ÉCOLE DES PONTS PARISTECH.
SUPAERO (ISAE), ENSTA PARISTECH,
TELECOM PARISTECH, MINES PARISTECH
MINES DE SAINT-ÉTIENNE, MINES NANCY,
TÉLÉCOM BRETAGNE, ENSAE PARISTECH (Filière MP).
ÉCOLE POLYTECHNIQUE (Filière TSI).

CONCOURS 2015

ÉPREUVE D'INFORMATIQUE

(Durée de l'épreuve : 1h30 heures)
L'usage de l'ordinateur ou de la calculatrice est interdit.

Sujet mis à la disposition des concours :
Cycle international, Écoles des Mines, TELECOM INT, TPE-EIVP.

L'énoncé de cette épreuve comporte 10 pages de texte.

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Tests de validation d'une imprimante

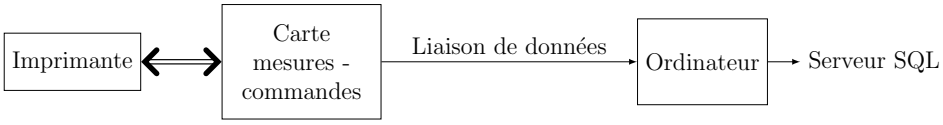
Le sujet comporte des questions de programmation. *Le langage à utiliser est* `Python` *dans les parties I à IV. Dans la partie V, on demande d'utiliser* `Scilab`.

Introduction

Les imprimantes sont des systèmes mécatroniques fabriqués en grande série dans des usines robotisées. Pour améliorer la qualité des produits vendus, il a été mis en place différents tests de fin de chaîne pour valider l'assemblage des produits. Pour un de ces tests, un opérateur connecte l'outil de test sur la commande du moteur de déplacement de la tête d'impression et sur la commande du moteur d'avance papier. Une autre connexion permet de récupérer les signaux issus des capteurs de position.

Différentes commandes et mesures sont alors exécutées. Ces mesures sont envoyées par liaison de données sous la forme d'une suite de caractères ASCII vers un ordinateur.

Cet ordinateur va effectuer différentes mesures pour valider le fonctionnement de l'électromécanique de l'imprimante. L'ensemble des mesures et des analyses est sauvegardé dans un fichier texte. Cette sauvegarde s'effectue dans une base de données et, afin de minimiser l'espace occupé, les fichiers sont compressés. La base de données permet à l'entreprise d'améliorer la qualité de la production par diverses études statistiques.



Rappels et définitions :

- Une liste commence par un crochet `[` et se termine par un crochet `]`. Les éléments d'une liste sont ordonnés (indexés).
- On pourra utiliser la surcharge de l'opérateur + : `['a']+['b']=['a','b']`.
- Un dictionnaire définit une relation une à une entre des clés et des valeurs. Celui-ci se note entre accolades `{}`.
- Un tuple est une collection d'éléments ordonnés comme dans une liste mais une fois le tuple créé, ses éléments ne peuvent pas être modifiés indépendamment les uns des autres. Il se note entre parenthèses `()`. Exemple : (4,'e',[1,3])

I Réception des données issues de la carte d'acquisition

Les mesures sont faites à l'aide de convertisseurs analogique/numérique (CAN). Le résultat de conversion est codé sur 10 bits signés en complément à 2.

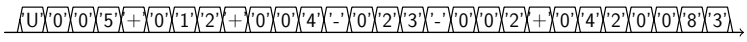
Q1. Quelle plage de valeurs entières pourra prendre le résultat de la conversion ?

Q2. Si on considère que les valeurs analogiques converties s'étendent en pleine échelle de -5V à 5V, quelle est la résolution de la mesure en volt ?

