Corrigé

Éléments de correction du sujet Mines Ponts 2015

Julien ressources UPSTI

Question 1 10 bits permettent d'obtenir $2^{10} = 1024$ valeurs. Étant donné qu'elles sont signées, on pourra représenter des entiers de - 512 à 511.

Question 2 La résolution est de :$10/1023 = 0,009\,775$ V/mesure.

```
def lect_mesures() :
    fin =0
    data =[]
    while (fin ==0):
        # obtention du type de donnee et stockage si caractere correct
        carac = com. read (1)
        if ( carac == 'U' or carac == 'I' or carac == 'P' ):
            fin = 1
            data . append(carac)
            # obtention du nombre de donnees a lire
            nb_data = int ( com . read (3))
           # reception de toutes les donnees par paquet de 4 et fabrication de la liste des mesures
           mesures = []
           for i in range ( nb_data ):
               data _read = com.read (4)
               mesures.append(int(carac))
           data.append ( mesures)
           # reception du checksum
          data.append(int(com.read(4)))
    return data



def check(mesure,CheckSum):
    somme =0
    for i in range (len(mesure)):
        somme += abs ( mesure [ i ])
    if somme % 10000 == CheckSum :
        return True
    else :
        return False
```

Question 5 Version n'utilisant pas numpy :

```
def affichage(mesure):
    t=[0]
    for i in range(len(mesure)-1):
        t.append(t[-1]+2)     #le temps est en milliseconde
```

```
    for i in range(len(mesure)) :
        mesure[i] = mesure[i]*4e-3
    plot(t,mesure)
```

Version utilisant numpy :

```
def affichage (mesure) :
    t=arange(0,len(mesure)*2,2)
    mesure =array(mesure)*4e-3
    plot(t,mesure)
```

Question 6 Version de base :

```
def moyenne(mesure)
    dt=2
    #calcul du temps final
    tfinal = (len(mesure)-1)*dt
    #définition du tableau des temps
    temps=arange(0,len(mesure)*dt,dt)
    #calcul de l'intégrale
    integ = 0
    for i in range(0,len(temps)-1) :
        integ += (mesure[i+1]+mesure[i])*(temps[i+1]-temps[i])/2
    return 1.0/tfinal*integ
```

Version optimisée permettant de limiter l'accès à mesure[i] :

```
def moyenne ( mesure):
    dt  =  2
    tfinal = (len(mesure) -1)* dt
    integrale = (mesure[0]+ mesures [ -1])/2*2 # prise en compte des extrémités
    for i in range (1,,len(mesure)-2) :
        integrale += measures[i]*dt
    Imoy=integrale/tfinal
    return Imoy
```

Question 7

```
def calcul_ecart_type (mesure)
    Imoy = moyenne (mesure)
    tmp  =  [ ]  # calcul de  l'ecart I  -  Imoy
    for  i  in range  (len(mesure)):
        tmp.append((mesures[i]  -  Imoy )**2)
    Iec = moyenne  ( tmp ) / tfinal
    return Iec ** (0.5)
```

Question 8

```
SELECT nSerie FROM testfin WHERE ( Imoy > Imin AND Imoy < Imax)
```

Question 9 La sous requête permettant d'obtenir la valeur moyenne de l'écart type est :

```
SELECT AVG(Iec) FROM testfin
```

La requête complète est :

```
SELECT nSerie, Iec, fichierMes FROM testfin WHERE Iec > (SELECT AVG(Iec) FROM testfin)
```

```
SELECT nSerie,fichierMes FROM testfin WHERE nSerie NOT IN (SELECT nSerie FROM production)
```

Question 11 test(x) permet de tester que l'élément x est bien un tuple, de taille 2, dont le premier élément est une chaine de caractère et le second un entier. Renvoie un booléen True si c'est vrai False sinon.

get1(x) permet d'extraire le premier élément de x. Le type de la valeur retournée sera du même type que celui de la valeur du premier élément de x (ici une chaîne de caractère).

get2(x) permet d'extraire le second élément de x. Le type de la valeur retournée sera du même type que celui de la valeur du deuxième élément de x (ici une chaîne d'entiers).

Question 12 On aura

```
node1= [('F',1),('E',1), ['F','E'],2] et Node2= [('D',1),('B',1), ['D','B'],2]
```

Question 13 On aura

```
node3 = [ [('F',1),('E',1), ['F','E'],2], [('D',1),('B',1), ['D','B'],2], ['F','E','D','B'], 4]
```

Question 14 Une documentation sur les dictionnaires permettrait de mieux comprendre le fonctionnement de cette fonction. Le résultat sera

```
f = {'C':1,'B':3,'A':2}
```

Question 15 Il s'agit d'une définition récursive car la fonction fait appel à elle-même. La définition est correcte car l'appel récursif augmente l'argument pos d'une unité ce qui fait que nécessairement l'un des deux premiers critères finira par être vérifié.

