Mines Informatique commune MP 2015 — Corrigé

Ce corrigé est proposé par Julien Dumont (Professeur en CPGE) ; il a été relu par Guillaume Batog (Professeur en CPGE) et Jean-Julien Fleck (Professeur en CPGE).

Le sujet porte sur différents aspects de tests de validation d'une imprimante. Les parties sont indépendantes.

- La première partie porte sur la réception des données issues de la carte d'acquisition, en se concentrant plus particulièrement sur la trame contenant les informations.
- La deuxième, très courte et proche du cours, met en place des programmes de calcul approché d'intégrales, qui sont utilisés pour obtenir la moyenne et l'écart-type des données physiques reçues dans les trames.
- La troisième partie s'intéresse à la gestion d'un parc d'imprimantes en cours de validation à l'aide de bases de données.
- La quatrième aborde le thème de la compression de données, principalement en étudiant un long programme fourni en annexe.
- Enfin, la cinquième partie examine un schéma numérique de résolution d'équations différentielles afin de définir un test de validation des moteurs utilisés dans l'imprimante.

Ce sujet est intéressant sur le principe mais il est décevant : beaucoup de notions sont hors programme et ne sont pas définies ; le programme proposé dans la quatrième partie ne marche pas, ce qui conduit à des questions n'ayant pas vraiment de sens pour un candidat rigoureux ; les formulations des questions sont parfois très imprécises et on ne sait pas réellement ce qui est demandé... Bref, cet énoncé a tout pour désarçonner un candidat ayant sérieusement préparé le concours. Cependant, en vue des révisions, c'est un sujet qui balaie de nombreux domaines et, si on lit les indications du corrigé pour savoir ce qu'il est possible de faire ou pas, ce problème varié mérite que l'on s'y attarde.

Indications

> La mention (HP) dans une indication signale une question hors programme, pour laquelle une explication détaillée est fournie dans le corrigé.

Q1 (HP)

Q3 Penser à regarder soigneusement les structures demandées en retour de fonction.

Q4 Ne pas oublier le modulo.

Q5 (HP)

Q6 Cette question et la suivante se rapprochent des algorithmes de première année.

Q8 (HP) Faire comme si l'on pouvait définir une constante en SQL comme `Imin` ou `Imax`.

Q9 Utiliser des requêtes imbriquées.

Q10 Écrire une requête portant sur une table dans laquelle la clause `WHERE` dépend d'une requête portant sur une autre table.

Q11 (HP)

Q12 On suppose que le programme proposé marche.

Q14 (HP) On peut deviner la réponse en s'inspirant de l'énoncé, sans plus de connaissances sur les dictionnaires.

Q16 (HP) Un invariant d'itération est l'équivalent d'un invariant de boucle pour les fonctions récursives.

Q22 (HP)

Q23 Il faut construire ici une fonction permettant d'évaluer si un moteur est défectueux ou non. Par exemple, on peut proposer un test comparant quelques solutions simulées et la sortie mesurée.

Tests de validation d'une imprimante

Q1 Le complément à 2 est une méthode de représentation des entiers relatifs sur un nombre fini de bits. Un des bits est utilisé pour coder le signe, les suivants permettent de représenter la valeur absolue du nombre. La façon de représenter cette valeur absolue n'est pas nécessaire pour répondre à cette question. On donne ici une réponse générale indépendante de la convention, l'emploi de la terminologie « complément à 2 » imposant en fait l'intervalle demandé.

En complément à 2, puisqu'un bit sert à coder le signe, cela signifie qu'il en reste 9 pour coder la valeur absolue, soit $2^9 = 512$ valeurs possibles. Traditionnellement, **on considère que l'on code ici les entiers compris entre -512 et $+511$.** Mais on peut représenter de façon générale tout intervalle de nombres entiers contenant $2^{10} = 1024$ valeurs, si l'on décide arbitrairement d'une convention.

Le résultat de la conversion peut prendre toute plage de valeurs entières de 1024 valeurs.

