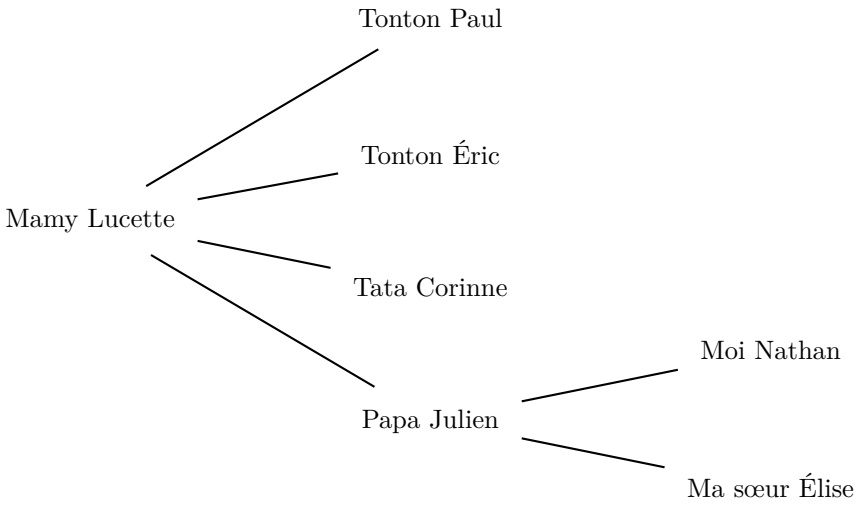
Chapitre 6 : Structures de données hiérarchiques les arbres

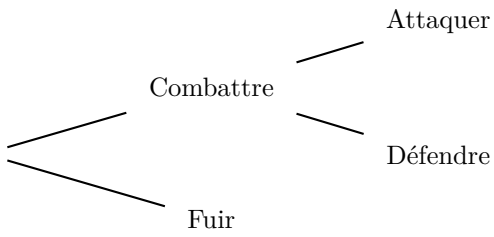
6.1 Introduction et définition

En informatique on est souvent confronté à des situations dans lesquelles une structure hiérarchique est sous-jacente. Les structures de données étudiées dans les séquences précédentes telles que les piles, listes ou dictionnaires sont insuffisantes à réprésenter.
De nombreuses informations peuvent être organisées sous la forme d'un arbre :

