teaching sciences for innovation

Proposition de corrigé

Concours : Concours Commun Mines-Ponts

Année : 2018

Filière : MP - PC - PSI

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

Concours commun Mines Ponts 2018
Correction sujet Informatique

Partie I. Stockage interne des données

Q1 - nombre d'octets correspondant à 20 minutes d'enregistrement

8 caractères par lignes sur 8 bits = 8 octets par mesures
20 minutes à 2 Hz = 20*60*2 = 2400 mesures
soit 19200 octets en 20 minutes

Q2 nombre d'octets correspondant à la campagne de mesures

En 15 jours = 15 x 24 x 2 x 19 200 = 13 824 000 octets
soit 13.8 Mo (13.13 Mio)
La carte mémoire est largement suffisante.

Q3 - Gain relatif d'espace mémoire

Un chiffre de moins donne 1 octet de moins par mesure soit 12.5% de moins (soit 1.8Mo de moins)

Q4 - Lecture du fichier

```
def liste_niveaux():
    fichier = open('donnees.txt','r')
    ligne = fichier.readline()
    liste_niveaux = []
    while ligne:
        ligne = fichier.readline()
        if ligne:
            liste_niveaux.append(float(ligne))
    fichier.close()
    return liste_niveaux
```

Concours commun Mines Ponts 2018
Correction sujet Informatique

Partie II. Analyse “vague par vague”

Q5 - Analyse de la courbe

H1 = 6 - (-3) = 9
H2 = 6.9 - (-2) = 8.2
H3 = 5 - (-1.2) = 6.2

T1 = 15.5 - 3.75 = 11.75 s
T2 = 28.25- 15.5 = 12.75 s

Remarque : La précision des temps sur le graphique est largement supérieur à la fréquence d'échantillonage de 2Hz indiquée.

Q6 - Calcul de la moyenne

```
def moyenne(liste_niveaux):
    somme = 0
    for niveau in liste_niveaux:
        somme += niveau
    return somme / len(liste_niveaux)
```

Q7 - Calcul de la moyenne précise

```
def integrale_precise(liste_niveaux):
    integrale = 0
    for i in range(20 * 60* 2 - 1):
        trapeze = (liste_niveaux[i] + liste_niveaux[i+1] ) / 2 * 0.5
        integrale += trapeze
    return integrale

def moyenne_precise(liste_niveaux):
    return integrale_precise(liste_niveaux) / (20 * 60)
```

Q8 - Détermination de l'indice du premier PND

```
def ind_premier_pzd(liste_niveaux):
    i = 0
    moy = moyenne(liste_niveaux)
    while not(liste_niveaux[i] > moy and \
              liste_niveaux[i+1] < moy) and \
              i < len(liste_niveaux):
        i += 1
    if liste_niveaux[i] > moy and liste_niveaux[i+1]:
        return i
    else:
        return -1
```

Q9 - Détermination de l'indice du dernier PND

```
def ind_dernier_pzd(liste_niveaux):
    i = len(liste_niveaux) - 2
    moy = moyenne(liste_niveaux)
    while not(liste_niveaux[i] > moy and \
            liste_niveaux[i+1] < moy) and i >0:
        i -= 1
    if liste_niveaux[i] > moy and liste_niveaux[i+1]:
        return i
    else:
        return -2
```

Q10 - Compléter l'algorithme 1

```
def construction_successeurs(liste_niveaux):
    n = len(liste_niveaux)
    successeurs = []
    m = moyenne_precise(liste_niveaux)
    for i in range(n-1):
        if i+1 > ind_premier_pzd(liste_niveaux):
            successeurs.append(i+1)
    return successeurs
```

Remarque : Cet algorithme est inefficace car cela augmente la complexité. La démarche de l'interrogateur de demander de compléter un code déjà existant est très réductrice pour ce cas.

Q11 - Décomposition d'une liste de niveaux en liste de vagues

```
def decompose_vagues(liste_niveaux):
    m = moyenne(liste_niveaux)
    i_debut = ind_premier_pzd(liste_niveaux)
    i_fin = ind_dernier_pzd(liste_niveaux)
    vagues = []
    i = i_debut + 1
    while i < i_fin:
        i_vague = ind_premier_pzd(liste_niveaux[i:])
        vagues.append(liste_niveaux[i: i + i_vague + 1])
        i += i_vague + 1
    return vagues
```

