MINES-PONTS - 2018 - INFORMATIQUE COMMUNE

Rédigé par Jérémy Larochette (Lycée Carnot, Dijon).

On utilise Python 3.

Partie I. Stockage interne des données

❏ **Q1** – On compte 2 échantillons par seconde pendant 20 minutes soit 1200 secondes. On a donc 2401 échantillons (en comptant celui pour $t = 0$.) Chaque échantillon pèse 8 octets (64 bits).
Un enregistrement de 20 minutes pèse donc 19,208 ko.

❏ **Q2** – 15 jours d'enregistrements à raison de 48 enregistrements par jour donnent $15 \cdot 48 = 720$ enregistrements. Donc de l'ordre de $720 \cdot 20$ ko ≈ 14 Mo. Une carte de 1 Go est amplement suffisante.

❏ **Q3** – 1 chiffre en moins représente une économie relative de $\frac{1}{8} = 12,5\ \%$.

❏ **Q4** –

```
with open("donnees.txt") as fichier:
    fichier.readline() # on saute la 1ère ligne.
    liste_niveaux = [float(ligne) for ligne in fichier]
```

Partie II. Analyse "vague par vague"

❏ **Q5** – On mesure $H_1 \approx 9$ m, $H_2 \approx 8,8$ m, $H_3 \approx 6,2$ m, $T_1 \approx 12$ s, $T_2 \approx 12,5$ s.

❏ **Q6** –

```
def moyenne(liste_niveaux):
    return sum(liste_niveaux) / len(liste_niveaux)
```

❏ **Q7** – Avec la formule $I \approx h\sum_{k=0}^{n-1} \frac{\eta(t_k)+\eta(t_{k+1})}{2}$, avec $h = 0,5$ s vu $f = 5$ Hz.

```
def integrale_precise(liste_niveaux):
    n = len(liste_niveaux)
    return sum(liste_niveaux[k] + liste_niveaux[k + 1] for k in range(n - 1)) / 4
```

ou avec $I \approx h\left(\sum_{k=0}^{n} \eta(t_k) - \frac{\eta(t_0)+\eta(t_n)}{2}\right)$.

```
def integrale_precise(liste_niveaux):
    return (sum(liste_niveaux) - (liste_niveaux[0] + liste_niveaux[-1]) / 2) / 2
```

Puis, comme la durée de l'échantillon est de 1200 secondes,

```
def moyenne_precise(liste_niveaux):
    return integrale_precise(liste_niveaux) / 1200
```


❏ **Q8** –

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

❏ **Q9** –

```
def ind_dernier_pzd(liste_niveaux):
    m = moyenne_precise(liste_niveaux)
    n = len(liste_niveaux)
    for i in range(n - 2, -1, -1):
        if liste_niveaux[i] >= m > liste_niveaux[i + 1]:
            return i
    return -2
```

Difficile sans recalculer la moyenne... Donc pas de $O(1)$ contrairement à ce que stipule l'énoncé.

❏ **Q10** –

```
if liste_niveaux[i] >= m > liste_niveaux[i + 1]:
    successeurs.append(i + 1)
```

❏ **Q11** –

```
def decompose_vagues(liste_niveaux):
    succ = construction_successeurs(liste_niveaux)
    N = len(succ)
    return [liste_niveaux[succ[i]:succ[i + 1]] for i in range(N - 1)]
```

❏ **Q12** –

```
def decompose_vagues(liste_niveaux):
    liste_vagues = decompose_vagues(liste_niveaux)

    # 1ère vague à traiter à part
    prop = [[max(liste_niveaux[:succ[0]]) - min(liste_vagues[0]),
             0.5 * len(liste_vagues[0])]]

    for i in range(1, len(liste_vagues)): # T indéterminé pour la dernière
        H = max(liste_vagues[i - 1]) - min(liste_vagues[i])
        T = 0.5 * len(liste_vagues[i]) # /!\ 2 Hz
        prop.append([H, T])

    return prop
```

Partie III. Contrôle des données

❏ **Q13** –

```
def Hmax(liste_niveaux):
    return max(p[0] for p in proprietes(liste_niveaux))
```

ou bien

```
def Hmax(liste_niveaux):
    prop = proprietes(liste_niveaux)
    h = -1
    for i in range(len(prop)):
        if prop[i][0] > h:
            h = prop[i][0]
    return h
```

❑ **Q14** – Au premier appel, `g = 0` et `d = len(liste) - 1`.
Ligne 2 : `pivot = liste[g][0]` (par exemple...)

❑ **Q15** – Soit on ajoute au début (ligne 2) :

```
if d - g <= 15:
    triInsertion(liste, g, d)
else:
```

(et on indente tout le reste), soit on modifie à partir de la ligne 16 :

```
if g < j:
    if j - g > 15:
        triRapide(liste, g, j)
    else:
        triInsertion(liste, g, j)
if i < d:
    if d - i > 15:
        triRapide(liste, i, d)
    else:
        triInsertion(liste, i, d)
```

❑ **Q16** –

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

❑ **Q17** – On peut économiser en calculant la moyenne une seule fois avant la boucle.

❑ **Q18** – La complexité pour `K` est du même ordre de grandeur, le même nombre de termes dans la somme et les opérations sont du même type.

Partie IV. Base de données relationnelle

❑ **Q19** –

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

```
SELECT idBouee FROM Bouee
EXCEPT
SELECT idBouee FROM Tempete;
```

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


Partie V. Analyse spectrale

❑ **Q20** – Comme pour le tri fusion, on a une complexité en $T(N) = 2T(N/2) + O(N)$, le $O(N)$ correspondant au partitionnement de x en termes d'indices pairs et impairs puis au calcul des X_k.
On a alors $\boxed{T(N) = O(N \ln N).}$

❑ **Q21** –

```
from numpy import exp, pi

def FFT(x):
    if len(x) == 1:
        return x
    xp, xi = x[::2], x[1::2] # indices pairs / impairs
    P, I = FFT(xp), FFT(xi)
    N = len(x)
    w = exp(-2j * pi / N)
    wk = 1 # va contenir w**k
    X = [0 for _ in range(N)]
    for k in range(N//2):
        X[k] = P[k] + wk*I[k]
        X[k + N//2] = P[k] - wk*I[k]
        wk *= w
    return X
```