Une liaison série asynchrone permet la communication entre la carte de commande/acquisition et le PC. Les échantillons correspondant à une mesure sont envoyés par la carte électronique sous la forme d'une trame (suite de caractères ASCII). Cette suite de caractères se présente sous la forme suivante :

- un entête qui permet d'identifier la mesure sur un caractère ('U' tension moteur, 'I 'courant moteur, 'P' position absolue),
- le nombre de données envoyées (3 caractères),
- les données constituées des mesures brutes issues de la conversion analogique-numérique, chaque mesure étant codée à l'aide du caractère '+' ou '-' suivi de 3 caractères pour la valeur absolue,
- un checksum, somme des valeurs absolues des données précédentes modulo 10000 sur 4 caractères. Le nombre de données transmises n'est pas inclus dans le checksum.

Exemple : Mesure de la tension sur 5 échantillons.

Caractères reçus : 

La commande `carac_recus=com.read(nbre_car)` permet de récupérer *nbre_car* caractères reçus sous la forme d'une chaîne de caractères. En supposant que les caractères reçus correspondent à l'exemple précédent, après l'exécution de `carac_recus=com.read(5)`, la variable `carac_recus` contiendra la chaîne "U005+".

Après une nouvelle exécution de `carac_recus=com.read(3)`, la variable `carac_recus` contiendra la chaîne "012".

Q3. Écrire une fonction `lect_mesures()` en langage `Python` qui retourne une liste contenant : le type de la mesure ('U','I' ou 'P'), une liste contenant l'ensemble des valeurs des mesures reçues et le checksum. Exemple : ['U',[12,4,-23,-2,42],83]. Cette fonction doit attendre que le caractère d'entête reçu soit correct ('U','I' ou 'P') avant de réaliser le stockage des informations dans la liste qui sera retournée.

Q4. On suppose que toutes les mesures sont disponibles dans la liste `mesures[]`, et le checksum reçu dans la variable `CheckSum`. Écrire une fonction `check(mesure,CheckSum)` en langage `Python` qui retourne True si la transmission présente un checksum valide et False sinon.

Q5. Les mesures étant dans la liste `mesures`, écrire une fonction `affichage(mesure)` en langage `Python` qui produit un affichage graphique comme représenté en Figure 1, sachant que la résolution de la conversion analogique-numérique du courant est de 4 mA et que les mesures ont été effectuées toutes les 2 ms. On ne demande pas de légender les axes ni de donner un titre à la figure. On suppose que les bibliothèques nécessaires ont été importées.

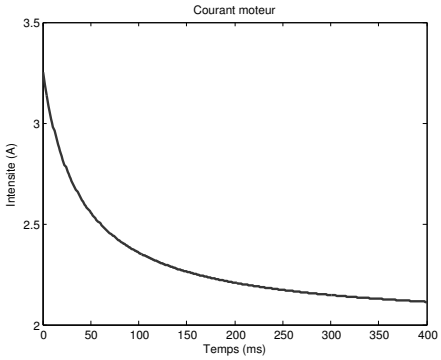


FIGURE 1 – Affichage de la mesure.

II Analyse des mesures

La suite des valeurs de mesure du courant en Ampère du moteur de la tête d'impression est contenue dans une liste. Les mesures ont été effectuées toutes les 2 ms. Ces mesures sont disponibles dans la liste `mesure`. Deux traitements permettent de valider le fonctionnement de l'imprimante :

- Le calcul de la valeur moyenne I_{moy} du signal $I(t)$ sur la durée d'acquisition.

$$I_{moy} = \frac{1}{t_{final}} \int_0^{t_{final}} I(t)\,\mathrm{d}t$$

- Le calcul de l'écart type I_{ec} du signal $I(t)$ sur la durée d'acquisition.

$$I_{ec} = \sqrt{\frac{1}{t_{final}} \int_0^{t_{final}} (I(t) - I_{moy})^2\,\mathrm{d}t}$$

Q6. Écrire une fonction en langage `Python` qui retourne I_{moy} après l'avoir calculée par la méthode des trapèzes.

Q7. Écrire une fonction en langage `Python` qui retourne I_{ec} après l'avoir calculé en utilisant la fonction précédente.

