CNC 2017 CORRIGE

PARTIE I

Q1 : Ecrire en algèbre relationnelle, une requête qui donne pour résultat: les noms des fichiers dont la taille originale est comprise entre 1 Kilo-octet et 1 Méga-octet.

$\pi_{nomF}(\sigma_{tailleF>1000}(\sigma_{tailleF<1000000}))\pi_{nomF}(\sigma_{tailleF>1000}(\sigma_{tailleF<1000000}))$

Q2 : Ecrire une requête SQL, qui donne pour résultat: les noms et les tailles des fichiers textes, dont le nom se termine par: .doc ou .docx, triés dans l'ordre alphabétique des noms des fichiers.

```
SELECT nomF, tailleF

FROM Fichier

WHERE nomF like "%.doc" OR nomF like "%.docx"

ORDER BY nomF;
```

Q3 : Ecrire une requête SQL qui donne pour résultat: les noms des fichiers compressés, les formats de compression, les types de compression, et le taux de compression de chaque fichier, dont le taux de compression dépasse 40%, triés dans l'ordre des numéros des fichiers.

```
SELECT F.nomF, C.format,A.type,(1-cast(C.tailleC as float)/F.tailleF)**100 AS taux

FROM Algorithme A JOIN Fichier F JOIN Compression C ON A.format=C.format AND
C.numeroF=F.numeroF

WHERE taux>=40

ORDER BY C.numeroF;
```

Q4 : Ecrire une requête SQL qui donne pour résultat: les algorithmes sans perte de données et le compte des fichiers compressés par chacun de ces algorithmes, dont ce compte est égal à 3, 5 ou 8.

```
SELECT A.format, COUNT(**)

FROM Algorithme A JOIN Compression C ON A.format=C.format

WHERE type=0

GROUP BY A.format
HAVING COUNT(**) IN (3,5,8);
```

Q5 : Ecrire une requête SQL qui donne pour résultat: Les 3 premiers grands taux de compressions,triés dans l'ordre croissant.

```
SELECT taux

FROM (SELECT (1-cast(tailleC as float)/tailleF) * 100 AS taux

      FROM Compression C JOIN Fichier F ON C.numeroF=F.numeroF


      ORDER BY taux DESC


      Limit 3)
```

ORDER BY taux ASC

Q6 : Ecrire un programme python qui saisi un format de compression, et qui affiche le taux moyen des taux des compressions de ce format, si le format saisi existe dans la base de données. Si le format saisi n'existe pas, le programme doit afficher le message: " Le format saisi n'existe pas dans la BD".

In [1]:

```
import sqlite3
db=sqlite3.connect("C:/disque dur secours/valise
professeur/MEKNES/documents pédagogiques/spe/PREPARATION AU
CONCOURS/corriges cnc 2010 2015/cnc2017.db")
cur=db.cursor()
cur.execute(""" SELECT format
                FROM Algorithme;""")
L=cur.fetchall()
L=[L[i][0] for i in range(len(L))]
cur.close()
format=input("saisir le format de compression: ")
if format in L:
    cur1=db.cursor()
    cur1.execute(""" SELECT AVG((1-cast(tailleC as float)/tailleF) * 100)
                     FROM Compression C JOIN Fichier F ON
C.numeroF=F.numeroF
                     WHERE C.format=?""",(format,))
    moy=cur1.fetchone()[0]
    print("Le taux moyen de compression du format", format,"est:","{:.2f}
%".format(moy))
    cur1.close()
    db.close()
else:
    print("Le format saisi n'existe pas dans la BD")
```

```
saisir le format de compression: zip
Le taux moyen de compression du format zip est: 43.27 %
```

PARTIE 2

II.1 Quel est le plus grand nombre entier positif, avec bit de signe, qu'on peut coder sur 12 bits? justifier votre réponse.