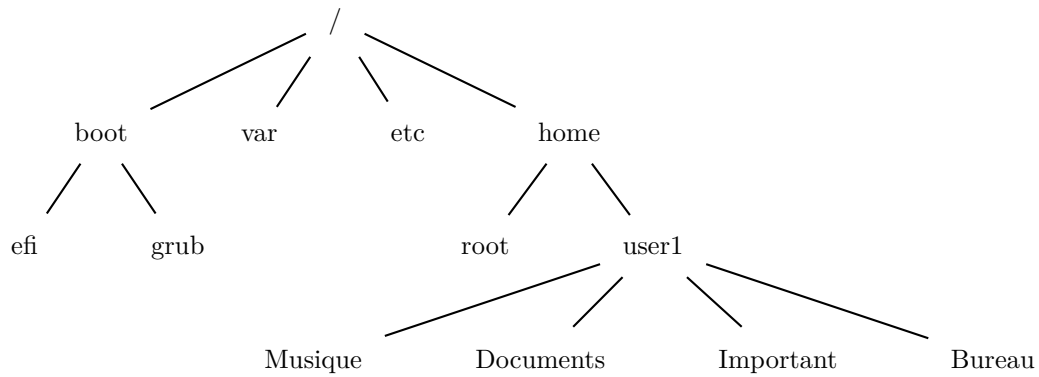
- un arbre généalogique,



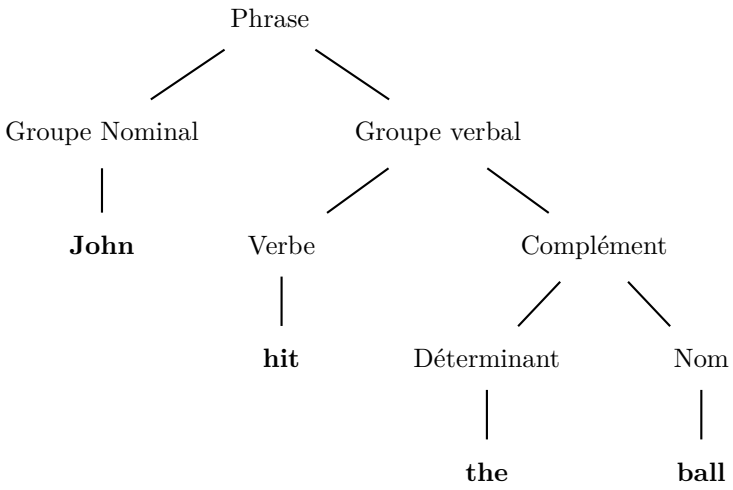
- une série de choix successifs,



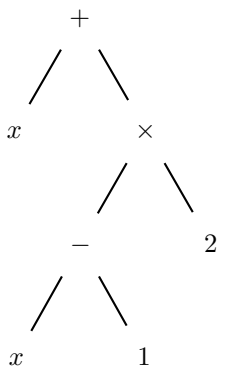
- un système de classement comme par exemple la « classification scientifique des espèces » ou les fichiers et sous-fichiers d'un système de stockage de données.



- une structure syntaxique en linguistique, exemple « *John hit the ball* »



- une expression arithmétique, exemple $x + (x - 1) \times 2$



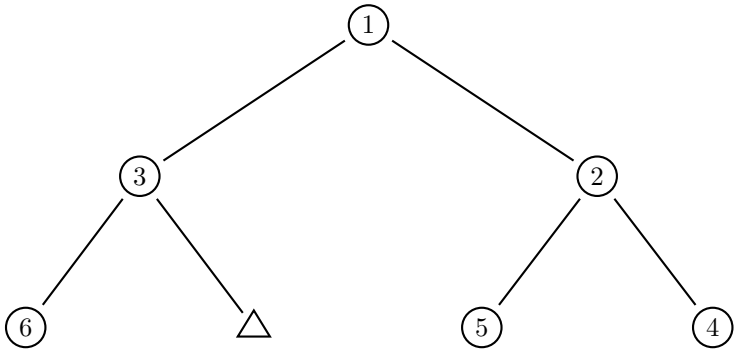
Pour définir formellement un arbre on peut utiliser ce qu'on appelle une définition récursive comme pour les listes.

Définition 1
Un arbre est

- soit vide
- soit constitué d'un élement auquel sont liés un ou plusieurs sous-arbres.

6.2 Vocabulaire et terminologie

Considérons l'exemple d'arbre ci-dessous :



N.B : on convient que le symbole triangle représente l'abre vide

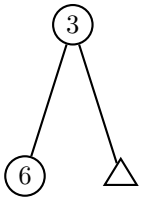
Chaque sommet est un **nœud** l'ensemble des nœuds de cet arbre est {1, 2, 3, 4, 5, 6}.
Le nœud initial 1 est appelé nœud **racine** ou plus simplement **racine**. L'arc reliant deux nœuds est appelé **branche**.
Les nœuds 1, 2, 3 possèdent des fils (2 et 3 pour le nœud 1, 5 et 4 pour le nœud 2 et 6 pour le nœud 3).
Les nœuds qui n'ont pas de fils (ici 6, 4 et 5) sont appellés **feuille** de l'arbre ou parfois nœud terminal.

Définition 2
Lorsqu'un arbre admet, pour chaque nœud, **au plus** n fils, l'arbre est dit **n-aire**. Dans le cas ou $n = 2$ on parle d'arbre **binaire**.

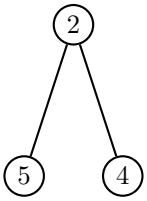
Remarque : tout sous-arbre d'un arbre binaire est un arbre binaire.

L'arbre précédent est un exemple d'arbre binaire, chaque nœud à au maximum 2 fils. Pour les arbres binaires, on parlera plus spécifiquement de **sous-arbre gauche** et de **sous-arbre droit**.

