Corrigé du TP 10 - Tableaux 2

Chenevois-Jouhet-Junier

```
# -*- coding: utf-8 -*-
```

1 Exo 1

```
def inversions(t):
    '''retourne le nombre de couples du tableau t tels que i<j et t[i]>t[j]'''
    n = 0
    for i in range(len(t)):
        for j in range(i+1,len(t)):
            if t[j]-t[i] < 0:
                n += 1
    return n

"""
>>> inversions([5,1,2,4,3])
5
"""
```

Il y a $(n-1)+(n-2)+...+1 = \frac{(n-1)n}{2}$ comparaisons dans la boucle interne donc la complexité est **quadratique** (triangulaire), en $O(n^2)$.

2 Exo 2

```
def indice_maxi(t):
    '''retourne l'indice d'un élément maximal'''
    imax = 0
    maxi = t[imax]
    for i in range(1,len(t)):
        if t[i] > maxi:
            maxi, imax = t[i], i
    return i

"""
Il y a $n-1$ comparaisons (un parcours de tableau) donc la complexité est \textbf{linéaire} en $O(n)$.
"""

"""
# Exo 3
"""

def indices_difference_maxi(t):
    '''retourne un couple (i,j) tel que i>=j et abs(ti-tj) maximal'''
```

```
    couple = (0,0)
    maxi = 0
    for j in range(len(t)):
        for i in range(j+1,len(t)):
            ecart = abs(t[i]-t[j])
            if ecart > maxi:
                couple = (i,j)
                maxi = ecart
    return couple

"""
Il y a $(n-1)+(n-2)+...+1 = \frac{(n-1)n}{2}$ comparaisons dans la boucle interne donc la complexité est \textb
"""

"""
# Exo 4
"""

def undoublon(t):
    '''retourne True si t contient au moins un doublon et False sinon'''
    for j in range(len(t)):
        for i in range(j+1,len(t)):
            if t[i] == t[j]:
                return True
    return False

"""
    >>> undoublon([10,2,5,23,2])
    True
    >>> undoublon([10,2,5,23,12])
    False
    >>> undoublon([23,2,5,23,12])
    True
    >>> undoublon([2,2,2])
    True
"""

"""
Complexité \textbf{quadratique} (triangulaire) dans le pire des cas.
"""

"""
# Exo 5
"""

def doublons(t):
    '''retourne la liste des doublons'''
    L = []
    for i in range(len(t)):
        for j in range(i+1,len(t)):
            if t[i] == t[j]:
                L.append((i,j))
    return L

"""
    >>> doublons([2,5,2,42])
    [(0, 2)]
    >>> doublons([2,5,3,42])
    []
"""
```

```
"""
Complexité \textbf{quadratique} (triangulaire).
"""

"""
# Exo 6 Évaluation de polynômes
"""

P0 = [4, -3, 1, 5]

def evaluation(P,t):
    '''Evaluation naive du polynome P en t'''
    s=0 # la somme provisoire
    for i in range(len(P)):
        s = s + P[i]*t**i
    return s

def evaluation_meilleure(P,t):
    '''Amélioration de l'évaluation d'un polynome P en t'''
    s=0 # la somme provisoire
    p = 1  #la puissance de t provisoire
    for i in range(len(P)):
        s = s + P[i]*p
        p = p*t
    return s

def horner(P,t, verbose=False):
    '''Evaluation d'un polynome P en t avec l'algorithme de Horner'''
    s = 0 # la somme provisoire
    for i in range(-1,-len(P)-1,-1):
        if verbose:
            print('--'*(len(P) + 1 + i)+ '>travail sur ({}, {})'.format(P[:len(P)+i+1], t))
        s = s*t + P[i]
    return s

def hornerec(P,x):
    '''Une version récursive de l'algorithme de Horner. Les appels de fonction
     se font dans l'ordre inverse de la version itérative'''
