Concours communs polytechniques (CCP) - Filière PSI

Jean-Loup Carré

Informatique commune – 2016

Culture générale

Le mot *pixel* a plusieurs définitions.

Il peut désigner un « petit carré » qui compose l'image, auquel cas un pixel est composé de plusieurs couleurs (en fait des longueurs d'onde), par exemple il y a 3 couleurs en RGB, et 352 dans le cas étudié ici.

Il peut aussi désigner une seule couleur d'un de ces « petits carré » (soit le rouge, soit le vert soit le bleu dans le cas particulier du RGB).

Dans ce sujet, c'est la seconde définition qui est utilisée.

Q1. Un cube de données hyperspectrales a 64 pixels dans chaque direction spaciale, et 352 longueurs d'onde. Soit un total de $64^2 \times 352$ pixels. Chaque pixel est codé sur 12 bits. Le cube prend donc $64^2 \times 352 \times 12$ bits soit $64^2 \times 352 \times 12/8$ octets.

Première solution : poser les multiplications au brouillon. $64^2 \times 352 \times 12/8 = 32^2 \times 352 \times 6$. En posant la multiplication à la main on obtient 2 162 688 octets, soit environs 2 Mo

Seconde solution : remarquer que le nombre recherché est presque une puissance de 2. En factorisant 352 par la plus grand puissance de 2 possible, on remarque que $352 = 11 \times 2^5$.
D'où $64^2 \times 352 \times 12/8 = (2^6)^2 \times 2^5 \times 11 \times 3 \times 2^2 \times 2^{-3} = 33 \times 2^{16} \approx 32 \times 2^{16} = 2^{21}$.
Or 2^{21} octets font 2 Mio soit environs 2 Mo.

Culture générale

Le mébioctet (abrégé en Mio) vaut 2^{20} octets soit 1024^2 octets. Un mébioctet vaut approximativement un mégaoctet (Mo). Un mégaoctet vaut 10^6 octets (de la même manière qu'un mégajoule vaut 10^6 joules).
Le préfixe "mébi-" a été créé pour les informaticiens préférant raisonner en base 2 plutôt qu'en base 10. Historiquement, d'aucuns utilisèrent le préfixe "méga-" pour 2^{20} au lieu de 10^6, cependant cette utilisation a été jugée abusive par différents organismes de normalisation : Commission électronique internationale, Bureau international des poids et mesures, etc.

Remarque

Étant donné que la calculatrice est interdite, je conjecture que donner une valeur approximative de 2 Mo est suffisant comme réponse.

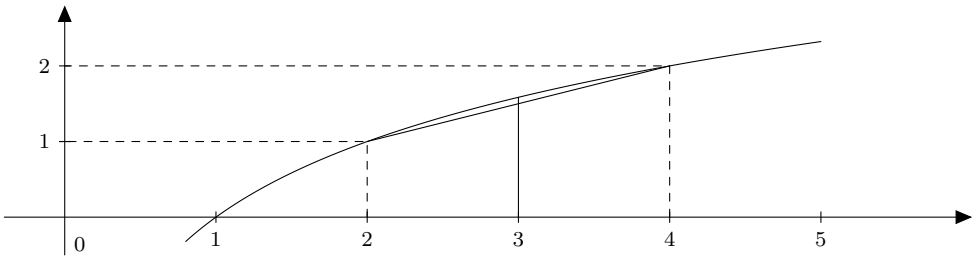
Q2. Le taux de compression recherché est d'approximativement $\frac{2\text{ Mo}-1\text{ Mo}}{1\text{ Mo}} = 50\%$.

Q3. Les valeurs 4 et 7 (deux valeurs) apparaissent 3 fois, les autres valeurs (0, 1, 5 et 8 soit 4 valeurs) n'apparaissent qu'une seule fois.

$$H(S_n) = \underbrace{2 \times (-0.3\log_2(0.3))}_{\text{Contributions de 4 et 7}} + 4 \times (-0.1\log_2(0.1)) = -0.6\log_2(3) + \log_2(10).$$

Pour approximer $\log_2(10)$, on utilise l'égalité bien connue $10^3 = 1000 \approx 1024 = 2^{10}$. D'où $3\log_2(10) \approx 10$ et donc $\log_2(10) \approx 3.33$.

