
racine.gauche = Noeud(3)
racine.droite = Noeud(10)
```

On obtient :



Insertion dans un arbre binaire de recherche

L'**insertion** se fait récursivement :

- si l'arbre est vide → le nouveau nœud devient la racine ;
- sinon :
  - si la valeur est **plus petite**, on insère à gauche ;
  - sinon, on insère à droite.

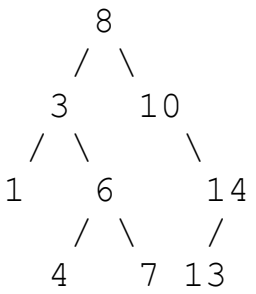
Exemple de fonction d'insertion :

```
def inserer(racine, valeur):
    if racine is None:
        return Noeud(valeur)
    if valeur < racine.valeur:
        racine.gauche = inserer(racine.gauche, valeur)
    elif valeur > racine.valeur:
        racine.droite = inserer(racine.droite, valeur)
    return racine
```

Exemple d'utilisation :

```
racine = None
for v in [8, 3, 10, 1, 6, 14, 4, 7, 13]:
    racine = inserer(racine, v)
```

On obtient l'arbre suivant :



Recherche dans un ABR

La **recherche** est également récursive :

- Si la valeur recherchée est égale à la racine → trouvée ;
- Si elle est plus petite → chercher à gauche ;
- Si elle est plus grande → chercher à droite.

Exemple :

```
def rechercher(racine, valeur):
    if racine is None:
        return False
    if racine.valeur == valeur:
        return True
    elif valeur < racine.valeur:
        return rechercher(racine.gauche, valeur)
    else:
        return rechercher(racine.droite, valeur)
```

Exemple d'utilisation :

```
print(rechercher(racine, 6))   # True
print(rechercher(racine, 15))  # False
```

Parcours d'un arbre binaire

a. Parcours en préfixe (Pré-ordre)

Racine → Gauche → Droite
Utilisé pour **afficher la structure complète de l'arbre**.

```
def parcours_prefixe(racine):
    if racine:
        print(racine.valeur, end=" ")
        parcours_prefixe(racine.gauche)
        parcours_prefixe(racine.droite)
```

Résultat : `8 3 1 6 4 7 10 14 13`

b. Parcours en infixe (In-ordre)

Gauche → Racine → Droite
Utilisé pour **obtenir les valeurs triées** dans un ABR.

```
def parcours_infixe(racine):
    if racine:
        parcours_infixe(racine.gauche)
        print(racine.valeur, end=" ")
        parcours_infixe(racine.droite)
```

Résultat : `1 3 4 6 7 8 10 13 14`

c. Parcours en postfixe (Post-ordre)

Gauche → Droite → Racine
Utilisé pour **supprimer** ou **libérer** un arbre.

```
def parcours_postfixe(racine):
    if racine:
        parcours_postfixe(racine.gauche)
        parcours_postfixe(racine.droite)
        print(racine.valeur, end=" ")
```

Résultat : `1 4 7 6 3 13 14 10 8`

Hauteur et taille d'un arbre

Taille = nombre total de nœuds

```
def taille(racine):
    if racine is None:
        return 0
    return 1 + taille(racine.gauche) + taille(racine.droite)
```

Hauteur = profondeur maximale

```
def hauteur(racine):
    if racine is None:
        return 0
    return 1 + max(hauteur(racine.gauche), hauteur(racine.droite))
```

Exemple

```
class Noeud:
    def __init__(self, valeur):
        self.valeur = valeur
        self.gauche = None
        self.droite = None

def inserer(racine, valeur):
    if racine is None:
        return Noeud(valeur)
    if valeur < racine.valeur:
        racine.gauche = inserer(racine.gauche, valeur)
    elif valeur > racine.valeur:
        racine.droite = inserer(racine.droite, valeur)
    return racine

def rechercher(racine, valeur):
    if racine is None:
        return False
    if valeur == racine.valeur:
        return True
    elif valeur < racine.valeur:
        return rechercher(racine.gauche, valeur)
    else:
        return rechercher(racine.droite, valeur)

def parcours_infixe(racine):
    if racine:
        parcours_infixe(racine.gauche)
        print(racine.valeur, end=" ")
        parcours_infixe(racine.droite)

def taille(racine):
    if racine is None:
        return 0
    return 1 + taille(racine.gauche) + taille(racine.droite)

def hauteur(racine):
    if racine is None:
        return 0
    return 1 + max(hauteur(racine.gauche), hauteur(racine.droite))