Par exemple pour l'arbre précédent :
• son sous-arbre gauche est :



• son sous-arbre droit est :



De part sa définition on sera souvent amené a écrire des algorithmes récursifs sur les arbres. Les algorithmes récursifs sur une structure de données récursifs sont non seulement plus naturels mais aussi plus concis que leurs homologues itératifs.
À retenir pour l'écriture récursive d'algorithme sur les arbres binaires :

Définition 3
Un arbre binaire est :

- soit vide,
- soit un élement (nœud) auquel sont reliés un sous-arbre gauche binaire et un sous-arbre droit binaire.

On considère des arbres binaires constitués de nœuds appartenant à un ensemble V, on va dégager de ce qui précède la structure de données abstraites ou interface des arbres binaires de V.

Les données : élements de l'ensemble V
Les opérations primitives :

`creer_arbre()` : crée un arbre binaire vide

`creer_arbre_racine(v)` : crée un arbre binaire dont la racine est `v` un élement de V et dont les sous-arbres gauche et droit sont vides.

`est_vide(A)` : renvoie vrai si l'arbre `A` est vide sinon faux.

`racine(A)` : renvoie la racine de l'arbre `A`.

`sous_arbre_gauche(A)` : renvoie le sous-arbre gauche de `A`.

`sous_arbre_droit(A)` : renvoie le sous-arbre droit de `A`.

`modifie_racine(A, v)` : modifie la racine de l'arbre binaire `A` par `v`, un élement de V.

`modifie_arbre_gauche(A, G)` : remplace le sous-arbre gauche de l'arbre binaire `A` par l'arbre binaire `G`.

`modifie_arbre_droite(A, D)` : modifie la racine de l'arbre binaire `A` par `v`, un élement de V.

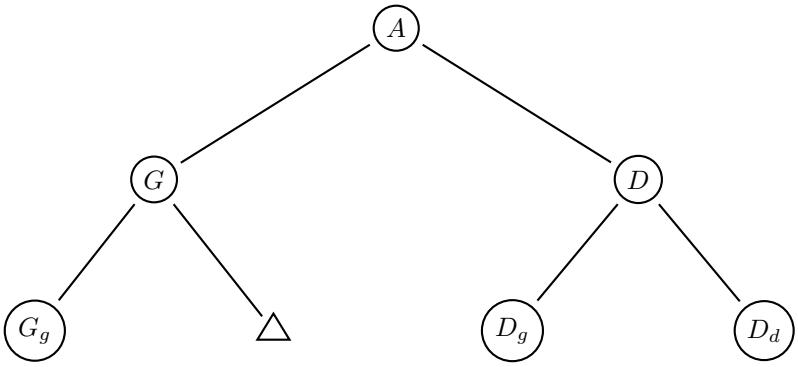
6.3 Implémentation en Python et réprésentation

a. Implémentation objet

Pour cette implémentation, on va utiliser des notions de programmation objet vues dans la séquence 1 en créant une classe `Noeud` qui exploitera la nature récursive des arbres binaires.

```
class Noeud :
    def __init__(self, racine):
        self.gauche = None
        self.racine = racine
        self.droit = None
    def __repr__(self):
        """cette methode est recursive car str invoque __repr__.
           Il existe plusieurs manieres de parcourir l'arbre.
        """
        return f'{self.racine}({str(self.gauche)},{str(self.droit)})'
```

Jusqu'ici on a représenté graphiquement les arbres c'est une façon efficace pour appréhender leurs structures hiérarchiques. Cependant cette représentation n'est pas toujours simple à dessiner, on peut représenter un arbre binaire en « ligne » à l'aide de la notation parenthésée.



peut être noté $A(G,(G_g(\emptyset,\emptyset),\emptyset),D,(D_g(\emptyset,\emptyset),D_d(\emptyset,\emptyset)))$. On simplifie la notation en supprimant les arbres complétement vides $A(G,(G_g,\emptyset),D,(D_g,D_d))$.
Avec la classe `Noeud` on peut instancier cet arbre binaire comme ci-dessous :

```
arbre_G = Noeud('G')
arbre_Gg = Noeud('G_g')
arbre_G.gauche = arbre_Gg

arbre_D = Noeud('D')

arbre_Dg = Noeud('D_g')
```

```
arbre_Dd = Noeud('D_d')
arbre_D.gauche = arbre_Dg
arbre_D.droit = arbre_Dd

arbre_A = Noeud('A')
arbre_A.gauche = arbre_G
arbre_A.droit  = arbre_D
```