Les entiers sont codés en compléments à 2, par conséquent pour un codage sur 12 bits l'intervalle des valeurs sera $-2_{11}-211$ et $2_{11}-1211-1$. Donc le plus grand nombre entier positif avec bit de signe codable sur 12 bits est $2_{11}-1211-1$.

II.2.a) Ecrire la fonction : binaire(N), qui reçoit en paramètre un entier naturel N, et qui retourne une chaine de caractères qui contient la représentation binaire de N. Le bit du poids le plus fort se trouve à la fin de la chaine. Ne pas utiliser la fonction prédéfinie bin() de Python.

In [2]:

```
def binaire(N):
    if N==0:return("0")
    ch=""
    while N!=0:
        ch+=str(N%2)
        N=N//2
```

```
    return(ch)
print(binaire(23))
```

```
11101
```

II.2.b) Déterminer la complexité de la fonction binaire(N).
Le coût de la fonction binaire(N) est celui de la boucle while, celle ci itérera autant de fois que le nombre de bits de la représentation binaire de N, or le nombre de bits de N correspond à $\log_2(N)+1$log2(N)+1. A chaque itération de la boucle while un bloc constant s'exécute composé de deux affectations, une concaténation, un calcul de quotient, un calcul de reste et un appel à la fonction str dont le cout est c considéré comme étant constant indépendant de N, s'ajoute à ce coût un test et une affectation avant la boucle, donc en considérant N la taille du problème, la complexité dans le pire des cas est: T(N)=1+1+log2(N)∗∗(c+1+1+1+1+)=O(log2(N))
II.2.c) En utilisant le principe de la méthode de Horner, écrire la fonction de complexité linéaire: entier(B), qui reçoit en paramètre une chaine de caractère B contenant la représentation binaire d'un entier naturel, dont le premier bit est celui du poids le plus faible, et qui calcule et retourne la valeur décimale de cet entier.

In [3]:

```
def entier(B):
    V=0
    while len(B)!=0:
        V=V*2+int(B[-1])
        B=B[:-1]
    return(V)
entier("11101")
```

Out[3]:

```
23
```

II.3 - Liste des fréquences Ecrire une fonction: **frequence(L)**, qui reçoit en paramètre une liste d'entiers L, et qui retourne une liste de tuples: chaque tuple est composé d'un élément de L et de son occurrence (répétition) dans L. Les premiers éléments des tuples de la liste des fréquences doivent être tous différents (sans doublon).

In [4]:

```
def frequence(L):
    e=set(L)
    L1=[]
    for elt in e:
        L1.append((elt,L.count(elt)))
    return(L1)
L=[12,29,46,29,31,8,12,55,29,8,12,29,31,29,8,29,8]
frequence(L)
```

Out[4]:

```
[(8, 4), (12, 3), (46, 1), (55, 1), (29, 6), (31, 2)]
```

II.4 - Ecrire la fonction: tri(F), qui reçoit en paramètre la liste F des fréquences, et qui trie les éléments de liste F dans l'ordre croissant des occurrences. Cette fonction doit être de complexité quasi-linéaire O(nlog(n)). Ne pas utiliser la fonction sorted() ou la méthode sort() de Python.

In [5]:

```
def fusion(L1,L2):
    L3=[0]*(len(L1)+len(L2))
    i=0
```

```
    j=0
    for k in range(len(L3)):
        if i>=len(L1) or (j<len(L2) and L1[i][1]>L2[j][1]):
            L3[k]=L2[j]
            j+=1
        else:
            L3[k]=L1[i]
            i+=1
    return(L3)
def tri(F):
    if len(F)<=1:
        return(F)
    else:
        m=len(F)//2
        return(fusion(tri(F[:m]),tri(F[m:])))
F=[(8, 4), (12, 3), (46, 1), (55, 1), (29, 6), (31, 2)]
tri(F)
```

Out[5]:

```
[(46, 1), (55, 1), (31, 2), (12, 3), (8, 4), (29, 6)]
```

II.5-a) Ecrire la fonction de complexité linéaire: insere(F,T), qui reçoit en paramètre la liste F des fréquences triée dans l'ordre croissant des occurrences, et un tuple T, de même nature que les éléments de F. La fonction insère le tuple T dans F de façon à ce que la liste F reste triée dans l'ordre croissant des occurrences.