    if len(P) == 1:
        print('--'*(len(P))+ '>appel de hornerec({}, {})'.format(P, x))
        print('--'*(len(P))+ '>sortie de hornerec({}, {})'.format(P, x))
        return P[0]
    print('--'*(len(P))+ '>appel de hornerec({}, {})'.format(P, x))
    res = P[0] + x*hornerec(P[1:], x)
    print('--'*(len(P))+ '>sortie de hornerec({}, {})'.format(P, x))
    return res

"""
    >>> [f(P0, 2) for f in [evaluation, evaluation_meilleure, horner]]
    [42, 42, 42]
    >>> horner(P0, 2, verbose=True)
    -------->travail sur ([4, -3, 1, 5], 2)
    ------>travail sur ([4, -3, 1], 2)
    ---->travail sur ([4, -3], 2)
    -->travail sur ([4], 2)
    42
    >>> hornerec(P0, 2)
    -------->appel de hornerec([4, -3, 1, 5], 2)
    ------>appel de hornerec([-3, 1, 5], 2)
    ---->appel de hornerec([1, 5], 2)
    -->appel de hornerec([5], 2)
```

```
    -->sortie de hornerec([5], 2)
    ---->sortie de hornerec([1, 5], 2)
    ------>sortie de hornerec([-3, 1, 5], 2)
    -------->sortie de hornerec([4, -3, 1, 5], 2)
    42
"""

"""Complexité

* Avec `evaluation`, pour  un polynome de degré $n$ on effectue:

    * $n+1$ additions
    * $1+2+3+...+n+(n+1) = \frac{(n+1)(n+2)}{2}$ produits

Soit un total de $\frac{(n+1)(n+3)}{2}$ opérations. La complexité est donc __quadratique__ , en $O(n^2)$.

* Avec `evaluation_meilleure`, pour un polynome de degré $n$ on effectue :

    * $n+1$ additions
    * et $2(n+1)$ produits

Soit un total de $3(n+1)$ opérations.  La complexité est donc __linéaire__ , en $O(n)$.

* Avec `horner`, pour un polynome de degré $n$ on effectue :

    * $n+1$ additions
    * et $n+1$ produits

Soit un total de $2(n+1)$ opérations. La complexité est donc __linéaire__ , en $O(n)$.

"""

"""
# Exo7
"""

"""
Ci-dessous des liens permanent vers des simulations de représentations de matrices avec [Python Tutor](http://w

* Exemple de l'exo 7: <http://pythontutor.com/visualize.html#code=t1+%3D+%5B%5B0%5D+*+3%5D+*+2%0At2+%3D+%5B%5B0

* Un autre exemple : <http://pythontutor.com/visualize.html#code=L+%3D+%5B0,0,0%5D*4%0AN+%3D+%5B%5B0,0,0%5D%5D*
"""

"""
# Exo8 Matrice nulle et copie de matrice
"""

def matrice_nulle(n,p):
    '''Retourne la matrice nulle de n lignes et p colonnes'''
    return [[0 for j in range(p)] for i in range(n)]
    e

def dimensions(m):
    '''dimensions d'une matrice'''
    return len(m),len(m[0])

def copie(m):
    '''retourne une copie de la matrice m'''
    nlines,ncols = dimensions(m)
    cp = matrice_nulle(nlines,ncols) # matrice copie
```

```
    for i in range(nlines):
        for j in range(ncols):
            cp[i][j]=m[i][j]
    return  cp


"""
    >>> m1 = matrice_nulle(3, 2) : i
    >>> m1
    [[0, 0], [0, 0], [0, 0]]
    >>> dimensions(m1)
    (3, 2)
    >>>
    >>> m1[1][1] = 3
    >>> m1
    [[0, 0], [0, 3], [0, 0]]
    >>> m2
    [[0, 0], [0, 0], [0, 0]]
"""

"""
# Exo9 Addition de matrices
"""

def addition(m1, m2):
    """retourne la matrice somme de deux matrices m et n"""
    assert dimensions(m1) == dimensions(m2), "Les matrices n'ont pas la même dimension"
    nlines, ncols = dimensions(m1)
    s = matrice_nulle(nlines,ncols)