III Base de données

Une représentation simplifiée de deux tables de la base de données qu'on souhaite utiliser est donnée ci-dessous :

testfin

nSerie	dateTest	...	Imoy	Iec	...	fichierMes
230-588ZX2547	2012-04-22 14-25-45		0.45	0.11		mesure31025.csv
230-588ZX2548	2012-04-22 14-26-57		0.43	0.12		mesure41026.csv
⋮	⋮	⋮	⋮	⋮	⋮	⋮

production

Num	nSerie	dateProd	type
20	230-588ZX2547	2012-04-22 15-52-12	JETDESK-1050
21	230-588ZX2549	2012-04-22 15-53-24	JETDESK-3050
⋮	⋮	⋮	⋮

Après son assemblage et avant les différents tests de validation, un numéro de série unique est attribué à chaque imprimante. A la fin des tests de chaque imprimante, les résultats d'analyse ainsi que le fichier contenant l'ensemble des mesures réalisées sur l'imprimante sont rangés dans la table `testfin`. Lorsqu'une imprimante satisfait les critères de validation, elle est enregistrée dans la table `production` avec son numero de série, la date et l'heure de sortie de production ainsi que son type.

Q8. Rédiger une requête SQL permettant d'obtenir les numéros de série des imprimantes ayant une valeur de Imoy comprise strictement entre deux bornes Imin et Imax.

Q9. Rédiger une requête SQL permettant d'obtenir les numéros de série, la valeur de l'écart type et le fichier de mesures des imprimantes ayant une valeur de Iec strictement inférieure à la valeur moyenne de la colonne Iec.

Q10. Rédiger une requête SQL qui permettra d'extraire à partir de la table `testfin` le numéro de série et le fichier de mesures correspondant aux imprimantes qui n'ont pas été validées en sortie de production.

IV Préparation du fichier texte avant envoi : la compression

Le fichier de résultat va être stocké sous la forme d'un fichier binaire. Une des étapes de l'algorithme de compression utilise le codage de Huffman.

IV.1 Présentation :

Le codage de Huffman utilise un code à longueur variable pour représenter un symbole de la source (par exemple un caractère dans un fichier). Le code est déterminé à partir d'une estimation des probabilités d'apparition des symboles de source, un code court étant associé aux symboles de source les plus fréquents. La première étape du codage de Huffman consiste à créer un

dictionnaire contenant la liste des caractères présents dans le texte, associé à leur fréquence dans ce texte. Exemple : "AABCDCCEF" donnera `{'A':2,'B':1,'C':3,'D':1,'E':1,'F':1}`. La deuxième étape consiste à construire un arbre de Huffman qui permet ensuite de coder chaque caractère.

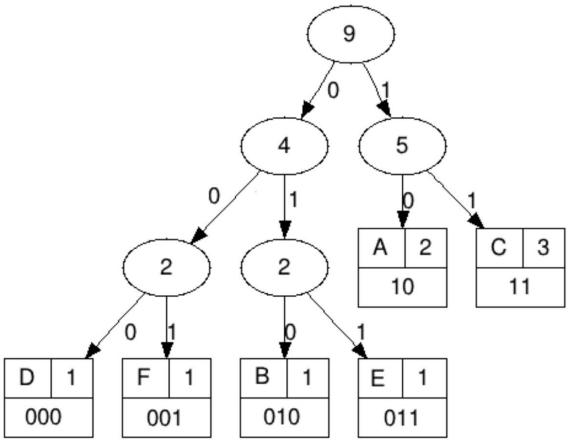


FIGURE 2 – Arbre de Huffman.

Notre texte "AABCDCCEF" de 9 caractères ASCII (72 bits) sera ainsi codé en binaire "10 10 010 11 000 11 11 011 001" (22 bits).

Chaque caractère constitue une des feuilles de l'arbre à laquelle on associe un poids valant son nombre d'occurrences. Puis l'arbre est créé suivant un principe simple : on associe à chaque fois les deux nœuds de plus faibles poids pour créer un nœud dont le poids équivaut à la somme des poids de ses fils jusqu'à n'en avoir plus qu'un, la racine. On associe ensuite par exemple le code 0 à la branche de gauche et le code 1 à la branche de droite.