In [6]:

```
def insere(F,T):
    F.append(T)
    i=len(F)-2
    while i>=0 and T[1]<F[i][1]:
        F[i+1]=F[i]
        i-=1
    F[i+1]=T
F=[(46, 1), (55, 1), (31, 2), (12, 3), (8, 4), (29, 6)]
T=(17,5)
insere(F,T)
print(F)
```

```
[(46, 1), (55, 1), (31, 2), (12, 3), (8, 4), (17, 5), (29, 6)]
```

II.5-b) Ecrire la fonction: arbre_Huffman(F), qui reçoit en paramètre la liste des fréquences F, composée des tuples formés des différents éléments de la liste à compresser, et de leurs occurrences. la fonction retourne un arbre binaire de Huffman, crée selon le principe décrit ci-dessus.

In [7]:

```
def arbre_Huffman(F):
    while len(F)>1:
        t1=F.pop(0)
        t2=F.pop(0)
        if type(t1[0])!=list:
            if type(t2[0])!=list:
                t3=([t1[1]+t2[1],[t1[0],[],[]],[t2[0],[],[]]],t1[1]+t2[1])
```

```
            else:
                t3=([t1[1]+t2[1],[t1[0],[],[]],t2[0]],t1[1]+t2[1])
        else:
            if type(t2[0])!=list:
                t3=([t1[1]+t2[1],t1[0],[t2[0],[],[]]],t1[1]+t2[1])
            else:
                t3=([t1[1]+t2[1],t1[0],t2[0]],t1[1]+t2[1])
        insere(F,t3)
    return(F[0][0])
arbre_Huffman([(46, 1), (55, 1), (31, 2), (12, 3), (8, 4), (29, 6)])
```

Out[7]:

```
[17,
 [7, [12, [], []], [8, [], []]],
 [10, [4, [31, [], []], [2, [46, [], []], [55, [], []]]], [29, [], []]]]
```

II.6- Ecrire la fonction: codes_Huffman(A,code,codesH), qui reçoit trois paramètres:

- A: est un arbre binnaire de Huffman, crée selon le principe dela question précédente.
- code: est une chaine de caractères initialisée par la chaine vide.
- codesH: est un dictionnaire initialisé par le dictionnaire vide. La fonction effectue un parcours de l'arbre, en ajoutant dans le dictionnaire codesH, les feuilles de l'arbre comme clés et les codes binaires correspondants comme valeurs des clés.

In [8]:

```
def codes_Huffman(A,code,codesH):
    if len(A[1])==0 and len(A[2])==0:
        codesH[A[0]]=code
    elif len(A[1])==0:
        codes_Huffman(A[2],code+'1',codesH)
    elif len(A[2])==0:
        codes_Huffman(A[2],code+'0',codesH)
    else:
        codes_Huffman(A[1],code+'0',codesH)
        codes_Huffman(A[2],code+'1',codesH)

code=""
codesH={}
A=[17,
 [7, [12, [], []], [8, [], []]],
 [10, [4, [31, [], []], [2, [46, [], []], [55, [], []]]], [29, [], []]]
codes_Huffman(A,code,codesH)
print(codesH)
```

```
{55: '1011', 8: '01', 12: '00', 29: '11', 46: '1010', 31: '100'}
```

II.7- Ecrire la fonction: compresse(L,codesH), qui reçoit en paramètres la liste L des entiers, et le dictionnaire codesH des codes binaires Huffman. La fonction retourne une chaine de caractères contenant la concaténation des codes binaires Huffman correspondants à chaque élément de L.