Pour approximer grossièrement $\log_2(3)$ on fait une approximation affine de $\log_2$ entre 2 et 4, comme illustré sur le graphique ci-dessous.



On a $\log_2(3) \approx \frac{\log_2(2)+\log_2(4)}{2} = 1.5$.

Finalement, $H(S_n) \approx -0.6 \times 1.5 + 3.33 = 2.43$

Cette valeur est proche de la valeur de 2.4 obtenu par le codage du tableau 1, ce codage est donc à peu près optimal.

Remarque

Pour pouvoir comparer avec la valeur 2.4 de l'énoncé, on est obligé de faire une estimation numérique. En l'absence de calculatrice, on se contente d'une approximation grossière. À la calculatrice on trouve 2.37, ainsi notre estimation rapide est raisonnable.

Q4, Q5, Q6.

```
def entropie (S):
```

Question 4

```
    # On crée une liste contenant un exemplaire
    # de chaque valeur présente dans S
```

Question 5

```
    valeurs = list(set(S))
    proba=[0]*len(valeurs)
    n = len(valeurs)
    for i in range(n):
        occ_i = 0
        for x in S:
            if valeurs[i] == x:
                occ_i +=1
        proba[i] = occ_i/n
```

Question 6

```
    H = 0
    for k in range(len(valeurs)):
        H -= proba[i] * log2(proba[i])
    return H
```

Q7.

```
H = entropie(bloc_image)
print((12-H)/12)
```

Rapport du jury

« Q7 : Abordée par 60% des candidats, cette question a été assez mal traitée. Beaucoup de candidats n'ont pas fait appel à l'instruction print pour l'affichage du résultat. »

Remarque

Lorsqu'une question parle *d'afficher*, il faut utiliser `print`, sinon, ne l'utilisez pas.

Q8.

```
def pretraitement12(donnees_brutes):
    # Prétraitement 1
    luminance_moy = zeros(64)
    for j in range(64):
        for k in range(1, 32, 10):
            luminance_moy[j] += donnees_brutes[k, j]
        luminance_moy[j] /= 4
    # Prétraitement 2
    j_max = 0
    for j in range(64):
        if luminance_moy[j] > luminance_moy[j_max]:
            j_max = j
    return luminance_moy, luminance_moy[j_max], j_max
```

Q9.
```
def pretraitement34(donnees_brutes, luminance_max, j_max):
    return donnes_brutes[:, jmax]/luminance_max
```

Q10.
```
def pretraitement5(luminance_moy, spectre_max):
    matrice_modele = zeros([32, 64])
    for i in range(32):
        for j in range(64):
            matrice_modele[i, j] = spectre_max[i] * luminance_moy[j]
    return matrice_modele
```

Q11. Étudions `pretraitement12` :

- La complexité de `zeros` est en $\mathcal{O}(m)$.
- Les deux `for j` sont exécutés m fois.
- Le `for k` est exécuté 4 fois.

La complexité totale de cette fonction est donc en $\mathcal{O}(m) + m \times 4 \times \mathcal{O}(1) + m \times \mathcal{O}(1) = \mathcal{O}(m)$.

La complexité de `pretraitement34` est en $\mathcal{O}(n)$ car la fonction recopie une colonne

Étudions `pretraitement5` :

- La complexité de `zeros` est en $\mathcal{O}(nm)$,
- Le `for i` est exécuté n fois,
- Le `for i` est exécuté m fois.

La complexité totale de cette fonction est donc en $\mathcal{O}(nm) + m \times n \times \mathcal{O}(1) = \mathcal{O}(nm)$.

La complexité totale du prétraitement est donc en $\mathcal{O}(nm)$.

Q12.
```
def prediction(x):
    Delta = []
    for k in range(1, len(x)):
        Delta.append(x[k]-x[k-1])
    return Delta
```

Culture générale

En accord avec la PEP 3131, on peut utiliser les lettres Unicode (par exemple des lettres grecques) dans le code Python. Ainsi, il est possible d'utiliser dans le code le Δ de l'énoncé.