Pour se donner une idée, on peut réprésenter l'arbre `arbre_A` en saisisant tout simplement `arbre_A` dans une console Python ou `print(arbre_A)` depuis le mode execution (rappel : c'est la méthode `__repr__` qui est invoquée dans les deux cas) ce qui donne :

```
>>> arbre_A
A( G( G_g(None,None),None), D( D_g(None,None), D_d(None,None)))
```

Ce qui est conforme à la notation parenthésée, en effet la méthode `__repr__` parcours récursivement en profondeur (« du haut vers le bas ») l'arbre binaire `A`

b. Différents parcours et représentations

- parcours prefixe : on liste la racine en premier puis l'arbre gauche suivi du droit ;
- parcours postfixe : on liste l'arbre gauche, le droit puis la racine ;
- parcours infixe : on liste l'arbre gauche, la racine puis l'arbre droit.

Donner, ci-dessous, les parcours postfixe et infixe de `arbre_A`.

6.4 Quelques mesures des arbres binaires

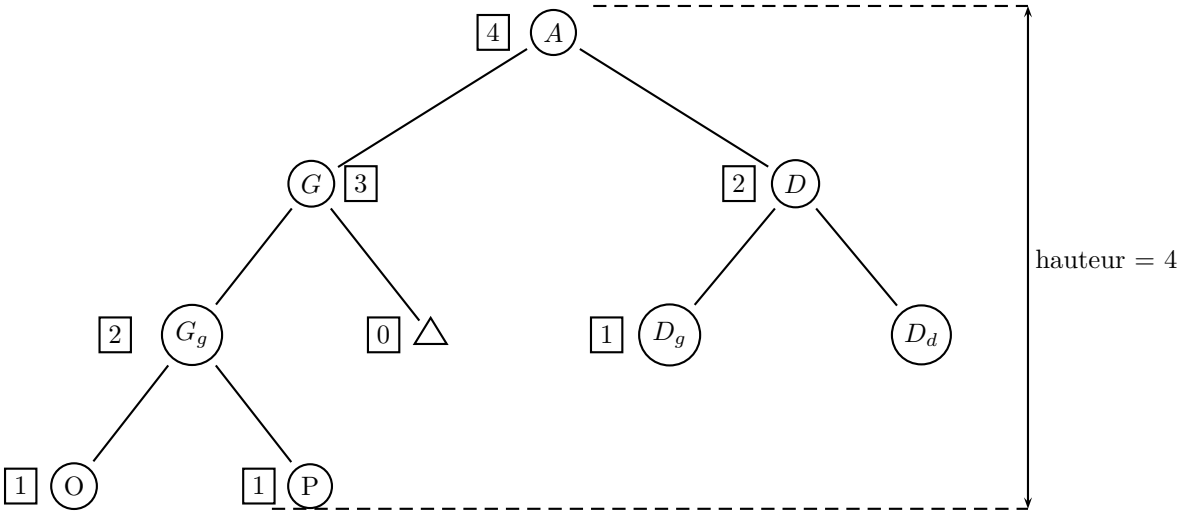
Intuitiviement on peut attribuer des mesures à des arbres binaires, la plus naturelle pour un arbre est la hauteur.
Il semble naturel d'appeler hauteur d'un arbre le nombre maximum de niveaux entre la racine et ses feuilles. Autrement dit, le nombre de niveaux séparant le point le plus « haut » (la racine) et les points les plus « bas » (les feuilles).

Définition 4
La hauteur ou profondeur d'un arbre est égale au maximum du niveau de ses feuilles.

Cette définition naturelle et intuitive peut être complétée par une définition récursive équivalente qui pourra être exploité pour établir un algorithme récursif calculant la hauteur d'un arbre binaire.