    for i in range(nlines):
        for j in range(ncols):
            s[i][j] = m1[i][j]+m2[i][j]
    return s

def addition2(m1,m2):
    n, p = dimensions(m1)
    assert (n,p) == dimensions(m2),"Les matrices n'ont pas la même dimension"
    return [[m[i][j] + m2[j][j] for j in range(p)] for i in range(n)]

"""
\newpage
"""

"""
# Exo 10 Multiplication d'une matrice par un scalaire
"""

def multiplication_externe(m, alpha):
    nlines, ncols = dimensions(m)
    s = matrice_nulle(nlines, ncols)
    for i in range(nlines):
        for j in range(ncols):
            s[i][j] = m[i][j]*alpha
    return s

def multiplication_externe2(m, alpha):
    '''multiplie tous les coefficients de la matrice m par le scalaire alpha'''
    return [[m[i][j]*alpha for j in range(len(m[0]))] for i in range(len(m))]

"""
# Exo11 Transposition d'une matrice
```

```
"""


def transposition(m):
    """retourne la transposée d'une matrice"""
    nlines,ncols = dimensions(m)
    s = matrice_nulle(ncols, nlines)
    for i in range(ncols):
        for j in range(nlines):
            s[i][j] = m[j][i]
    return s


def transposition2(m):
    return [[m[j][i] for j in range(len(m))] for i in range(len(m[0]))]

"""
# Exo 12 Multiplication de deux matrices
"""

def multiplication(m,n):
    '''retourne la matrice poduit de m par n'''
    mlines,mcols = dimensions(m)
    nlines,ncols = dimensions(n)
    assert mcols == nlines, "Matrices incompatibles"
    p = matrice_nulle(mlines,ncols)
    for i in range(mlines):
        for j in range(ncols):
            for k in range(mcols): # ou nlines c'est pareil
                p[i][j] = p[i][j] + m[i][k]*n[k][j]
    return p


def multiplication2(m,n):
    '''retourne la matrice poduit de m par n'''
    assert dimensions(m)[1]==dimensions(n)[0], "Matrices incompatibles"
    return [[sum([m[i][k]*n[k][j] for k in range(len(m[0]))])
    for j in range(len(n[0]))] for i in range(len(m))]

"""
# Exo13 Matrice identité et puissance d'une  matrice
"""

def identite(n):
    return [[i == j and 1 or 0  for j in range(n)] for i in range(n)]


def puissance(A, q):
    n, p = dimensions(A)
    assert n == p
    B = identite(n)
    for i in range(q):
        B = multiplication(B, A)
    return B

"""
# Exo 14 Complexité des opérations matricielles
"""

"""
```

```
* Addition de deux matrices de dimensions $n \times p$  :

    * $np$  additions de scalaires
    * $O$ multiplication de scalaires
    * Complexité en $O(np)$

* Multiplication d'une matrice de dimension $n \times p$ par un scalaire :

    * $O$ addition de scalaires
    * $np$ multiplications de scalaires
    * Complexité en $O(np)$

* Transposition d'une matrice de dimension $n \times p$ :

    * $O$ addition de scalaires
    * $O$ multiplication de scalaires
    * $np$ affectations dans un nouveau à tableaux à deux dimensions représentant une matrice de dimensions $p
    * Complexité en $O(np)$

* Multiplication de deux matrices de dimensions $n \times p$ et $p \times q$ :

    * $np(q-1)$ additions de scalaires
    * $npq$ multiplications de scalaires
    * Complexité en $O(npq)$

* Puissance de matrice $A^q$ avec A de dimension $n \times n$ :