Cependant, rien n'interdit de coder plusieurs fois une même valeur. Ainsi, une autre méthode de représentation de nombres qui consiste à coder le signe par le premier bit et la valeur absolue par les suivants en tant qu'entier naturel conduit à coder deux fois le zéro. Par exemple, sur 4 bits, les binaires 0000 et 1000 représentent « respectivement » $+0$ et -0, codant deux fois la même chose. L'intérêt des différentes méthodes de représentations de nombres est précisément de pallier ce genre de défaut.

[1]Q2 La résolution de la mesure se déduit de la possibilité de représenter 1024 éléments sur $N = 10$ bits. 2^N éléments permettent de définir $2^N - 1$ intervalles. La plage de tension P étant de 10 V, la résolution σ vaut

$$\sigma = \frac{P}{2^N - 1} = 1.10^{-2} \text{ V}$$

Q3 Le principe du programme est le suivant. On initialise une variable `nonstop` à la valeur booléenne `True`. On lit alors un à un les caractères reçus jusqu'à tomber sur un caractère d'en-tête. On change alors la valeur de `nonstop` pour sortir de la boucle. On lit par la suite le nombre N de données envoyées. On sait que la longueur totale de la trame est exactement $8 + 4N$: un caractère correspond à l'en-tête, trois au nombre de données envoyées, 4N permettent de détailler ces dernières et quatre caractères donnent le checksum. On utilise également à répétition la conversion du type chaîne de caractères vers le type entier grâce à `int`.

```
def lect_mesures():
    '''Fonction qui renvoie une trame à partir du premier en-tête'''
    nonstop=True
    resultat=[ ] #Liste que l'on renverra à la fin
    ###Recherche de l'en-tête
    while nonstop:
        test=com.read(1)
        if test in ['U','I','P']:#A-t-on trouvé l'en-tête ?
            resultat=[test]      #Si oui, on le stocke
            nonstop=False    #Et on fait en sorte de sortir du while
```

```
    ###Lecture du nombre de données à venir
    N=int(com.read(3)) #On stocke le nombre de données à lire
    ###Lecture des données
    donnees=[ ] #Liste vide contenant les données
    for inc in range(N):
        #On lit 4 caractères que l'on convertit
        #en entier pour les stocker
        donnees.append(int(com.read(4)))
    resultat.append(donnees) #On stocke donnees
    ##Ajout du checksum
    resultat.append(int(com.read(4)))
    return resultat
```

Q4 Notons une confusion de l'énoncé entre `mesure` et `mesures` qui peut déstabiliser à la lecture du sujet, surtout lors de l'emploi de la notation `mesures[]`, utilisée par exemple en Java... Ce qui est demandé est toutefois relativement compréhensible entre les lignes. Enfin, ne pas oublier le modulo 10000.

```
def check(mesure,CheckSum):
    '''Vérifie la validité d'un checksum'''
    #On calcule la somme des données reçues
    somme=0
    for x in mesure:
        somme += abs(x)
    #On teste la validité du checksum
    return somme%10000==CheckSum
```

Q5 On suppose ici que la bibliothèque `matplotlib.pyplot` a été importée sous l'alias `plt`.

```
def affichage(mesure):
    ###Création du vecteur des temps
    Temps=[ ]
    #On connaît exactement la plage nécessaire : on part de 0ms
    #et on va à 400ms par pas de 2ms. On met donc 401 pour atteindre
    #400 inclus mais sans dépasser cette valeur.
    for t in range(0,401,2):
        Temps.append(t)
    ###Création du vecteur des mesures
    mesures=[ ] #Initialisation des mesures
    for m in mesure: #On balaie les données fournies
        mesures+=[m*4e-3] #Conversion des données en intensités
    ###Représentation graphique proprement dite
    plt.plot(Temps,mesures)
    ###Compléments : axes et titre
    plt.xlabel('Temps (ms)')
    plt.ylabel('Intensité (A)')
    plt.title('Courant moteur')
```

Ce programme suivant est enrichi des commandes qui auraient permis d'obtenir tout le graphique proposé. Selon l'interface de développement, on peut être amené à ajouter la fonction `show` pour que le graphique créé s'affiche effectivement. On peut également le sauver à l'aide de la fonction `savefig`.

Notons toutefois que, d'après le programme officiel, aucune commande