Pour obtenir le code binaire de chaque caractère, on descend sur la Figure 2 de la racine jusqu'aux feuilles en ajoutant à chaque fois au code un 0 ou un 1 selon la branche suivie.

Pour pouvoir être décodé par l'ordinateur, l'arbre doit aussi être transmis.

Le code `Python` permettant de construire un arbre de Huffman et de coder chaque caractère est fourni en *annexe*.

Les feuilles de l'arbre de Huffman (leaf) sont codées sous la forme de tuple avec comme premier élément le caractère et comme deuxième élément le poids. Pour l'arbre donné en exemple en Figure 2, on aura 6 tuples : `('A',2), ('B',1), ('C',3), ('D',1), ('E',1)` et `('F',1)`.

Q11. La documentation de l'instruction `isinstance(object,classinfo)` décrit le fonctionnement suivant : "Return True if the object is an instance of the classinfo argument. If object is not a class instance of the given type, the function returns False." Décrire succinctement le rôle des fonctions suivantes et indiquer le type de la variable retournée :

- `test()`
- `get1()`
- `get2()`

IV.2 Analyse des fonctions `make_huffman_tree()` et `freq_table()`

Q12. Donner le contenu des variables `node1` et `node2` suite à l'exécution des commandes
`node1=make_huffman_tree(('F',1),('E',1))`.
`node2=make_huffman_tree(('D',1),('B',1))`.

Q13. De même, donner le contenu de la variable `node3` suite à l'éxécution de la commande `node3=make_huffman_tree(node1,node2)`.

Q14. Donner le contenu de la variable `f` suite à l'éxécution de la commande `f=freq_table('AABBCB')`.

IV.3 Analyse de la fonction `insert_item()`

Cette fonction permet d'insérer un nœud ou une feuille dans une liste de nœuds et de feuilles triés par poids croissant, en conservant l'ordre de tri.

Q15. Quelle est la particularité de cette fonction ?

Q16. Montrer qu'un invariant d'itération est "tous les éléments de la sous liste `lst[0 à pos-1]` ont un poids inférieur à celui de la variable item". On démontrera qu'il est vrai à l'initialisation, puis à chaque itération, sachant que cette fonction doit être appelée avec l'argument d'entrée `pos=0` produisant ainsi une liste vide.

IV.4 Analyse de `build_huffman_tree()`

Q17. D'après la description précédente et le résultat de la question Q13, commenter la fonction de manière à expliquer comment l'arbre de Huffman est construit. On demande de proposer des rédactions pour les commentaires correspondant aux lignes ## 5, ## 6, ## 7, ## 8 dans la fonction `build_huffman_tree()`.

Q18. Donner le nombre de tests dans le meilleur des cas et dans le pire des cas effectués dans la fonction `insert_item()` pour une liste `lst` contenant n éléments. Les résultats devront être justifiés.

Q19. Donner la complexité en temps dans le meilleur des cas et dans le pire des cas de la fonction `build_huffman_tree` pour une liste `lst` contenant n éléments en tenant compte de la fonction `insert_item`. On négligera le coût en complexité de la fonction `lst.sort`. Les résultats devront être justifiés.

Q20. Donner le contenu de la variable `htree` après l'éxécution de la commande `htree=build_huffman_tree("ZBBCB")`.

Q21. Dessiner l'arbre de Huffman correspondant à `htree` de la question précédente en vous inspirant de la Figure 2.

V Simulation physique

Une possible source de pannes dans chaque imprimante est la défectuosité d'un moteur particulier. On suppose disposer d'enregistrements du signal d'entrée e et du signal de sortie s de cet élément. Ces variables sont supposées satisfaire une équation différentielle linéaire du premier ordre

$$\frac{d}{dt}s = -k\frac{s-e}{10}$$

où k est un paramètre réel strictement positif. On va chercher à vérifier le bon fonctionnement du moteur en analysant des simulations.