    * $qn^2(n-1)$ additions de scalaires
    * $qn^3$ multiplications de scalaires
    * Complexité en $O(qn^3)$

"""

"""
# Exo 15
"""

def est_symetrique(m):
    '''Retourne un booleen indiquant si la matrice m est symetrique'''
    n, p = dimensions(m)
    if n != p:
        return False
    for i in range(n):
        for j in range(i):
            if m[i][j] != m[j][i]:
                return False
    return True

def est_symetrique2(m):
    n, p = dimensions(m)
    if n != p:
        return False
    return m == transposition(m)

"""
Complexité :


* pour `est_symetrique` : dans le pire des cas (matrice symétrique) il faut faire $1 + 2 + \cdots + n - 1 = \fr
```

```
* pour `est_symetrique2` : $2n^2$ affectations pour créer une matrice
nulle de meme dimensions puis la transposée puis $n^2$ comparaisons pour comparer la matrice initiale et sa tra


Les deux complexités sont quadratiques, en $O(n^2)$ mais les constantes sont bien plus petites pour la fonction
"""

"""
# Exo 16
"""

def pos_maxi_j_premiers(t,k):
    '''Écrire une fonction prenant en entrée un tableau T et un entier k
     majoré par la longueur de T, et renvoyant un indice i0 tel que T[i0]
     est maximal parmi T[0:k] (donc parmi les k premiers éléments T [0],
     ... T [k - 1])'''
    assert k <= len(t)
    imax, maxi = 0,t[0]
    for i in range(1, k):
         if t[i] > maxi:
             imax, maxi = i, t[i]
    return imax

"""
    >>> pos_maxi_j_premiers([3, 42, -5, 8, 50, 112], 4)
    1
"""

"""
# Exo 17
"""

"""
    i0 <- indice_du max(t,6) = 2  (au sens : parmi les 6 premiers éléments)
    échange des positions 2/5; t = [3, 42, -5, 50, 8, 112]
    i0 <- indice du max(t,5) = 3
    échange des positions 3/4; t = [3, 42, -5, 8, 50, 112]
    i0 <- indice du max(t,4) = 1
    échange des positions 1/3; t = [3, 8, -5, 42, 50, 112]
    i0 <- indice du max(t,3) = 1
    échange des positions 1/2; t = [3, -5, 8, 42, 50, 112]
    i0 <- indice du max(t,2) = 0
    échange des positions 0/1; t = [-5, 3, 8, 42, 50, 112]
"""

"""
# Exo 18 Échange sur place de deux éléments d'un tableau
"""

def echange(t,i,j):
    '''Ecrire une fonction réalisant en place (sans rien renvoyer)
    l'échange de deux éléments d'un tableau '''
    t[i],t[j] = t[j],t[i]

"""
    >>> t0 = [3, 42, -5, 8, 50, 112]
    >>> echange(t0, 1, 3)
    >>> t0
    [3, 8, -5, 42, 50, 112]
"""
```

```
"""
# Exo 19 Tri par sélection
"""

def tri_selection(t):
    '''Écrire un programme réalisant le tri d'un tableau.
    Le tri se fera obligatoirement en place, c'est-à-dire en modifiant
    le tableau donné en entrée, sans rien renvoyer.'''
    for i in range(len(t)-1,0,-1):
        echange(t,i,pos_maxi_j_premiers(t,i+1))


"""
    >>> t1 = [112, 3, 42, -5, 8, 50]
    >>> tri_selection(t1)
    >>> t1
    [-5, 3, 8, 42, 50, 112]
    >>>

Deux pages web proposant des simulateurs de l'algorithme de tri par sélection :

* <https://interstices.info/jcms/c_6973/les-algorithmes-de-tri>
* <http://www.sorting-algorithms.com/>

"""



"""
# Exo 20 Complexité du tri par sélection
"""

"""
* Chaque appel de `pos_maxi_j_premiers(t, j)` va induire $j-1$ comparaisons.