Définition 5
La hauteur d'un arbre binaire vide est nulle
La hauteur d'un arbre binaire est égale à la hauteur de sa racine.
La hauteur d'un nœud est égale à un de plus que le maximum du sous-arbre gauche et du sous-arbre droit.

Dans l'exemple ci-dessous, ont été représentées dans les rectangles les hauteurs des nœuds. On peut facilement comprendre le cheminement en partant du bas où les feuilles on pour hauteur 1 puis lorsqu'on remonte d'un niveau le nœud aura pour hauteur $= \max(\text{hauteur fils gauche}, \text{hauteur fils droit}) + 1$.



On peut ajouter une méthode à la classe `Noeud` permettant de calculer récursivement la hauteur d'un arbre binaire.

```
def hauteur(self):
        if self == None :
            return 0
        else :
            hauteur_gauche = self.gauche.hauteur() if self.gauche else 0
            hauteur_droite = self.droit.hauteur() if self.droit else 0
            return 1 + max(hauteur_gauche, hauteur_droite)
```

On peut s'intéresser aussi à la « taille » d'un arbre qui serait tout simplement le nombre d'éléments de cet arbre.

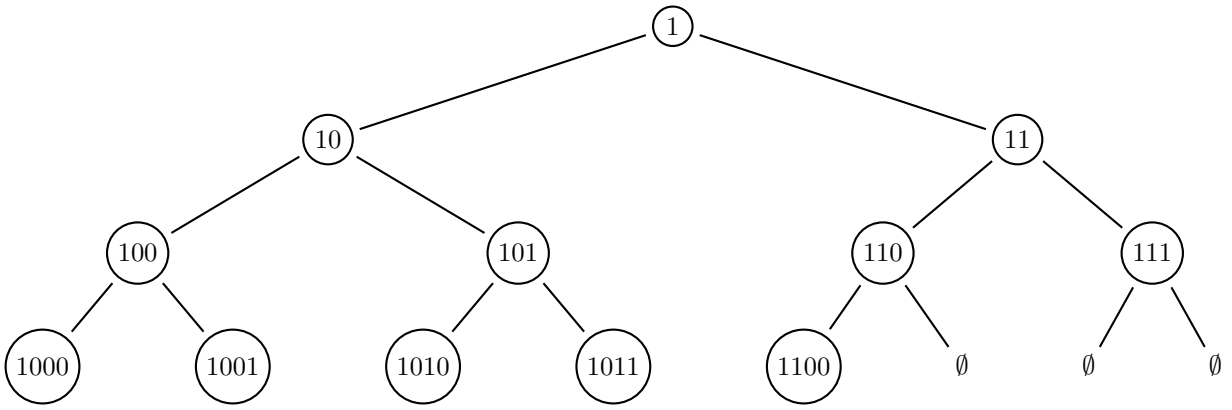
Définition 6
La taille d'un arbre est le nombre d'élements de celui-ci.
Version récursive :

- La taille d'un arbre binaire vide est nulle.
- La taille d'un arbre binaire non-vide est $1 +$ la taille de sous-arbre gauche $+$ la taille de sous-arbre droit.

▶ Exercice 1

En s'insipirant de la méthode `hauteur` écrire une méthode `taille` de la classe `Noeud` retournant la taille d'un objet de type `Noeud`.

Il existe une relation entre la taille et la hauteur.
Soit n un nombre entier d'éléments. L'arbre de hauteur minimale pouvant conteniir c'est n éléments est un arbre binaire parfaitement équilibré et complet avec autant de nœud à gauche qu'à droite et ce à chaque niveau.
On considére le nombre n en base 2, il s'écrit avec r digits par exemple $n = 12$ s'écrit en base deux $n = 1100$ avec 4 chiffres. On cherche à placer 12 élements dans un arbre binaire en minimisant la hauteur d'un tel arbre. Voici comment on peut procéder, on place le premier au premier niveau puis le second et le troisième de gauche à droite au deuxième niveau et ainsi de suite ce qui donne en base deux :



Pour placer 12 élements qui s'écrit avec 4 chiffres en base deux il faut un arbre binaire au minimum de hauteur 4.