In [9]:

```
def compresse(L,codesH):
    ch=""
    for i in range(len(L)):
        ch+=codesH[L[i]]
    return(ch)
L=[12,29,46,29,31,8,12,55,29,8,12,29,31,29,8,29,8]
codesH={55: '1011', 8: '01', 12: '00', 29: '11', 46: '1010', 31: '100'}
compresse(L,codesH)
```

Out[9]:

```
'00111010111000100101111010011100111011101'
```

II.8- Ecrire la fonction: decompresse(codesH,B), qui reçoit en paramètres le dictionnaire codesH des codes de Huffman, et la chaine de caractères binaires B. La fonction construit et retourne la liste initiale.

In [10]:

```
def decompresse(codesH,B):
    L=[]
    mot=""
    for i in range(len(B)):
        mot+=B[i]
        for cle in codesH:
            if codesH[cle]==mot:
                L.append(cle)
                mot=""
                break
    return(L)
codesH={55: '1011', 8: '01', 12: '00', 29: '11', 46: '1010', 31: '100'}
B='00111010111000100101111010011100111011101'
decompresse(codesH,B)
```

Out[10]:

```
[12, 29, 46, 29, 31, 8, 12, 55, 29, 8, 12, 29, 31, 29, 8, 29, 8]
```

PARTIE 3

III.1- Ecrire une fonction:inversible(M), qui reçoit en paramètre une matrice carrée M, et qui retourne True si la matrice M est inversible, sinon, elle retourne False.

In [11]:

```
import numpy as np
def inversible(M):
    return(np.linalg.det(M)!=0)

M=np.array([[1,-1,2,-2],[3,2,1,-1],[2,-3,-2,1],[-1,-2,3,-3]],dtype=np.
float)
inversible(M)
```

Out[11]:

```
True
```

III.2- Ecrire la fonction: identite(n), qui reçoit en paramètre un entier strictement positif n, et qui crée et retourne la matrice identité d'ordre n (n lignes et n colonnes), de nombres réels.

In [12]:

```
import numpy as np
def identite(n):
    M=np.zeros((n,n),dtype=np.float)
    for i in range(n):
        M[i][i]=1
    return(M)
identite(4)
```

Out[12]:

```
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
```

III.3- Ecrire une fonction transvection(M,p,q,r), qui reçoit en paramètres une matrice M, deux entiers p et q représentant les indices de deux lignes dans la matrice M, et un nombre réel r. Dans la matrice M, la fonction remplace la ligne p par la ligne résultat de la combinaison linéaire des deux lignes p et q, suivante: Mp+r**Mq.

In [13]:

```
def transvection(M,p,q,r):
    M[p]=M[p]+r*M[q]
M=np.array([[1,-1,2,-2],[3,2,1,-1],[2,-3,-2,1],[-1,-2,3,-3]],dtype=np.
float)
transvection(M,2,0,-3)
M
```

Out[13]:

```
array([[ 1., -1.,  2., -2.],
       [ 3.,  2.,  1., -1.],
       [-1.,  0., -8.,  7.],
       [-1., -2.,  3., -3.]])
```

III.4- Ecrire la fonction: echange_lignes(M,p,q), qui reçoit en paramètres une matrice M, et deux entiers p et q représentant respectivement deux indices de deux lignes dans la matrice M, la fonction échange les lignes p et q dans la matrice M.

In [14]:

```
def echange_lignes(M,p,q):
    for i in range(len(M)):
        M[p][i],M[q][i]=M[q][i],M[p][i]
M=np.array([[1,-1,2,-2],[3,2,1,-1],[2,-3,-2,1],[-1,-2,3,-3]],dtype=np.
float)
print(M,end="\n--------------------\n")
echange_lignes(M,1,3)
print(M)
```

```
[[ 1. -1.  2. -2.]
 [ 3.  2.  1. -1.]
 [ 2. -3. -2.  1.]
 [-1. -2.  3. -3.]]
--------------------
[[ 1. -1.  2. -2.]
 [-1. -2.  3. -3.]
 [ 2. -3. -2.  1.]
```

```
 [ 3.  2.  1. -1.]]
```

III.5- Ecrire la fonction: ligne_pivot(M,c) qui reçoit en paramètres une matrice M, et un entier c représentant une colonne dans la matrice M.

