Corrigé du TP 10 - Tableaux 2

Chenevois-Jouhet-Junier

1 Parcours de tableaux

1.1 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 $O(n^2)$.

1.2 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 **linéaire** en $O(n)$.

1.3 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 **quadratique** (triangulaire) en $O(n^2)$.

1.4 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
"""
```

La fonction `undoublon(t)` est de complexité **quadratique** (triangulaire) dans le pire des cas. Si l'on connaît la plage des valeurs de `t`, on peut déterminer si `t` possède un doublon en le parcourant et en mettant à jour un un tableau ees éléments déjà vus. La complexité est alors **linéaire**.

```
def undoublon_lineraire(t, n, m):
    '''retourne True si t contient au moins un doublon et False sinon.
    , les valeurs de t étant des entiers compris entre n et m '''
    dejavu = [False] * (m - n  + 1)
    for e in t:
        if dejavu[e - n] :
            return True
        dejavu[ e - n ] = True
    return False
```

1.5 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 **quadratique** (triangulaire).

1.6 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 $O(n)$.

2 Tableaux à deux dimensions

2.1 Exo7

Ci-dessous des liens permanent vers des simulations de représentations de matrices avec Python Tutor :

- Exemple de l'exo 7: http://pythontutor.com/visualize.html#code=t1+%3D+%5B%5B0%5D+*+3%5D+*+2%0At2+%3D+%5B%5B0%5D+*+3+for+_+in+range(2)%5D%0A%0At1%5B1%5D%5B1%5D+%3D+42%0At2%5B1%5D%5B1%5D+%3D+42%0A%0At1,+t2&mode=display&origin=opt-frontend.js&cumulative=false&heapPrimitives=false&drawParentPointers=false&textReferences=false&showOnlyOutputs=false&py=3&rawInputLstJSON=%5B%5D&curInstr=0
- Un autre exemple : http://pythontutor.com/visualize.html#code=L+%3D+%5B0,0,0%5D*4%0AN+%3D+%5B%5B0,0,0%5D%5D*4%0AN%5B1%5D%5B1%5D+%3D+2%0AM+%3D+%5B%5B0,0,0%5D+for+i+in+range(4%29%5D%0AM%5B1%5D%5B1%5D+%3D+2%0AP+%3D+%5B%5B0+for+i+in+range(3%29%5D+for+j+in+range(4%29%5D%0AP%5B1%5D%5B1%5D+%3D+2%0AQ+%3D+%5B%5B0%5D*3+for+j+in+range(4%29%5D%0AR+%3D+%5B%5D%0Afor+nligne+in+range(4%29%3A%0A++++ligne+%3D+%5B%5D%0A++++for+ncol+in+range(3%29%3A%0A++++++++ligne.append(0%29%0A++++R.append(ligne%29&mode=display&origin=opt-frontend.js&cumulative=false&heapPrimitives=false&textReferences=false&py=3&rawInputLstJSON=%5B%5D&curInstr=0

2.2 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

def copie2(m):
    '''retourne une copie de la matrice m'''
    nlines,ncols = dimensions(m)
    cp = []  # matrice copie
    for i in range(nlines):
        ligne = []
        for j in range(ncols):
            ligne.append(m[i][j])
        cp.append(ligne)
    return  cp

def copie3(m):
    '''retourne une copie de la matrice m'''
    return [ligne[:] for ligne in m]

def copie_superficielle(m):
    '''retourne une copie superficielle de la matrice m, pas ce qu'on veut'''
    return [ligne for ligne in m]

"""
>>> 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]]
"""
```

3 Exo 9 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)]
```

3.1 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))]
```

3.2 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]))]
```

3.3 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))]
```