# Test complet
racine = None
for v in [8, 3, 10, 1, 6, 14, 4, 7, 13]:
    racine = inserer(racine, v)

print("Parcours infixe (valeurs triées) :")
parcours_infixe(racine)
print("\nTaille =", taille(racine))
print("Hauteur =", hauteur(racine))
print("Recherche 6 :", rechercher(racine, 6))
print("Recherche 15 :", rechercher(racine, 15))
```

Résultat attendu :

```
Parcours infixe (valeurs triées) :
1 3 4 6 7 8 10 13 14
Taille = 9
Hauteur = 4
Recherche 6 : True
Recherche 15 : False
```

Application concrète : arbre de contacts

On peut utiliser un ABR pour gérer un **répertoire téléphonique** trié par nom.

```
class Contact:
    def __init__(self, nom, numero):
        self.nom = nom
        self.numero = numero
        self.gauche = None
        self.droite = None

def ajouter_contact(racine, nom, numero):
    if racine is None:
        return Contact(nom, numero)
    if nom < racine.nom:
        racine.gauche = ajouter_contact(racine.gauche, nom, numero)
    elif nom > racine.nom:
        racine.droite = ajouter_contact(racine.droite, nom, numero)
    return racine

def rechercher_contact(racine, nom):
    if racine is None:
        return None
    if nom == racine.nom:
        return racine.numero
    elif nom < racine.nom:
        return rechercher_contact(racine.gauche, nom)
    else:
        return rechercher_contact(racine.droite, nom)
```

TP :Mini-calculatrice récursive avec arbre d'expression

Objectif

L'objectif de ce TP est de manipuler la **récursivité** et les **arbres binaires** à travers la création d'une mini-calculatrice capable d'évaluer des expressions arithmétiques.

Un **arbre d'expression** est une structure de données utilisée pour représenter une expression mathématique.
Chaque **nœud interne** contient un **opérateur** (`+`, `-`, `*`, `/`)
et chaque **feuille** contient un **nombre**.

Exemple pour l'expression :

```
(2 * 4) + 3
```

L'arbre correspondant est :

1- Définition d'une classe Noeud

Créer une classe `Noeud` permettant de représenter un nœud d'arbre.
Chaque nœud doit contenir :

- une valeur (`valeur`),
- une référence vers un sous-arbre gauche (`gauche`),
- une référence vers un sous-arbre droit (`droite`).

2- Construction d'un arbre d'expression simple

Construire **manuellement** l'arbre correspondant à l'expression suivante :

```
(2 * 4) + 3
```

3- Parcours récursif

Écrire une fonction `afficher_infixe(noeud)` qui affiche l'expression dans sa forme classique (infixe).
Par exemple, pour l'arbre ci-dessus, le programme doit afficher :

```
(2 * 4) + 3
```

4- Évaluation récursive

Écrire une fonction `evaluer(noeud)` qui :

- si le nœud est une **feuille**, retourne sa valeur numérique ;
- sinon, évalue récursivement les sous-arbres gauche et droit, puis applique l'opérateur contenu dans le nœud.

Tester la fonction sur l'arbre de l'étape 2.

5- Construction automatique depuis une expression postfixée

Une expression postfixée (ou polonaise inversée) est une notation où les opérateurs sont placés **après** leurs opérandes.
Exemple :

```
(2 * 4) + 3  →  2 4 * 3 +
```

Écrire une fonction `construire_arbre_postfixe(expression)` qui :

1. lit une chaîne représentant une expression postfixée (exemple : "`2 4 * 3 +`"),
2. construit et retourne l'arbre d'expression correspondant.

6- Tests

Tester votre programme avec les expressions suivantes :

Expression classique	Expression postfixée	Résultat attendu
(2 * 4) + 3	2 4 * 3 +	11
(5 + 2) * (8 - 3)	5 2 + 8 3 - *	35
7 + (6 * 5^2 + 3)	7 6 5 2 ^ * 3 + +	160

7- Gestion des puissances et division entière

1. Ajouter la gestion de l'opérateur ^ pour la puissance.

2. Ajouter un opérateur `//` pour la division entière.
3. Adapter les fonctions précédentes pour les prendre en compte.

8- Interface console

Créer un petit menu dans le terminal :

```
=== Mini Calculatrice Arborescente ===
1. Entrer une expression postfixée
2. Afficher l'expression infixée
3. Évaluer l'expression
4. Quitter
```

L'utilisateur pourra saisir une expression postfixée, voir son affichage arborescent et obtenir le résultat.

Compétences

- Récursivité
- Structures arborescentes
- Construction et parcours d'arbres
- Manipulation d'expressions arithmétiques