Q22. Écrire en langage Scilab une fonction qui calcule de manière approchée la solution de l'équation différentielle précédente pour le signal $e(t) = \sin(t)$, avec $s(0) = 0$, sur l'intervalle $t \in [0,10]$ pour une valeur quelconque de k. La fonction doit retourner une estimation de $s(t)$ aux instants [0, 0.1, 0.2,..., 10].

Q23. Trois valeurs sont possibles pour le paramètre k : 0.5, 1.1 et 2. On décide de déclarer le moteur défectueux si aucune de ces valeurs ne permet d'expliquer l'enregistrement s réalisé expérimentalement aux instants [0, 0.1, 0.2,..., 10]. Définir un code Scilab utilisant la fonction précédemment définie pour réaliser une validation ou une invalidation de la défectuosité du moteur suivant un critère à proposer.

Annexe

Huffman tree :

```
#############################################################################
### Auto Generation of Huffman Trees
#############################################################################

def make_leaf(symbol, weight):
    return (symbol, weight)

def test(x):
    return isinstance(x, tuple) and \
            len(x) == 2 and \
            isinstance(x[0], str) and \
            isinstance(x[1], int)

def get1(x):
    return x[0]

def get2(x):
    return x[1]

def get3(huff_tree):
    return huff_tree[0]

def get4(huff_tree):
    return huff_tree[1]

def get5(huff_tree):
    if test(huff_tree):
        return [get1(huff_tree)] #Attention le symbole est dans une liste
    else:
        return huff_tree[2]

def get6(huff_tree):
    if test(huff_tree):
        return get2q(huff_tree)
    else:
        return huff_tree[3]

def make_huffman_tree(left_branch, right_branch):
    return [left_branch,
            right_branch,
            get5(left_branch) + get5(right_branch),
            get6(left_branch) + get6(right_branch)]

### entr\'{e}e : string txt et retourne un dictionnaire contenant chaque
### caractere avec son occurrence dans le string txt
def freq_table(txt):
    ftble = {}
    for c in txt:
        if c not in ftble:
            ftble[c] = 1
        else:
            ftble[c] += 1
    return ftble
```

```
### Fonction de comparaison qui permet de comparer
### les noeuds de Huffman entre eux selon leur occurrence.
def freq_cmp(node1, node2):
    freq1, freq2 = get6(node1), get6(node2)
    if freq1 < freq2:
        return -1
    elif freq1 > freq2:
        return 1
    else:
        return 0

### ins\'{e}re un item \'{a} sa place appropri\'{e}e dans une liste de
   noeuds et feuilles
def insert_item(item, lst, pos):
    if pos == len(lst):
        lst.append(item)
    elif freq_cmp(item, lst[pos]) <= 0 :
        lst.insert(pos, item)
    else:
        insert_item(item, lst, pos+1)
    return

### Construction de l'arbre de Huffman
def build_huffman_tree(txt):
    ### 1. construire une table des occurrences \'{a} partir de txt
    ftble = freq_table(txt)
    ### 2. obtenir la liste des feuilles de Huffman
    lst = list(ftble.items())
    ### 3. classer leaf_lst par occurrence de la plus petite \'{a} la plus
    ###    grande
    lst.sort(key=lambda lst: lst[1])
    ### 4. construction de l'arbre de huffman
    if len(lst) == 0:
        return None
    elif len(lst) == 1:
        return lst[0]
    else:
        ## 5. ------------------------------------------------------------
          while len(lst) > 2:
            ## 6. ------------------------------------------------------
            new_node = make_huffman_tree(lst[0], lst[1])
            ## 7. ------------------------------------------------------
            del lst[0]
            del lst[0]
            ## 8. ------------------------------------------------------
            insert_item(new_node, lst, 0)
        else:
            return make_huffman_tree(lst[0], lst[1])
```

Manual and Auto Generation of Huffman Trees in Python, Vladimir Kulyukin `http://vkedco.blogspot.fr/2012/02/manual-and-auto-generation-of-huffman.html`
Huffman tree generator `http://huffman.ooz.ie/`