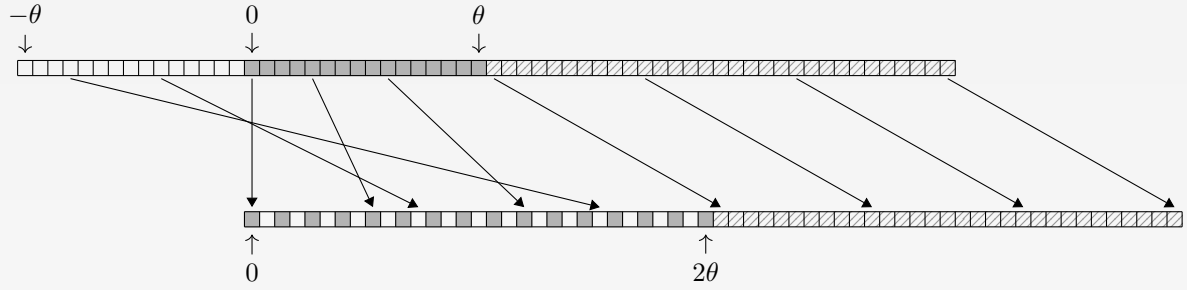
```
def prediction(x):
    Δ = []
    for k in range(1, len(x)):
        Δ.append(x[k]-x[k-1])
    return Δ
```

Cependant, il est recommandé de ne pas le faire à ce concours, étant donné que le jury, dans son rapport, se montre très hostile à l'utilisation de caractères non-ASCII.

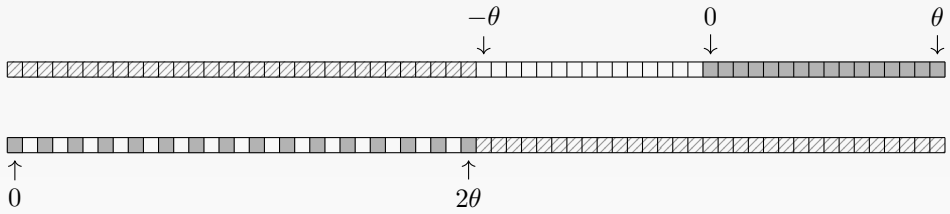
Q13.
```
def mappage(erreur, x):
    delta = []
    for k in range(len(erreur)):
        theta = min(x[k], 4095-x[k])
        A = abs(erreur[k])
        if 0 <= erreur[k] <= theta:
            return 2*A
        elif -theta <= erreur[k] < 0:
            return 2*A-1
        else:
            return theta + A
```

Culture générale

Le mappage peut être résumé par le dessin suivant (lorsque $\theta_k = \widehat{x}_k$). Les « petites » valeurs positives (en gris) sont envoyés sur les « petits » nombres pairs, et les entiers négatifs (en blanc) sont envoyés sur les « petits » nombres impairs. Les « grands » nombres (en hachuré) sont décalés à la fin.



Lorsque $\theta_k = x_{\max}\widehat{x}_k$, on a une situation similaire, mais cette fois-ci ce sont les « grands » entiers négatifs (en hachuré) qui sont envoyés vers les « grands » entiers.



Pour plus de détails sur ce mappage, on pourra consulter le rapport *Some Practical Universal Noisless techniques, Part III, Module PSI14K+* écrit par Robert F. Rice le 15 novembre 1991 et disponible sur le site de la NASA.

Q14.

Décimal	Code de Rice
4	0100
10	10010
18	110010
18	110010
6	0110
1	0001
3	0011

Q15, Q16.

```
def codage(delta_k, p_opt):
```

Question 15

```
    code1 = "1" * (delta_k // 2**p_opt) + "0"
```

Question 16

```
    code3 = bin(delta_k % 2**p_opt)[2:]
    code2 = "0" * (p_opt - len(code3)) + code3
    return code1 + code2
```

Remarque

Selon la question 15 du sujet, `code1` est « *un tableau à une ligne* » ; et l'annexe précise que les tableaux à une dimension sont représentés en Python soit par une liste, soit par ce que le sujet appelle un « vecteur » (en réalité, un `ndarray` de `numpy`). Néanmoins, selon le rapport du jury, il faudrait comprendre de l'énoncé que `code1` n'est pas un tableau mais une chaîne de caractères.
J'ai ici donné un corrigé conforme au rapport.