Q12 - Caractérisation des vagues

```
def proprietes(liste_niveaux):
    vagues = decompose_vagues(liste_niveaux)
    liste = []
    for i in range(len(vagues)-1):
        Hi=max(vagues[i])-min(vagues[i+1])
        Ti=len(vagues[i]) * 0.5 # fréquence de 2 Hz
        liste.append([Hi, Ti])
    return liste
```

Partie III. Contrôle des données

Q13 - Calcul de H_{max}

```
def h_max(liste):
    liste = proprietes(listes)
    maxi = liste[0][0]
    for Hi, Ti in liste:
        if Hi > maxi:
            maxi = Hi
    return maxi
```

Question ambigüe sur l'argument liste_niveaux *ou* proprietes(liste_niveaux)

Q14 - Compléter la fonction Tri rapide

L'appel de la fonction se fera par : triRapide(proprietes , 0 , len(proprietes)-1)
donc g =0 et d =le dernier indice de la liste à trier

```
#Algorithme 2
def triRapide(liste, g , d ):
  pivot = liste[g][0]
  i = g
   j = d
   while True :
      while i <= d and liste[i][0] < pivot:
          i = i + 1
      while j >= g and liste[j][0] > pivot:
          j = j-1
      if i > j:
          break
      if i < j:
          liste[i], liste[j] = liste[j], liste[i]
      i = i + 1
      j = j - 1
   if g < j:
      triRapide(liste, g, j)
   if i < d:
      triRapide(liste, i, d)
```

Q15 - Appel de la fonction Tri_insertion quand le nombre de données est inférieur à 15

```
#Algorithme 2
def triRapide(liste, g , d ):
    if d - g < 15:
        triInsertion(liste, g, d)
    else:
        pivot = liste[g][0]
        i = g
        j = d
        while True :
            while i <= d and liste[i][0] < pivot:
                i = i + 1
            while j >= g and liste[j][0] > pivot:
                j = j-1
            if i > j:
                break
            if i < j:
                liste[i], liste[j] = liste[j], liste[i]
            i = i + 1
            j = j - 1
        if g < j:
            triRapide(liste, g, j)
        if i < d:
            triRapide(liste, i, d)
```

Q16 - Compléter la fonction Tri insertion

```
#Algorithme 3
def triInsertion(liste, g, d):
    for i in range(g+1, d+1):
        j = i - 1
        tmp = liste[i]
        while j >= 0 and liste[j][0] > tmp[0]:
            liste[j+1] = liste[j]
            j -= 1
        liste[j+1] = tmp
```

Q17 -Modification simple de la fonction pour diminuer le temps d'exécution

```
#Algorithme 4
def skewness(liste_hauteurs):
    n=len(liste_hauteurs)
    et3 = (ecartType(liste_hauteurs))**3
    S = 0
    m3 = moyenne(liste_hauteurs)**3
    for i in range(n):
        S += (liste_hauteurs[i] - m3)
    S  = n / (n - 1) / (n - 2) * S / et3
    return S
```

Q18 - Différence de type de la complexité

Dans les deux cas la complexité est la même en O(n)

Q19 - Requêtes SQL

- Quels sont le numéro d'identification et le nom de site des bouées localisées en Méditerranée ?

```
SELECT idBouee, nomSite FROM Bouee
WHERE localisation = "Mediterranee";
```

- Quel est le numéro d'identification des bouées où il n'y a pas eu de tempêtes ?

```
SELECT idBouee FROM Bouee
WHERE idBouee NOT IN (SELECT idBouee FROM Tempete);
```

- Pour chaque site, quelle est la hauteur maximale enregistrée lors d'une tempête ?

```
SELECT nomSite, MAX(Hmax) FROM Bouee
JOIN Tempete ON Tempete.idBouee = Bouee.idBouee
GROUP BY nomSite;
```

Partie V. Analyse “spectrale”

Q20 - Complexité en temps de cet algorithme

Calculer Pk est de complexité O(N/2-1) Ainsi que pour Ik

Calculer Xk est donc de complexite O(N-2) donc pour tous les k $O(N^2-2N)$ soit $O(N^2)$

Q21 - Fonction récursive pour calculer la transformée de Fourrier discrète rapide

```
import numpy as np

def transformee(liste_x):
    if len(liste_x) == 1:
        return liste_x
    else:
        n = len(liste_x)
        pair = liste_x[: : 2]
        impair = liste_x[1: : 2]
        w = np.exp(-np.pi * 2j / n)
        P = transformee(pair)
        I = transformee(impair)
        X = [0] * n
        for k in range(n // 2):
            X[k] = P[k] + (w**k) * I[k]
            X[k+n//2] = P[k] - (w**k) * I[k]
        return X
```