```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec  9 15:40:28 2016

@author: stephane
"""

# Exo 1 : DONE

from cadeau_piles import tableau_aleatoire, creer_pile, est_vide, empiler,\
    depiler, affichage, chaine_vers_npi, graphe0

from random import randint

#  Exo 2 : DONE


# Exos 3, 4 et 5 : fusion


def fusion(t1, t2):
    resultat = []
    i1, i2 = 0, 0  # les indices des éléments à insérer
    n1, n2 = len(t1), len(t2)
    while i1 < n1 or i2 < n2:  # il reste au moins un élément à insérer
        if i2 == n2 or (i1 < n1 and t1[i1] <= t2[i2]):  # Test paresseux
            resultat.append(t1[i1])
            i1 += 1
        else:
            resultat.append(t2[i2])
            i2 += 1
    return resultat


def tri_fusion(t):
    if len(t) <= 1:
        return t[:]  # une copie, c'est dans le cahier des charges !
    milieu = len(t) // 2  # biais à droite
    gauche = t[: milieu]
    droite = t[milieu:]
    gtrie = tri_fusion(gauche)
    dtrie = tri_fusion(droite)
    return fusion(gtrie, dtrie)

foo = tableau_aleatoire(10)
print(foo, tri_fusion(foo))

"""
([33, 97, 6, 9, 64, 58, 56, 73, 63, 70],
 [6, 9, 33, 56, 58, 63, 64, 70, 73, 97])
"""


"""
>>> test_tri(tri_fusion, 10, 1000)
0.007473564147949219
>>> test_tri(tri_fusion, 10, 10000)
0.0827981948852539
>>> test_tri(tri_fusion, 10, 100000)
0.9785527944564819
>>> test_tri(tri_fusion, 10, 1000000)
11.786953163146972
```



```
OK, c'est légèrement sur−linéaire
"""

# Exos 6 et 7


def preparation_zone1(t, debut, fin):
    """ préparation de la zone [debut,fin], avec debut comme pivot
    le pivot est placé à la bonne position, laquelle est renvoyée"""
    i1, i2 = debut, debut  # fin des plus petits, fin de la zone traitée
    pivot = t[debut]
    while i2 < fin:
        if t[i2+1] < pivot:  # il faut le placer en fin de première zone
            t[i1+1], t[i2+1] = t[i2+1], t[i1+1]
            i1 += 1
        i2 += 1
    t[debut], t[i1] = t[i1], t[debut]
    return i1


def preparation_zone2(t, debut, fin):
    """ préparation de la zone [debut,fin], avec debut comme pivot
    le pivot est placé à la bonne position, laquelle est renvoyée"""
    i1, i2 = debut, fin+1  # fin des plus petits, début des plus grands
    pivot = t[i1]
    while i1 < i2 - 1:  # Il y a au moins un élément à traiter
        if t[i1+1] <= pivot:  # avançons la borne gauche vers la droite
            i1 += 1
        else:  # allons voir à droite
            if t[i2-1] >= pivot:  # reculons la borne droite vers la gauche
                i2 -= 1
            else:  # faisons un échange, et double-racourcissons l'intervalle
                t[i1+1], t[i2-1] = t[i2-1], t[i1+1]
                i1 += 1
                i2 -= 1
    # arrivé ici, i1 = i2-1
    t[debut], t[i1] = t[i1], t[debut]
    return i1


def tri_rapide_zone(t, debut, fin, preparation):  # quel algo de préparation ?
    """ tri en place la zone [debut,fin]"""
    if debut < fin:
        if preparation == 1:
            p = preparation_zone1(t, debut, fin)
        else:
            p = preparation_zone2(t, debut, fin)
        tri_rapide_zone(t, debut, p-1, preparation)
        tri_rapide_zone(t, p+1, fin, preparation)


def tri_rapide1(t):
    tri_rapide_zone(t, 0, len(t)-1, 1)


def tri_rapide2(t):
    tri_rapide_zone(t, 0, len(t)-1, 2)

foo = tableau_aleatoire(10)
print(foo)
tri_rapide1(foo)
```

```
print(foo)
foo = tableau_aleatoire(10)
print(foo)
tri_rapide2(foo)
print(foo)

"""
[40, 93, 72, 92, 62, 13, 44, 75, 2, 50]
[2, 13, 40, 44, 50, 62, 72, 75, 92, 93]
[12, 25, 93, 8, 15, 30, 98, 67, 18, 45]
[8, 12, 15, 18, 25, 30, 45, 67, 93, 98]


Dans un premier temps, j'avais :
>>> test_tri(tri_rapide1, 10, 1000)
0.08649899959564208
>>> test_tri(tri_rapide1, 10, 10000)
8.478795146942138


Exercice : quelle erreur avais−je pu faire pour que ce soit quadratique ?