Q17. Cette équation provient du principe fondamental de la dynamique (qu'on a divisée par la masse de la sonde).

$$\underbrace{\frac{\mathrm{d}^2\overrightarrow{r}(t)}{\mathrm{d}t^2}}_{\text{Accélération de la sonde}} = \underbrace{-Gm_s\frac{\overrightarrow{r}(t)}{||\overrightarrow{r}(t)||^3}}_{\text{Pesanteur du Soleil}} \underbrace{-Gm_v\frac{\overrightarrow{r}(t)-\overrightarrow{r}_v(t)}{||\overrightarrow{r}(t)-\overrightarrow{r}_v(t)||^3}}_{\text{Pesanteur de Vénus}}$$

Q18. On déroule les définitions de $\overrightarrow{r}(t)$, de $x_v(t)$ et de $y_v(t)$ et on obtient :

$$S_x(x,y,t) = -Gm_s\frac{x}{\sqrt{x^2+y^2}^3} - Gm_v\frac{x-r_v\cos(\Omega t+\Phi)}{\sqrt{(x-r_v\cos(\Omega t+\Phi))^2+(y-r_v\sin(\Omega t+\Phi))^2}^3}$$

$$S_y(x,y,t) = -Gm_s\frac{y}{\sqrt{x^2+y^2}^3} - Gm_v\frac{y-r_v\sin(\Omega t+\Phi)}{\sqrt{(x-r_v\cos(\Omega t+\Phi))^2+(y-r_v\sin(\Omega t+\Phi))^2}^3}$$

```
def eval_sm(x, y, t):
    Gms = 1.32724e20
    Gmv = 3.24916e14
    rv = 1.08209e11
    Omega = 3.23639e-7
    Phi = 0  # Remplacer par la bonne valeur de Φ, qui n'est pas donnée.
    norm3 = (x**2+y**2)**1.5  # Cube de la distance au Soleil
    xv = rv * cos(Omega*t+Phi)
    yv = rv * sin(Omega*t+Phi)
    dv3 = ((x-xv)**2 + (y-yv)**2)**1.5 # Cube de la distance à Vénus
    Sx = - Gms * x / norm3 - Gmv * (x - xv)/dv3
    Sy = - Gms * y / norm3 - Gmv * (y - yv)/dv3
    return Sx, Sy
```

Q19.
$$\begin{aligned} u_{i+1} &= u_i + S_x(x(t),y(t),t)\Delta t \\ v_{i+1} &= v_i + S_y(x(t),y(t),t)\Delta t \end{aligned}$$

Q20. $$\begin{array}{rcl} x_{i+1} & = & x_i + u_i \Delta t \\ y_{i+1} & = & y_i + v_i \Delta t \end{array}$$

Q21.
```
x = [x0]
y = [y0]
u = [u0]
v = [v0]
t = arange(0, n, 1) * deltaT
```

Q22.
```
for i in range(0, n-1):
    x.append(x[i] + u[i] * deltaT)
    y.append(y[i] + v[i] * deltaT)
    Sx, Sy = eval_sm(x[i], y[i], t[i])
    u.append(u[i] + Sx * deltaT)
    v.append(v[i] + Sy * deltaT)
```

Rapport du jury

Ne créez pas de fonctions si le sujet vous demande simplement une suite d'instructions. Dans le rapport, il est écrit : « *Lorsqu'il est demandé dans le sujet d'écrire une suite d'instructions, les correcteurs n'attendent pas l'écriture d'une fonction.* »

Q23. Le nombre total de secondes sur la période étudiée est $30 \times 24 \times 60 \times 60$, il y a 5 paramètres à retenir (x, y, u, v et t) chacun sur 8 octets.
Il y a donc $30 \times 24 \times 60 \times 60 \times 5 \times 8$ octets ≈ 100 Mo.
Cette simulation est largement faisable à l'heure des disques durs de plusieurs teraoctets.

Q24.
```
def vitesse_sonde(u, v, t):
    N = [(u[k]**2+v[k]**5)**0.5/1000 for k in range(len(u))]
    plot(t, N)
    xlabel("Temps en secondes")
    ylabel("Vitesse en km/s")
```

Cette fonction prend en argument les listes `u`, `v` et `t` comme calculées aux questions 21 et 22 et renvoie `None`.