La fonction retourne l'indice p tel que: |M[p,c]|=max{|M[i,c]|/0<=i<=c}

In [15]:

```
def ligne_pivot(M,c):
    imax=0
    for i in range(1,c+1):
        if abs(M[imax][c])<abs(M[i][c]):
            imax=i
    return(imax)
M=np.array([[1,-1,5,2],[10,1,-6,-4],[8,6,-3,7],[-9,5,7,-2]],dtype=np.float)
ligne_pivot(M,2)
```

Out [15]:

```
1
```

III.6- Ecrire la fonction: triangulaire_inf(M,E), qui reçoit en paramètre une matrice carrée inversible M, et une matrice identité E de même dimension que M. La fonction transforme la matrice M en matrice triangulaire inférieure. Les transformations effectuées sur la matrice M, doivent être effectuées simultanément sur la matrice E.

In [16]:

```
def triangulaire_inf(M,E):
    for i in range(len(M)-1,0,-1):
        ip=ligne_pivot(M,i)
        echange_lignes(M,i,ip)
        echange_lignes(E,i,ip)
        for k in range(i):
            r=-M[k][i]/M[i][i]
            transvection(M,k,i,r)
            transvection(E,k,i,r)
#test
M=np.array([[1,-1,2,-2],[3,2,1,-1],[2,-3,-2,1],[-1,-2,3,-3]],dtype=np.
float)
E=identite(len(M))
triangulaire_inf(M,E)
print(M,E,sep="\n\n")
```

```
[[ 1.25        0.          0.          0.        ]
 [ 3.33333333  2.66666667  0.          0.        ]
 [ 1.66666667 -3.66666667 -1.          0.        ]
 [-1.         -2.          3.         -3.        ]]

[[ 1.         -0.125       0.         -0.625     ]
 [ 0.          1.          0.         -0.33333333]
 [ 0.          0.          1.          0.33333333]
 [ 0.          0.          0.          1.        ]]
```

III.7- Les deux matrices M et Esont les résultats de la fonction triangulaire_inf(M,E). La matrice M est devenue triangulaire inférieure. Ecrire la fonction : diagonale(M,E), qui transforme la matrice

M en matrice diagonale. Les transformations effectuées sur la matrice M, doivent être aussi effectuées simultanément sur la matrice E.

In [17]:

```
def diagonale(M,E):
    for i in range(len(M)-1):
        for k in range(i+1,len(M)):
            c=-M[k][i]/M[i][i]
            transvection(M,k,i,c)
            transvection(E,k,i,c)
M=np.array([[1,-1,2,-2],[3,2,1,-1],[2,-3,-2,1],[-1,-2,3,-3]],dtype=np.
float)
E=identite(len(M))
triangulaire_inf(M,E)
diagonale(M,E)
print(M,E,sep="\n\n")
```

```
[[  1.25000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [ -4.44089210e-16   2.66666667e+00   0.00000000e+00   0.00000000e+00]
 [ -8.32667268e-16   0.00000000e+00  -1.00000000e+00   0.00000000e+00]
 [ -2.83106871e-15   0.00000000e+00   0.00000000e+00  -3.00000000e+00]]

[[  1.          -0.125        0.          -0.625     ]
 [ -2.66666667   1.33333333   0.           1.33333333]
 [ -5.           2.           1.           3.        ]
 [-16.2          6.9          3.          10.5       ]]
```