Ce résultat se généralise pour n et h deux entiers tels que $2^{h-1} \leqslant n < 2^h$ la hauteur minimale d'un arbre binaire contenant n élements sera h (n s'écrira en base deux avec h chiffres).

Propriété 1
Soit n la taille d'un arbre binaire et h sa hauteur h on a $n \leqslant 2^h - 1$ autrement dit la hauteur h de l'arbre binaire est au minimum le nombre de chiffres de n en base 2 et au maximum n.
Ce qui peut se noter

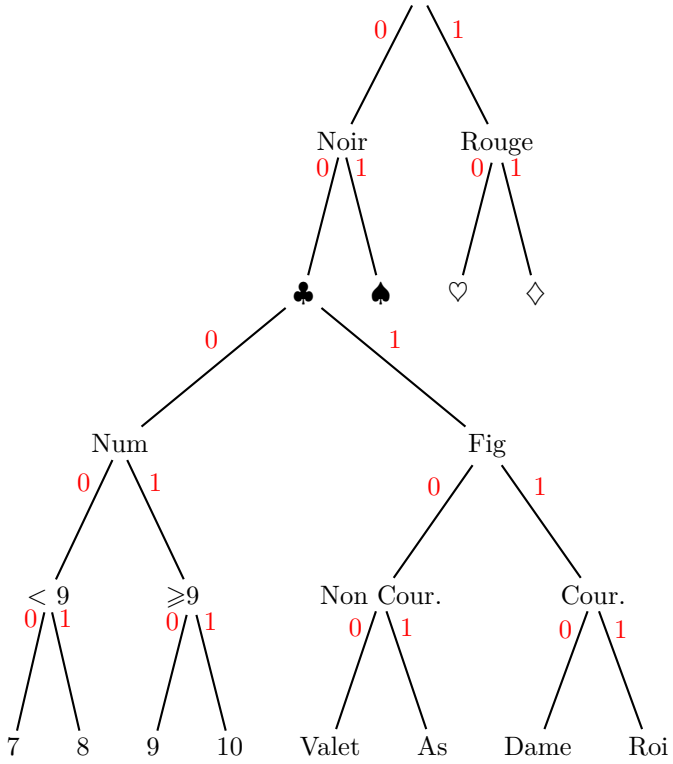
$$\log_2(n+1) \leqslant h \leqslant n$$

avec $\log_2$ le logarithme base 2.

On retient que pour stocker dans un arbre binaire de n nœuds il faudra une hauteur minimale de l'ordre du nombre de bits de ce nombre en base de 2.
Par exemple un arbre binaire doit contenir 197 éléments comme $2^7 = 128$ et $2^8 = 256$, $197_2 = 1100\,0101_2$ il faudra compter au minimum un arbre binaire de hauteur 8.

▶ Exercice 2

On peut utiliser la structure hiérarchisée d'arbre binaire pour coder les éléments d'un ensemble fini E en considérant le chemin menant de la racine à un élément et en associant un « 0 » lorsque l'on emprunte le sous-arbre gauche et un « 1 » lorsque l'on emprunte celui de droite. On retrouvera ce principe dans la compression numérique sans perte de données avec le codage et l'arbre d'Huffman (en positionnant les éléments les plus fréquents en haut de l'arbre) .
Considérons un paquet de 32 cartes ; pour représenter toutes les cartes on peut utiliser un arbre à 5 niveaux puisque $2^5 = 32$. Les critères sont illustrés ci-contre (une partie seulement de l'arbre est développée).
Ainsi l'As de trèfle pourra être codé 00101 et la Dame de cœur 10110, l'arbre sert à coder et à décoder.



Dans cet exemple chaque carte est codée sur 5 bits. Pour décoder une main il suffit de lire les codes par paquet de 5 mais en fait il n'est pas réellement nécessaire de connaître cette information. On peut lire cette suite de 0 et de 1, bit par bit en partant de la racine de l'arbre et ainsi interpréter le code lorsqu'une feuille est atteinte puis repartir de nouveau de la racine et poursuivre ainsi jusqu'à atteindre la fin de la chaîne binaire.