3.4 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
```

3.5 Exo 14 Complexité des opérations matricielles

- Addition de deux matrices de dimensions $n \times p$:
  - np additions de scalaires
  - 0 multiplication de scalaires
  - Complexité en $O(np)$
- Multiplication d'une matrice de dimension $n \times p$ par un scalaire :
  - 0 addition de scalaires
  - np multiplications de scalaires
  - Complexité en $O(np)$
- Transposition d'une matrice de dimension $n \times p$:
  - 0 addition de scalaires
  - 0 multiplication de scalaires
  - np affectations dans un nouveau à tableaux à deux dimensions représentant une matrice de dimensions $p \times n$
  - 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 $O(qn^3)$

3.6 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 $1 + 2 + \cdots + n - 1 = \frac{n(n-1)}{2}$ comparaisons.
- 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 transposée.

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

4 Extrait du sujet de Centrale 2016

4.1 Exo 16

- Question 1 :

```
conflit = [ [  0,  0,  0,100,100,  0,  0,150,  0],
            [  0,  0,  0,  0,  0, 50,  0,  0,  0],
            [  0,  0,  0,  0,200,  0,  0,300, 50],
            [100,  0,  0,  0,  0,  0,400,  0,  0],
            [100,  0,200,  0,  0,  0,200,  0,100],
            [  0, 50,  0,  0,  0,  0,  0,  0,  0],
            [  0,  0,  0,400,200,  0,  0,  0,  0],
            [150,  0,300,  0,  0,  0,  0,  0,  0],
            [  0,  0, 50,  0,100,  0,  0,  0,  0] ]

def nb_conflits():
    n = len(conflit)
    nb = 0
    for i in range(n - 1):
        for j in range(i + 1, n):
            if conflit[i][j] != 0:
                nb += 1
    return nb
```

- Question 2 :

  Cette fonction est de complexité temporelle : $3n - 1 + 3n - 2 + \ldots + 1 = \frac{(3n-1)3n}{2} = O(n^2)$

  Il s'agit d'une **complexité quadratique**.

4.2 Exo 17

- Question 1 : **nombre de vols par niveau relatif**

```
def nb_vol_par_niveau_relatif(regulation):
    cout = [0]*3
    for r in regulation:
        cout[r] += 1
    return cout

"""
In [14]: nb_vol_par_niveau_relatif([0, 1, 1])
Out[14]: [1, 2, 0]
"""
```

- Question 2 :

- Question 2 a) **Coût d'une régulation**, voir fonction `cout_regulation` ci-dessous.
- Question 2 b) Complexité de la fonction `cout_regulation` en fonction du nombre de sommets n

  La fonction `cout_regulation` effectue $n-1$ tours de boucles externes.

  Chaque tour de boucle externe comprend $n-k-1$ tours de boucles internes.

  Chaque tour de boucle interne comprend une affectation.

  La complexité de `cout_regulation` est donc en $n-1+n-2+\ldots+1=\frac{n(n-1)}{2}=O(n^2)$, elle est **quadratique**.
- Question 2 c) : coût de la régulation pour laquelle chaque avion vole à son RFL, voir fonction `cout_RFL()` ci-dessous.

```python
def cout_regulation(regulation):
    cout = 0
    n = len(regulation)
    sommet = [3*k + regulation[k] for k in range(n)]
    for k in range(n - 1):
        s = sommet[k]
        for j in range(k + 1, n):
            cout += conflit[s][sommet[j]]
    return cout

"""
In [19]: cout_regulation([0, 1, 1])
Out[19]: 250
"""
def cout_RFL():
    return cout_regulation([0]*(len(conflit)//3))

"""
In [21]: cout_RFL()
Out[21]: 500
"""
```

- Question 3 :

  Pour n vols, il existe 3^n régulations possibles qui sont les $n-listes$ d'élements pris dans l'ensemble `{0,1,2}`.

  Il n'est pas envisageable de calculer les couts de toutes les régulations possibles pour trouver celle de cout minimal, car cet algorithme aurait une complexité en $O(n^2 3^n)$.

4.3 Exo 18 Algorithme de coût minimal

- Question 1 : coû d'un sommet

```python
def cout_du_sommet(s, etat_sommet):
    cout = 0
    adjacents = conflit[s]
    for k in range(len(etat_sommet)):
        if etat_sommet[k] != 0:
            cout += adjacents[k]
    return cout

"""
In [23]: cout_du_sommet(8, [1, 0, 0, 0, 1, 0, 2, 2, 2])
Out[23]: 100
"""
```

- Question 2 :

  La complexité de la fonction `cout_du_sommet` est **linéaire**, en $3n = O(n)$, où n est le nombre de sommets.
- Question 3 : fonction `sommet_de_cout_min(etat_sommet)` qui retourne le numéro du sommet de coût minimal parmi les sommets qui n'ont pas encore été choisis ou supprimés.

