```
# /Users/mac/Documents/CLY/Tale NSI/arbres/DS03/correction DS03 NSI Tale.py
001| ############################################################
002| ### Le 24/01/2022                                        ###
003| ### MatheX                                              ###
004| ### Licence CC BY-NC-SA 4.0                             ###
005| ### Tale SPE NSI                                        ###
006| ### Correction DS03 - Exercice 1                        ###
007| ### Structures de données hiérarchiques                  ###
008| ############################################################
009|
010|
011| ###############################################################################
012| #                           Exercice 1: (10 points)                           #
013| ###############################################################################
014|
015| # 1.1. Représenter un arbre binaire de recherche dont les noeuds ont
016| #      les valeurs: 1 ; 5 ; 10 ; 12 ; 17 ; 21 ; 25 ; 26 ; 28 ; 30:
017| # ---------------------------------------------------------------------
018|
019| '''
020|                     ___21___
021|                    /        \
022|                  12          26
023|                 /  \        /  \
024|                5   17      25  28
025|               / \                 \
026|              1  10                 30
027| '''
028|
029|
030| # 1.2. Ajouter à cet arbre les noeuds de valeurs: 3 ; 14 ; 27 .
031| #      Préciser la taille et la hauteur de l'arbre obtenu:
032| # ---------------------------------------------------------------------
033|
034| '''
035|                     ___21___                        0
036|                    /        \
037|                  _12_        26                   1
038|                 /    \      /  \
039|                5     17    25  28                 2
040|               / \    /        /   \
041|              1  10  14       27   30              3
042|               \
043|                3                                  4
044|
045| hauteur: 4 (ou 5 si on commence à 1)
046| taille: 13
047|
048| '''
049|
050| # 1.3. Donner les valeurs des noeuds qu'afficheraient un parcours
051| #      d'abord en profondeur préfixé , infixé  et postfixé:
052| # ---------------------------------------------------------------------
053|
054| '''
055| préfixé (d'abord la valeur puis gauche puis droite:
056|     21   12   5   1   3   10   17   14   26   25   28   27   30
057|
058| infixé (d'abord gauche puis la valeur puis droite:
059|     1   3   5   10   12   14   17   21   25   26   27   28   30
060|
061| postfixé (d'abord gauche puis droite puis la valeur:
062|     1   3   5   10   12   14   17   21   25   26   27   28   30
063| '''
064|
065| # 1.4. Implémenter la classe Noeud modélisant un noeud de l'arbre
066| # 1.5. Implémenter une méthode qui recherche si une valeur est dans l'arbre
```

```python
# 1.6. Implémenter une méthode qui calcule la taille de l'arbre
# ------------------------------------------------------------------------

class Noeud:
    '''
    classe modélisant un noeud d'un arbre par une valeur, un noeud de gauche
    et un noeud de droite.
    Ainsi, le noeud racine définit l'arbre, le noeud de gauche, le sous arbre
    de gauche et le noeud de droite, le sous arbre de droite.
    '''
    def __init__(self, valeur):
        self.valeur = valeur  # la valeur du noeud, racine de l'arbre
        self.gauche = None  # le noeud racine du sous arbre gauche
        self.droite = None  # le noeud racine du sous arbre droite


    def recherche(self, valeur):
        ''' retourne True si la valeur est dans l'arbre'''
        # valeur rechechée égale à la valeur du noeud
        if valeur == self.valeur:
            return True              # on a trouvé la valeur recherchée
        # valeur rechechée plus petite que la valeur du noeud
        elif valeur < self.valeur:
            if self.gauche is None:  # plus petit et pas de sous arbre gauche
                return False
            else:
                return self.gauche.recherche(valeur) # appel récursif gauche
        # valeur rechechée plus grande que la valeur du noeud
        else:
            if self.droite is None:  # plus grand et pas de sous arbre droite
                return False
            else:
                return self.droite.recherche(valeur) # appel récursif droite


    def taille(self):
        '''retourne la taille de l'arbre (nombre de noeuds)'''
        # pas de sous arbres: un seul noeud, le noeud courant
        if self.gauche is None and self.droite is None:
            return 1
        # juste sous arbre gauche: noeud courant + taille gauche
        elif self.droite is None:
            return 1 + self.gauche.taille()
        # juste sous arbre droit: Noued courant + taille droite
        elif self.gauche is None:
            return 1 + self.droite.taille()
        # les deux sous arbres existent: Noeud courant + gauche + droite
        else:
            return 1 + self.gauche.taille() + self.droite.taille()


    def hauteur(self):
        '''retourne la hauteur de l'arbre (hauteur=0 pour juste racine ici)'''
        # pas de sous arbres
        if self.gauche is None and self.droite is None:
            return 0
        # juste sous arbre gauche
        elif self.droite is None:
            return 1 + self.gauche.hauteur()
        # juste sous arbre droit
        elif self.gauche is None:
            return 1 + self.droite.hauteur()
        # les deux sous arbres existent: on prend le max
        else:
            return 1 + max(self.gauche.hauteur(), self.droite.hauteur())


```

```python
# construction de l'arbre de 1.2
'''
                      ___21___                      0
                     /        \
                  _12_         26                    1
                 /    \       /  \
               5      17     25  28                  2
              / \    /         /   \
            1  10  14         27   30               3
             \
              3                                     4

hauteur: 4 (ou 5 si on commence à 1)
taille: 13
'''

# racine (21)
arbre = Noeud(21)
arbre.gauche = Noeud(12)
arbre.droite = Noeud(26)
# Noeud de valeur 12 (profondeur 1)
arbre.gauche.gauche = Noeud(5)
arbre.gauche.droite = Noeud(17)
# Noeud de valeur 5 (profondeur 2)
arbre.gauche.gauche.gauche = Noeud(1)
arbre.gauche.gauche.droite = Noeud(10)
# Noeud de valeur 1 (profondeur 3)
arbre.gauche.gauche.gauche.droite = Noeud(3)
# Noeud de valeur 17 (profondeur 2)
arbre.gauche.droite.gauche = Noeud(14)
# Noeud de valeur 26 (profondeur 1)
arbre.droite.gauche = Noeud(25)
arbre.droite.droite = Noeud(28)
# Noeud de valeur 28 (profondeur 2)
arbre.droite.droite.gauche = Noeud(27)
arbre.droite.droite.droite = Noeud(30)

# test de la méthode recherche
assert arbre.recherche(21)
assert arbre.recherche(12)
assert arbre.recherche(3)
assert not arbre.recherche(2)
assert not arbre.recherche(24)
assert not arbre.recherche(29)

# test de la méthode taille
assert arbre.taille() == 13

# test de la méthode hauteur
assert arbre.hauteur() == 4
arbre.gauche.gauche.gauche.droite = None
assert arbre.hauteur() == 3

################################################################################
#                              Fin de la correction                            #
################################################################################
```