1. J'ai dans les mains 5 cartes saurez-vous lire mon jeu dont le code est 00001010101000000111011101.

2. On suppose que l'on dispose de l'arbre binaire représenté ci-dessus sous la forme d'un objet `paquet` de la classe `Noeud` Écrire une fonction `Huffman` qui prenne en argument une chaîne de cinq caractères constituée de '0' et de '1' et qui renvoie la châine correspondant à la carte ainsi codée.

D'autres exemples et des situations mettant en œuvre des algorithmes sur les arbres seront présentés dans la séquence 9.

6.5 Réponses des exercices

▶ Exercice 1

Pour calculer la taille d'un arbre binaire on utilise la définition récursive de la taille qui donne directement l'algorihtme. On rajoute la méthode `taille` à la classe `Noeud`.

```
def taille(self):
        if self == None :
            return 0
        else :
            taille_gauche = self.gauche.taille() if self.gauche else 0
            taille_droit = self.droit.taille() if self.droit else 0
            return 1 + taille_gauche + taille_droit
```

▶ Exercice 2

1. On parcourt l'arbre binaire depuis la racine on choisit la branche de gauche si on rencontre un zéro et la branche de droite si c'est un un. Dès qu'on atteint une feuille on a la carte puis on repart de la racine de l'arbre et on recommence jusqu' à atteindre la fin de la chaîne.
   On obtient donc 8 de trèfle, 9 de pique, 7 de cœur, Dame de pique et As de carreau.

2. ```
   def huffman(paquet, chaine):
       """prend en argument un arbre et une chaine de 5 caracteres des 0 ou 1
          et renvoie le nom de la carte correspondant de l'arbre paquet"""
       arbre  = paquet
       for index, valeur in enumerate(chaine):
           if valeur == '0':
               arbre = arbre.gauche
           else :
               arbre = arbre.droit

           if index == 1:
               nom = arbre.racine
       valeur = arbre.racine
       return f'{valeur} de {nom}'
   ```

   L'arbre correspondant à un paquet de 32 de cartes conformément à la réprésentation de l'exercice est donné ci-dessous.

   ```
   #Arbre num
   arbre_7_8 = Noeud('<9')
   arbre_7_8.gauche = Noeud('7')
   arbre_7_8.droit = Noeud('8')

   arbre_9_10 = Noeud('>=9')
   arbre_9_10.gauche = Noeud('9')
   arbre_9_10.droit = Noeud('10')

   arbre_num = Noeud('Num')
   arbre_num.gauche = arbre_7_8
   arbre_num.droit = arbre_9_10

   #arbre fig
   arbre_non_cour = Noeud('Non cour')
   arbre_non_cour.gauche = Noeud('Valet')
   arbre_non_cour.droit = Noeud('As')
   ```
   ```

```
arbre_cour = Noeud('Cour')
arbre_cour.gauche = Noeud('Dame')
arbre_cour.droit = Noeud('Roi')

arbre_fig = Noeud('Fig')
arbre_fig.gauche = arbre_non_cour
arbre_fig.droit = arbre_cour

#arbre trefle, pique, coeur, carreau
arbre_trefle = Noeud('Trefle')
arbre_pique = Noeud('Pique')
arbre_coeur = Noeud('Coeur')
arbre_carreau = Noeud('Carreau')


arbre_trefle.gauche = arbre_num
arbre_trefle.droit = arbre_fig

arbre_pique.gauche = arbre_num
arbre_pique.droit = arbre_fig

arbre_coeur.gauche = arbre_num
arbre_coeur.droit = arbre_fig

arbre_carreau.gauche = arbre_num
arbre_carreau.droit = arbre_fig

arbre_coeur.gauche = arbre_num
arbre_coeur.droit = arbre_fig

#arbre Noir Rouge
arbre_noir = Noeud('Noir')
arbre_rouge = Noeud('Rouge')

arbre_noir.gauche = arbre_trefle
arbre_noir.droit = arbre_pique

arbre_rouge.gauche = arbre_coeur
arbre_rouge.droit = arbre_carreau

#arbre paquet de cartes
paquet = Noeud('/')
paquet.gauche = arbre_noir
paquet.droit = arbre_rouge
```