```python
def sommet_de_cout_min(etat_sommet):
    cout_min = None
    index_min = None
    for k in range(len(etat_sommet)):
        s = etat_sommet[k]
        if s == 2:
            c = cout_du_sommet(k, etat_sommet)
            if cout_min == None or c < cout_min:
                cout_min = c
                index_min = k
    return index_min

"""
In [26]: sommet_de_cout_min([1, 0, 0, 0, 1, 0, 2, 2, 2])
Out[26]: 8
"""
```

- Question 4 :

  La complexité de la fonction `sommet_de_cout_min` est **quadratique** en $3n \times O(n) = O(n^2)$.

- Question 5 : fonction `minimal`

```python
def minimal():
    m = len(conflit)
    n = m//3
    etat_sommet = [2]*m
    regulation = [0]*n
    for k in range(n):
        indexmin = sommet_de_cout_min(etat_sommet)
        q = indexmin // 3
        regulation[q] = indexmin%3
        for j in range(3):
            index = 3*q + j
            if index != indexmin:
                etat_sommet[index] = 0
            else:
                etat_sommet[index] = 1
    return regulation

"""
In [28]: minimal()
Out[28]: [1, 0, 1]

In [29]: cout_regulation([1, 0, 1])
Out[29]: 0
"""
```

- Question 6 :

  La complexité de la fonction `minimal` est de $n \times O(n^2) = O(n^3)$ en fonction du nombre n de sommets du graphe.

4.4 Exo 19 : Recuit simulé

- Question 1 :

```python
import random, math

def recuit(regulation):
    T = 1000
    m = len(conflit)
    n = m//3
    cout = cout_regulation(regulation)
```

```
    while T >= 1:
        k = random.randint(0, n - 1)
        rk = regulation[k]
        mk = [0,1,2]
        del(mk[rk])
        newrk = mk[random.randint(0, 1)]
        regulation[k] = newrk
        newcout = cout_regulation(regulation)
        deltac = newcout - cout
        # mise à jour du cout minimal si le nouveau cout est inférieur
        # sinon on rétablit l'ancienne version
        if newcout < cout or random.random() < math.exp(-deltac/T):
            cout = newcout
        else:
            regulation[k] = rk
        T = T * 0.99    #on diminue T de 1 %
    return regulation

"""
In [38]: recuit([0,0,0])
Out[38]: [0, 2, 0]

In [39]: cout_regulation([0, 2, 0])
Out[39]: 0

In [40]: sum([cout_regulation(recuit([0,0,0])) for i in range(100)])/100
Out[40]: 0.0
"""
```

5 Un peu de programmation dynamique

5.1 Exo 20, Projet Euler problème 15

```
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
"""
```

5.2 Exo 21 Project Euler problème 67, recupération du triangle

```
def exo21(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 = exo21('triangle.txt')
>>> len(triangle)
100
>>> triangle[0][0]
59
"""
```

5.3 Exo 22 Project Euler problème 67, chemin minimal

```
def exo22(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])

"""
>>> exo22(triangle)
7273
"""
```

5.4 Exo 23 Projet Euler 81, récupération de la matrice

```
def exo23(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 = exo23('matrix.txt')
>>> len(matrice)
80
>>> len(matrice[0])
80
>>> matrice[0][0]
4445
"""
```

5.5 Exo 24 Projet Euler 81, chemin minimal

```python
def exo24(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]

"""
>>> exo24(matrice)
427337
"""
```