* Globalement, `tri_selection(t)` va donc effectuer $(n-1)+(n-2)+ \cdots +1=\frac{n(n-1)}{2}$ comparaisons.

Il s'agit donc d'une complexité quadratique en $O(n^2)$.
\newpage.
"""

"""
# Un peu de programmation dynamique
"""

"""
## Exo 21, Projet Euler problème 15


![Projet Euler problème 15](p_015.jpg "parcours")
"""

def routes(n):
    '''Probleme  15 projet Euler  https://projecteuler.net/problems'''
    #on place des 1 sur les bords supérieurs et gauches du tableau
    tab = [[0+int(i==0 or j==0) for i in range(n+1)] for j in range(n+1)]
    for line in range(1,n+1):
        for col in range(1,n+1):
            tab[line][col] = tab[line-1][col]+tab[line][col-1]
    return tab
```

```
"""
    >>> routes(2)[-1][-1]
    6
    >>> routes(20)[-1][-1]
    137846528820
"""

"""
## Exo 22 Project Euler problème 67, recupération du triangle
"""

def exo22(fichier):
    triangle = []
    f  = open(fichier,'r')
    for ligne in f:
        row = ligne.rstrip().split(' ')
        row = list(map(int, row))
        triangle.append(row)
    f.close()
    return triangle

"""
    >>> triangle = exo22('triangle.txt')
    >>> len(triangle)
    100
    >>> triangle[0][0]
    59
"""

"""
##  Exo 23 Project Euler problème 67, chemin minimal
"""

def exo23(triangle):
    tab = [[(triangle[0][0], None)]]
    for i in range(1, len(triangle)):
        newline = [(tab[i - 1][0][0] + triangle[i][0], 0)]
        longlignepreced = len(triangle[i]) - 1
        for j in range(1, longlignepreced):
            valcourant = triangle[i][j]
            maxi, indexmaxi = tab[i - 1][j -1][0] + valcourant, j - 1
            maxpreced = tab[i - 1][j][0]
            if maxpreced + valcourant > maxi:
                maxi, indexmaxi = maxpreced + valcourant, j
            newline.append((maxi, indexmaxi))
        newline.append((tab[i - 1][-1][0] + triangle[i][- 1], longlignepreced))
        tab.append(newline)
    tab[-1].sort(key=lambda t : t[0])
    return (tab[-1][-1][0])

"""
    >>> exo23(triangle)
    7273
"""

"""
## Exo 24 Projet Euler 81, récupération de la matrice
"""

def exo24(fichier):
```

```
    '''Retourne la matrice carrée de taille 80*80 contenue dans fichier
    sous la forme d'une liste de listes'''
    f = open(fichier,'r')
    mat = [[int(i) for i in ligne.rstrip().split(',')] for ligne in f]
    f.close()
    return mat

"""
    >>> matrice = exo24('matrix.txt')
    >>> len(matrice)
    80
    >>> len(matrice[0])
    80
    >>> matrice[0][0]
    4445
"""

"""
## Exo 25 Projet Euler 81, chemin minimal
"""

def exo25(mat):
    '''Retourne le chemin minimal selon la définition du problème 81 du
    projet Euler , pour lma matrice mat (liste de listes)'''
    nrow,ncols = len(mat),len(mat[0])
    #matpoids[i][j]  est le poids du cheminpour atteindre mat[i][j]
    #mat[0][i] est le cumul des mat[0][j] avec j<=i
    matpoids = [[mat[0][0]]]
    for k in range(1,ncols):
        matpoids[0].append(matpoids[0][-1]+mat[0][k])
    for i in range(1,nrow):
        #mat[i][0] est le cumul des mat[j][0] avec j<=i
        matpoids.append([matpoids[i-1][0]+mat[i][0]])
        for j in range(1,ncols):
            matpoids[i].append(mat[i][j]+min(matpoids[i][-1],matpoids[i-1][j]))
    return matpoids[-1][-1]

"""
    >>> exo25(matrice)
    427337
"""
```