teaching sciences for innovation

Proposition de corrigé

Concours : Concours Commun Polytechniques

Année : 2017

Filière : TSI

Épreuve : Informatique

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles de l'Ingénieur et d'Informatique, membres de l'UPSTI (Union des Professeurs de Sciences et Techniques Industrielles), et publiée sur le site de l'association :

https://www.upsti.fr/espace-etudiants/annales-de-concours

A l'attention des étudiants

Ce document vous apportera des éléments de corrections pour le sujet traité, mais n'est ni un corrigé officiel du concours, ni un corrigé détaillé ou exhaustif de l'épreuve en question.

L'UPSTI ne répondra pas directement aux questions que peuvent soulever ces corrigés : nous vous invitons à vous rapprocher de vos enseignants si vous souhaitez des compléments d'information, et à vous adresser à eux pour nous faire remonter vos éventuelles remarques.

Licence et Copyright

Informez-vous !

Retrouvez plus d'information sur les Sciences de l'Ingénieur, l'orientation, les Grandes Ecoles ainsi que sur les Olympiades de Sciences de l'Ingénieur et sur les Sciences de l'Ingénieur au Féminin sur notre site : www.upsti.fr

L'équipe UPSTI

Programmation graphique du jeu de rugby

Q1.

```
import os
os.chdir('C:/CCP')

"""affichage stade"""
#bibliotheque image
import scipy.misc as scm

image=scm.imread("stade.bmp") #stocker l'image dans une variable
imshow(image)

print (image) #donne le nombre de pixels de l'image
```

Q2.

```
dim_long=image.shape[1] #taille de l'image
dim_larg=image.shape[0] #taille de l'image
print ("taille de l'image : %d x %d" % (dim_long,dim_larg))
```

Q3.

```
 def coul(image):
     coul_ter=image[int(dim_long/2)-2,int(dim_long/2)]
     # on assure le milieu avec partie entiere et on decale de 2 pixels
     return (coul_ter)

#ou aussi
 def coul(image):
     coul_ter=image[int(dim_long/2)+1,int(dim_long/2)]
     # on assure le milieu avec partie entiere et on decale de 1 pixels
     return (coul_ter)
```

Q4.

Coul_ter est une liste de 3 nombres entiers : Rouge, Vert, Bleu
Un pixel est donc codé sur 24 bit (3 octets pour chaque entier de couleur).

Q5.

```
def maillot(image):
    vert=coul(image)
    blanc=[255,255,255]
    return [vert,blanc]
```

Q6.

```
def filtrer1(filtreA,matB):
    nA=filtreA.shape[0] #matrice carree, donc shape[0]=shape[1]
    nb_ligneB=matB.shape[0]
    nb_colonneB=matB.shape[1]
    C=matB.copy()
    bordure =nA//2
    for i in range(bordure,nb_ligneB-bordure):
        for j in range(bordure,nb_colonneB-bordure):
            Bij=matB[i-bordure:i+bordure+1,j-bordure:j+bordure+1]
            C[i,j]=np.sum(Bij.dot(filtreA)) #on considere que tout numpy est importe
    return C
```

Q7.

```
def filtrer(filtreA,matB):
    taille=filtreA.shape[0] # matrice carree, donc shape[0]=shape[1]
    bordure=taille//2
    nb_lig,nb_col,nb_coul=matB.shape
    for i in range(bordure,nb_lig-bordure):
        for j in range(bordure,nb_col-bordure):
            for k in range(3): #ou nb_coul
                Bij=matB[i-bordure:i+bordure+1,j-bordure:j+bordure+1,k]
                matC[i,j,k]=np.sum(Bij.dot(filtreA))
    return matC
```

Q8.

```
def matriceFlouGaussien ( taille , sigma ):
    mat = zeros ([ taille , taille ])
    taille =taille // 2
    for x in range (- taille , taille +1) :
        for y in range (- taille , taille +1) :
            mat [x+ taille ,y+ taille ] = exp (-(x **2+ y **2) /(2*( sigma **2) )) #fonction exp de numpy
    return mat /sum ( mat ) #on utilise sum de numpy
```

Q9.

```
def FloutageGaussien ( tabPix , taille , sigma ):
% """
% tabPix : tableau de pixel
% taille : taille de la matrice ( impaire )
% sigma : ecart type (deviation standard )
% retourne un tableau de pixel en niveau de gris floute
% """
    filtre = matriceFlouGaussien ( taille , sigma )
    result=filtrer(filtre,tabPix)
    return result
```

Q10.

```
x=[resultat[i,0] for i in range(len(resultat))]
y_plaR=[resultat[i,3] for i in range(len(resultat))]
```

Q11.

```
# listes pour l'histogramme
barre=[]
abscisse=[]
for i in range (len(x)):
    for j in arange(0,y_plaR[i]+0.1,0.1): # arange importe de numpy
        barre.append(j)
        abscisse.append(x[i])
```

Q12.

```
def minMaxMoy(valeurs):
    """ retourne les min, max, et moyenne d'une liste"""
    min=valeurs[0]
    max=valeurs[0]
    somme=0
    for k in valeurs:
        if k>max:
            max=k
        if k<min:
            min=k
        somme+=k
        moyenne=somme/len(valeurs)

    return [min,max,moyenne]
```

Traitement du transfert des joueurs

Q13.
requete1="SELECT Nom FROM JOUEURS WHERE age >23 AND VMA >13"