Bon, après réparation :

>>> test_tri(tri_rapide1, 10, 1000)
0.006014251708984375
>>> test_tri(tri_rapide1, 10, 10000)
0.07456285953521728
>>> test_tri(tri_rapide1, 10, 100000)
0.9517591238021851
>>> test_tri(tri_rapide1, 10, 1000000)
12.545265817642212
>>> test_tri(tri_rapide2, 10, 1000)
0.0043240547180175785
>>> test_tri(tri_rapide2, 10, 10000)
0.056459593772888186
>>> test_tri(tri_rapide2, 10, 100000)
0.7539035081863403
>>> test_tri(tri_rapide2, 10, 1000000)
9.1167958412365478

CONCLUSION provisoire : le deuxième algorithme est un peu plus performant,
pour préparer le tableau avant la récursion.  Avec le premier algorithme, on a
des performances sensiblement identiques au tri fusion.

test_tri_monotone(tri_rapide1, 1000)
RuntimeError: maximum recursion depth exceeded

Idem pour tri_rapide2

On peut régler ce problème de récursion trop importante... mais la complexité
restera quadratique.
"""


#  Exo 8 :


"""
>>> p = creer_pile()
>>> for x in [7, 8, 12, 10, 42]:
      empiler(x, p)
```



```
>>> affichage(p)
'7 ; 8 ; 12 ; 10 ; 42'
>>> depiler(p)
42
>>> affichage(p)
'7 ; 8 ; 12 ; 10'
>>> depiler(p)
10
>>> affichage(p)
'7 ; 8 ; 12'
>>> empiler(945, p)
>>> affichage(p)
'7 ; 8 ; 12 ; 945'
"""

# Exo 9


def copie(pile):
    buf = creer_pile()   # pile temporaire
    LaCopie = creer_pile()
    while not est_vide(pile):
        empiler(depiler(pile), buf)
    while not est_vide(buf):
        x = depiler(buf)
        empiler(x, LaCopie)
        empiler(x, pile)
    return LaCopie

p3 = creer_pile()
for x in [7, 8, 12, 10, 42]:
    empiler(x, p3)
print(affichage(p3))
p4 = copie(p3)
print(affichage(p3))
print(affichage(p4))

"""
7 ; 8 ; 12 ; 10 ; 42
7 ; 8 ; 12 ; 10 ; 42
7 ; 8 ; 12 ; 10 ; 42
"""

# Exo 10


def echange(pile):
    assert not(est_vide(pile))
    x = depiler(pile)
    assert not(est_vide(pile))
    y = depiler(pile)
    empiler(x, pile)
    empiler(y, pile)

"""
>>> affichage(p3)
'7 ; 8 ; 12 ; 10 ; 42'
>>> echange(p3)
>>> affichage(p3)
'7 ; 8 ; 12 ; 42 ; 10'
"""
```

```
# Exo 11


def roll(pile, n):
    buf = creer_pile()
    assert not(est_vide(pile))
    haut = depiler(pile)
    for _ in range(n-1):
        assert not(est_vide(pile))
        empiler(depiler(pile), buf)
    empiler(haut, pile)
    for _ in range(n-1):
        empiler(depiler(buf), pile)

"""
>>> affichage(p4)
'7 ; 8 ; 12 ; 10 ; 42'
>>> roll(p4, 4)
>>> affichage(p4)
'7 ; 42 ; 8 ; 12 ; 10'
"""

# Exo 12


def superposition(pile1, pile2):  # pile1 SUR pile2... dans une autre pile
    buf = creer_pile()
    resultat = copie(pile2)
    while not(est_vide(pile1)):
        empiler(depiler(pile1), buf)
    while not(est_vide(buf)):
        x = depiler(buf)
        empiler(x, resultat)
        empiler(x, pile1)
    return resultat

"""
>>> affichage(p3)
'7 ; 8 ; 12 ; 10 ; 42'
>>> affichage(p4)
'7 ; 42 ; 8 ; 12 ; 10'
>>> affichage(superposition(p3, p4))
'7 ; 42 ; 8 ; 12 ; 10 ; 7 ; 8 ; 12 ; 10 ; 42'
>>> affichage(p3)
'7 ; 8 ; 12 ; 10 ; 42'
>>> affichage(p4)
'7 ; 42 ; 8 ; 12 ; 10'
"""


#
# Exo 13 : notation polonaise inversée
#


def evaluation(npi):  # pas de vérification sur la correction syntaxique
    operandes = creer_pile()
    for x in npi:
        if x == '+':  # éviter le oneliner dans ce qui suit
            z = depiler(operandes)
            y = depiler(operandes)
            empiler(y+z, operandes)
```



```
        elif x == '*':
            z = depiler(operandes)
            y = depiler(operandes)
            empiler(y*z, operandes)
        else:
            empiler(x, operandes)
    return depiler(operandes)  # et la pile est alors vide !


