```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#
# fichier: noeud.py
#    date: 2011/05/02
#
# (tous les symboles non internationaux sont volontairement omis)
#

from monome import *

class noeud(object):
  """ classe pour un noeud d'ABR representant un polynome """

  def __init__(self, monome, gauche =None, droite =None):
    """ constructeur """
    self.__monome = monome
    self.__gauche = gauche
    self.__droite = droite



  def __str__(self):
    """ afficher le monome """
    s = ""
    if self.__droite:
      s = " + " + str(self.__droite)
    s = str(self.__monome) + s
    if self.__gauche:
      s = str(self.__gauche) + " + " + s
    return s



  def chercher(self, monome):
    """ chercher un monome """
    if monome == self.__monome:
      return True
    if monome < self.__monome:
      if self.__gauche:
        return self.__gauche.chercher(monome)
      else:
        return False
    else:
      if self.__droite:
        return self.__droite.chercher(monome)
      else:
        return False



  def inserer(self, monome):
    """ inserer un monome (non existant) """
    if monome != self.__monome:
      if monome < self.__monome:
        if self.__gauche:
          self.__gauche.inserer(monome)
        else:
          self.__gauche = noeud(monome)
      else:
        if self.__droite:
          self.__droite.inserer(monome)
        else:
          self.__droite = noeud(monome)
```

```
  def plus_grand(self):
    """ plus grand monome (a droite i.e. plus faible degre) """
    if self.__droite:
      return self.__droite.plus_grand()
    else:
      return self.__monome



  def plus_petit(self):
    """ plus petit monome (a gauche i.e. plus grand degre) """
    if self.__gauche:
      return self.__gauche.plus_petit()
    else:
      return self.__monome



  def get_monome(self):
    """ accesseur du monome contenu dans le noeud """
    return self.__monome



  def set_monome(self, monome):
    """ mutateur du monome contenu dans le noeud """
    self.__monome = monome



  def fils_gauche(self):
    """ accesseur fils gauche """
    return self.__gauche



  def fils_droite(self):
    """ accesseur fils droit """
    return self.__droite



  def nombre_monomes(self):
    n = 1
    if self.__gauche:
      n += self.__gauche.nombre_monomes()
    if self.__droite:
      n += self.__droite.nombre_monomes()
    return n



  def chercher_parent(self, monome):
    """ rechercher le noeud parent d'un monome """
    if monome == self.__monome:
      return None
    if monome < self.__monome:
      if self.__gauche.__monome == monome:
        return self
      else:
        return self.__gauche.chercher_parent(monome)
    else:
      if self.__droite.__monome == monome:
        return self
      else:
        return self.__droite.chercher_parent(monome)
```

```
  def __supprimer_noeud(self, monome, parent):
    """ supprimer le noeud contenant un monome """
    if monome < self.__monome:
      self.__gauche.__supprimer_noeud(monome, parent)
    else:
      if monome > self.__monome:
        self.__droite.__supprimer_noeud(monome, parent)
      else:
        if self.__gauche is None and self.__droite is None:
          """ aucun fils """
          if parent.__gauche is self:
            parent.__gauche = None
          else:
            parent.__droite = None
        else:
          if self.__gauche is None or self.__droite is None:
            """ fils unique """
            if self.__gauche:
              t = self.__gauche
            else:
              t = self.__droite
            if parent.__gauche is self:
              s = parent.__gauche
              parent.__gauche = t
            else:
              s = parent.__droite
              parent.__droite = t
            s.__gauche = None
            s.__droite = None
          else:
            """ deux fils """
            if self.__gauche.nombre_monomes() > self.__droite.nombre_monomes():
              x = self.__gauche.plus_grand()
            else:
              x =  self.__droite.plus_petit()
            self.supprimer(x)
            self.__monome = x



  def supprimer(self, monome):
    """ supprimer un monome (existant) """
    parent = self.chercher_parent(monome)
    self.__supprimer_noeud(monome, parent)



  def retrouver(self, monome):
    """ retrouver un monome (existant) """
    if monome == self.__monome:
      return self.__monome
    if monome < self.__monome:
      if self.__gauche:
        return self.__gauche.retrouver(monome)
    else:
      if self.__droite:
        return self.__droite.retrouver(monome)



# def montrer_ordre_croissant(self):
#   """ montrer le polynome suivant les puissances croissantes """
#   if self.__gauche:
#     self.__gauche.montrer_ordre_croissant()
#   print self.__monome
#   if self.__droite:
#     self.__droite.montrer_ordre_croissant()
```

```
# def montrer_ordre_decroissant(self):
#   """ montrer le polynome suivant les puissances decroissantes """
#   if self.__droite:
#     self.__droite.montrer_ordre_decroissant()
#   print self.__monome
#   if self.__gauche:
#     self.__gauche.montrer_ordre_decroissant()



# def __liste_ordre_croissant(self, liste):
#   """ liste des monomes suivant les puissances croissantes """
#   if self.__gauche:
#     self.__gauche.__liste_ordre_croissant(liste)
#   liste.append(self.__monome)
#   if self.__droite:
#     self.__droite.__liste_ordre_croissant(liste)



# def liste_croissante(self):
#   """ accesseur liste des monomes suivant les puissances croissantes """
#   liste = []
#   self.__liste_ordre_croissant(liste)
#   return liste



  def __liste_ordre_decroissant(self, liste):
    """ liste des monomes suivant les puissances decroissantes """
    if self.__droite:
      self.__droite.__liste_ordre_decroissant(liste)
    liste.append(self.__monome)
    if self.__gauche:
      self.__gauche.__liste_ordre_decroissant(liste)



  def liste_decroissante(self):
    """ accesseur liste des monomes suivant les puissances decroissantes """
    liste = []
    self.__liste_ordre_decroissant(liste)
    return liste
```