Question 16 A l'initialisation, pos = 0 ainsi la sous liste est vide. Elle vérifie donc la propriété que ses éléments sont de poids inférieur à celui de item.

Si on suppose la propriété vraie à la ième itération alors à l'itération suivante, il y a 3 possibilités :

- pos vaut la taille de la liste, dans ce cas item à un poids supérieur à celui de tous les éléments de la liste et on ajoute l'élément à la fin de la liste et la propriété reste vraie ;
- le poids de l'item est inférieur strictement à celui de la position pos : dans ce cas, on insère l'item à cette position, décalant les éléments suivant vers la droite de la liste, la propriété reste vraie ;
- sinon, l'item à un poids plus grand ou égal à celui de la position pos et on va essayer de le placer à la position pos+1 et la propriété restera vraie.

```
##5 : travail sur la liste si celle-ci a plus de deux éléments
##6 : construction d'un noeud de huffman avec le deux feuilles/noeuds de poids le plus faible
##7 : effacement des deux premières feuilles/noeuds traités
##8 : insertion du noeud créé dans la liste àla bonne position pour avoir une liste de
##    noeuds triés par poids croissant
```

Question 18 Dans le meilleur des cas l'élément à insérer à une fréquence d'apparition inférieure à tous les autres éléments, dans ce cas, l'élément est inséré au premier appel de la fonction en première position, la complexité est en O(1).

Dans le pire des cas, l'élément à insérer à une fréquence d'apparition supérieure à tous les autres éléments, dans ce cas, l'élément doit être inséré en dernière position et pour cela il faut parcourir tous les éléments de la liste une fois. Ainsi la complexité est linéaire (en O(n)).

Question 19 Pour une liste d'origine de taille n, on passera n-2 fois dans la boucle while.

Dans le meilleur des cas, la construction de l'arbre passe par la création de nœuds qui ont toujours une fréquence d'apparition inférieure à celle du nœud suivant (exemple : 1,1,2,4,8,16...). Ainsi l'insertion d'un élément sera en $\mathcal{O}(1)$, la rapidité de la boucle ne dépendra que du nombre d'éléments de la liste initiale, la complexité sera linéaire ($\mathcal{O}(n)$) (n-2 comparaisons pour une liste de n tuples dans la fonction insert_item et autant pour le test de la boucle while).

Dans le pire des cas, l'élément à insérer sera toujours plus grand que le dernier élément de la liste, ainsi la boucle while fera appel à une fonction de complexité linéaire. Ainsi la complexité sera quadratique ($\mathcal{O}(n^2)$). En détail, à la première étape, la liste fait une taille n-2 après suppression des 2 feuilles/noeuds, il faut n-2 comparaisons pour insérer le nouvel élément. A l'étape suivante, la liste fera une taille de n-3 après la suppression des 2 feuilles, il faudra n-3 comparaisons pour insérer le nouvel élément... jusqu'à obtenir une liste de taille 1 après suppression des 2 feuilles pour laquelle il faudra 1 comparaison pour l'insertion... Il y a donc comparaisons. En ajoutant les (n-2) comparaisons du test de la boucle while, on obtient bien une complexité quadratique.

```
node3 = [ [('C',1),('Z',1), ['C','Z'],2], ('B',3), ['C','Z','B'], 5]
```

Question 21

```
function calc_eq ( k)
    temps = [ 0 : 0 . 1 : 1 0 ]

    s = zeros ( size ( temps , 1))
    e = sin ( temps )
    // utilisation d un schéma numérique
    for i  =  [1 , size ( temps , 1) -1]
        s [ i +1] = s [ i ] * (1 - k / 10 *0 .1) + k * e [ i ]
    endfor
```

Sujet

```
    return s
end function
```

Question 23 Le critère choisi consiste à vérifier que le maximum d'écart entre la solution approchée et la mesure est inférieur à un epsilon à définir.

```
function verif ()
    s_app1 = calc_eq (0.5)
    s_app2 = calc_eq (1.1)
    s_app3 = calc_eq (2)
    if  max (( s - s_app1 )) < eps then return True
    else  if  max (s - s_app2 ) < eps then return True
    else if max (s - s_app3 ) < eps then return True
    else return False
end function
```