Q14.
requete2="SELECT C.Nom FROM CLUBS as C JOIN JOUEURS as J ON J.Ident_Club=C.Id_Club WHERE J.Salaire >30000"

requete3="SELECT Count(*) FROM JOUEURS as J JOIN CLUBS as C ON J.Ident_Club=C.Id_Club WHERE C.Nom='Stade Toulousain' AND J.Poste='Talonneur"'

Q15.
requete4=”SELECT sum(J.Salaire)/C.Budget*100 FROM JOUEURS as J JOIN CLUBS as C ON J.Ident_Club=C.Id_Club GROUP BY C.Budget”’

Évaluation des performances de l’équipe

Q16.
monequipe est une liste de liste qui contient des string

Q17.

```
def echange(l,i,j):
    # echange 2 valeurs d'une liste
    l[i],l[j] = l[j], l[i]


def tri_1(liste,critere):
    for i in range(len(liste)-1):
        mini=i
        for j in range(i+1,len(liste)):
            if liste[j][critere]<liste[mini][critere]:
                mini=j
        echange(liste,i,mini)
    return liste

tri_1(monequipe,5)
#
print(monequipe)
```

Q18.
La complexité dans le pire des cas de tri1 est $O(n^2)$, car il y a 2 boucles imbriquées sur la taille de la liste. Le nombre d’échange est donc de l’ordre de 1+2+3+ ... (n-1) c’est à dire un nombre de coût : $\Sigma_{k=0}^{n} k = \frac{(n-1)(n+1)}{2} = O(n^2)$ Une complexité quadratique qui peut devenir très pénalisante si la taille devient grande.

Q19.

Donnée :

```
def segmente(T,val,i,j):
    g=i+1
    d=j
    p=T[i][val]
    while g<=d :
        while d>=0 and T[d][val]>p:
            d=d-1
        while g<=j and T[g][val]<=p:
            g=g+1
        if g<d:
            echange(T,g,d)
            d=d-1
            g=g+1
    k=d
    echange(T,i,d)
    return k

def Tri_2(L,val, i, j): # il s agit d une fonction recursive de tri rapide
    if i<j: # test d arret : on tri chaque sous liste jusqu a ce que i=j
        k=segmente(L,val,i,j) # on recupere l indice qui permet de mettre d un cote les
        elements plus petit que L[k] et de l autre les plus grands :
        \textbf{il s agit du pivot}
        Tri_2(L,val,i,k-1) # on reitere ce tri (appel recursif) sur la liste de gauche :
         on separe les plus petits a gauche, les plus grand a droite du pivot
        Tri_2(L,val,k+1,j) # meme chose sur la liste de droite.
    return L # la liste finale L a ete modifiee avec les appels successifs, elle est donc triee


def Tri_2(L,val, i, j):
    if i<j:
        k=segmente(L,val,i,j)
        return 2 + Tri_2(L,val,i,k-1) + Tri_2(L,val,k+1,j)
    else :
        return 0
```

Q20.

```
print(Tri_2(monequipe,3,0,len(monequipe)-1))
```

Q21.

74.25 en binaire sur 8 bits =1001010.01

Q22.

le bit de signe 0

l'écriture est $1.00101001 * 2^6$ donc l'exposant est 6+127=133 soit sur 8 bit 10000101

74,25 au format IEEE 754 se défini comme :

0 1000 0101 001 0100 1000 000 000 0000

Q23.
3000 joueurs avec 3 informations (poids, taille, vitesse) qui font chacune 32 bits. 1 octet = 8 bits et on divise par 1000 pour le Kilo octet. Espace de stockage : 3000*3*32/8/1000=36 Ko, ce qui est très faible .
En format double on aurait 72 Ko ce qui reste très faible !
Remarque : C'est une des raisons qui a amené à ne pas implémenter les flottants en simple précision en Python (remarque qui dépasse le cadre du programme).
Avantage : on gagne en place mémoire et un peu en temps d'exécution
Désavantage : on perd en précision. On traite beaucoup moins de nombres flottants.

Q24.

```
#Definition d'une fonction abscisse des temps
def liste_temps(tmax,pas):
        '''Renvoie une liste des abscisses en temps
        a partir du pas et de la borne superieure des temps'''
        t=0
        temps=[0]
        while t<=(tmax-pas):
                t=t+pas
                temps=temps+[t]
        return (temps)
```

Q25.
$v(t) = K_c \cdot U_0(1 - e^{-\frac{t}{\tau}})$

```

def s(k,tau,u,temps):
        s=[0]*(len(temps))
        for i in range(len(temps)):
                s[i]=k*u*(1-math.exp(-temps[i]/tau))
        return s
```

Q26.

```
def ordre1_euler(k,tau,u,temps):
        s=0
        sortie=[0]
        for i in range(1,len(temps)):
                f=(k*u-s)/tau
                s=s+f*(temps[i]-temps[i-1])
                sortie=sortie + [s]
        return sortie
```

Q27.

```
for i in (0.2,0.4,0.6):
        x=liste_temps(tmax,i)
        y=ordre1_euler(K_c,tau_c,U_0,x)
```

Q28.

```
import matplotlib.pyplot as plt
for i in (0.2,0.4,0.6):
        x=liste_temps(tmax,i)
        y=ordre1_euler(K_c,tau_c,U_0,x)
        plt.plot(x,y)

#Affichage superpose des courbes
plt.show()
```

Q29.

La complexité dans le pire des cas est $O(n^2)$ car il y a 2 boucles imbriquées.