print(evaluation(chaine_vers_npi('1*((2+(3+(4+5)))*(6*((7+8)+9)))')))
"""
2016
"""

#
# Tri par insertion
#

# Exo 14 et 15 : insertion dans une pile


def inserer(x, pile):
    buf = creer_pile()
    continuer = True
    while continuer and not(est_vide(pile)):
        y = depiler(pile)
        if y <= x:
            empiler(y, pile)
            continuer = False
        else:
            empiler(y, buf)
    empiler(x, pile)
    while not(est_vide(buf)):
        empiler(depiler(buf), pile)

"""
>>> p = creer_pile()
>>> for x in [7, 8, 10, 12, 42]:
...     empiler(x, p)
...
>>> affichage(p)
'7 ; 8 ; 10 ; 12 ; 42'
>>> inserer(9, p)
>>> affichage(p)
'7 ; 8 ; 9 ; 10 ; 12 ; 42'
>>> inserer(50, p)
>>> inserer(2, p)
>>> affichage(p)
'2 ; 7 ; 8 ; 9 ; 10 ; 12 ; 42 ; 50'
"""


# Exo 16 : tri en place


def tri_insertion_en_place(pile):
    buf = creer_pile()
    while not(est_vide(pile)):
        empiler(depiler(pile), buf)
    while not(est_vide(buf)):
        inserer(depiler(buf), pile)
```

```
p = creer_pile()
for _ in range(10):
    empiler(randint(1, 50), p)
print(affichage(p))
tri_insertion_en_place(p)
print(affichage(p))

"""
4 ; 3 ; 12 ; 34 ; 10 ; 38 ; 6 ; 36 ; 42 ; 31
3 ; 4 ; 6 ; 10 ; 12 ; 31 ; 34 ; 36 ; 38 ; 42
"""

# Exo 17 : tri extérieur


def tri_insertion_exterieur(pile):
    buf = creer_pile()
    res = creer_pile()
    while not(est_vide(pile)):
        empiler(depiler(pile), buf)
    while not(est_vide(buf)):
        x = depiler(buf)
        empiler(x, pile)
        inserer(x, res)
    return res

p = creer_pile()
for _ in range(10):
    empiler(randint(1, 50), p)
print(affichage(p))
pt = tri_insertion_exterieur(p)
print(affichage(p))
print(affichage(pt))

"""
21 ; 15 ; 18 ; 36 ; 13 ; 27 ; 14 ; 47 ; 33 ; 29
21 ; 15 ; 18 ; 36 ; 13 ; 27 ; 14 ; 47 ; 33 ; 29
13 ; 14 ; 15 ; 18 ; 21 ; 27 ; 29 ; 33 ; 36 ; 47
"""

#
# Exo 18 et 19 : parcours d'un graphe
#


def parcours(s0, graphe):  # liste des voisins; S = [0,...,n-1]
    todo = creer_pile()  # aussitôt empilés, les sommets sont déclarés traités
    vus = [False] * len(graphe)
    empiler(s0, todo)
    vus[s0] = True
    visites = [s0]
    while not est_vide(todo):
        s = depiler(todo)
        for v in graphe[s]:
            if not(vus[v]):
                vus[v] = True
                empiler(v, todo)
                visites.append(v)
    return visites

for origine in [0, 9, 5]:
    print(origine, parcours(origine, graphe0))
```



```
"""
(0, [0, 8, 3, 4, 9, 1, 2, 6, 10])
(9, [9, 3, 4, 8, 0, 1, 2, 6, 10])
(5, [5, 7])
"""
```