III.8 - Ecrire la fonction: inverse(M), qui reçoit en paramètre une matrice carrée inversible M, et qui calcule et renvoie la matrice inverse de M. (Ne pas utiliser la méthode prédéfinie inv() de Python)

In [18]:

```
def inverse(M):
    if not inversible(M):
        return(None)
    else:
        E=identite(len(M))
        triangulaire_inf(M,E)
        diagonale(M,E)
        for i in range(len(M)):
            c=1/M[i][i]
            M[i]*=c
            E[i]*=c
        return(E)
#test
M=np.array([[1,-1,2,-2],[3,2,1,-1],[2,-3,-2,1],[-1,-2,3,-3]],dtype=np.
float)
inverse(M)
```

Out [18]:

```
array([[ 0.8, -0.1,  0. , -0.5],
       [-1. ,  0.5,  0. ,  0.5],
```

```
       [ 5. , -2. , -1. , -3. ],
       [ 5.4, -2.3, -1. , -3.5]])
```

III.9- Ecrire la fonction: matrice_fichier(ch), qui reçoit en paramètre une chaine de caractère ch, qui contient le chemin absolu du fichier texte, contenant une matrice carrée:

In [22]:

```
def matrice_fichier(ch):
    try:
        f=open(ch,'r')
        ch=f.read()
    except FileNotFoundError:
        print("Fichier texte inexistant")
    except IOError:
        print("Erreur de lecture et écriture")
    else:
        f.close()
        L=ch.split('\n')
        L=[L[i].split(" ") for i in range(len(L))]
        M=np.array([[float(L[i][j]) for j in range(len(L[i]))] for i in
range(len(L))], dtype=np.float)
        I=inverse(M)
        if I==None:
            print("Matrice non inversible")
        else:
            #print(np.linalg.inv(M))
            return(I) #différence suite aux arrondis
#test
ch="C:\disque dur secours\\valise professeur\MEKNES\documents
pédagogiques\spe\PREPARATION AU CONCOURS\corriges cnc 2010
2015\matrice.txt".replace('\\','/')
matrice_fichier(ch)
```

```
[[  1.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
    0.00000000e+00   3.62845927e-18]
 [ -8.25979629e-34   1.00000000e+00   0.00000000e+00   0.00000000e+00
    0.00000000e+00   1.75472436e-17]
 [ -2.75891699e-34   0.00000000e+00   1.00000000e+00   0.00000000e+00
    0.00000000e+00   5.86108746e-18]
 [  3.39354545e-16   0.00000000e+00   0.00000000e+00   1.00000000e+00
    0.00000000e+00   1.32613056e-17]
 [  3.72195307e-17   0.00000000e+00   0.00000000e+00   0.00000000e+00
    1.00000000e+00   2.62834020e-17]
 [ -4.70717595e-17   0.00000000e+00   0.00000000e+00   0.00000000e+00
    0.00000000e+00   1.00000000e+00]]
C:\Users\Oustouh\Anaconda3\lib\site-packages\ipykernel\__main__.py:15:
FutureWarning: comparison to `None` will result in an elementwise object
comparison in the future.
```

Out[22]:

```
array([[ 0.25308584,  0.03268226,  0.12008777, -0.16350549,  0.18858736,
         0.23419161],
```

```
       [ 0.30400116,  0.15805152,  0.35635997, -0.0865681 ,  0.12640951,
         0.2455182 ],
       [ 0.14175007,  0.05279198, -0.0324928 , -0.0900088 ,  0.2748197 ,
         0.15214007],
       [ 0.12156406,  0.11944722,  0.04927087, -0.06598551,  0.04835943,
         0.16747871],
       [ 0.18311936,  0.23673984,  0.22223132, -0.24274864,  0.16254313,
         0.09296941],
       [ 0.12556866, -0.07311094,  0.17724581, -0.02414937,  0.15272487,
         0.34355919]])
```

Signée : L.OUSTOUH