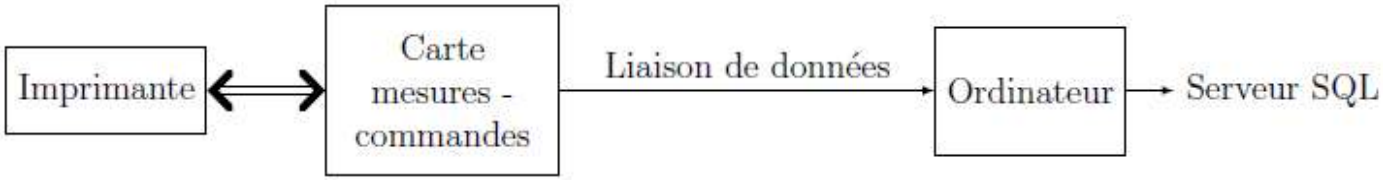
TP8 - TESTS DE VALIDATION D'UNE IMPRIMANTE

1. PRESENTATION

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 exécuter 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'es- pace occupé, les fichiers sont compressés. La base de données permet à l'entreprise d'améliorer la qualité de la production par diverses études statistiques.



2. RECEPTION DES DONNEES 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 : 'U''0''0''5''+''0''1''2''+''0''0''4''-''0''2''3''-''0''0''2''+''0''4''2''0''0''8''3' $\rightarrow t$

Q3. On suppose que toutes les mesures sont disponibles dans la liste `mesures[]`, et le checksum reçu dans la variable `CheckSum`. . Ecrire une fonction `check(mesure,CheckSum)` qui retourne `True` si la transmission présente un checksum valide et `False` sinon.

Vous pouvez ouvrir le programme `TP8_Huffman.py` pour tester avec une liste `mesure`.

Q4. Les mesures étant dans la liste mesures*, écrire une fonction* `affichage(mesure)` *qui produit un affichage graphique comme représenté en Figure1, 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.*

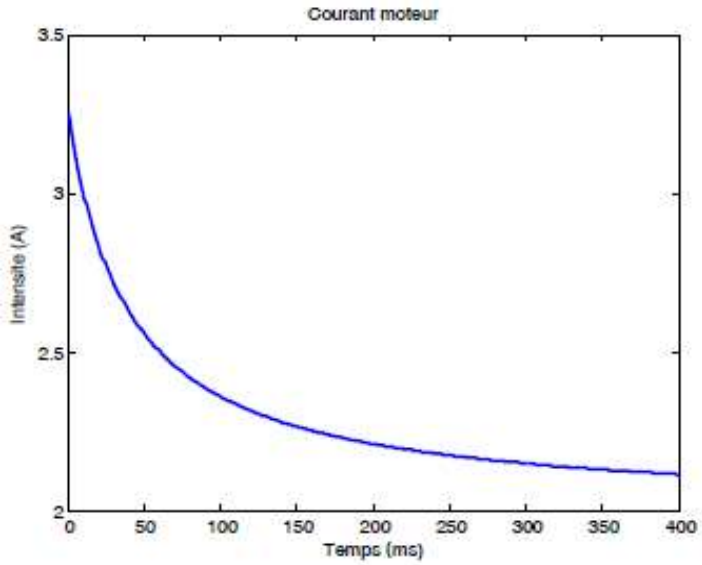


Figure 1

3. BASES DE DONNEES

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 rangées dans la table testfin. Lorsqu'une imprimante satisfait les critères de validation, elle est enregistrée dans la table production avec son numéro de série, la date et l'heure de sortie de production ainsi que son type.

Q5. 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.

Q6. 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.

Q7. 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.*

4. PREPARATION 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

4.1. 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.

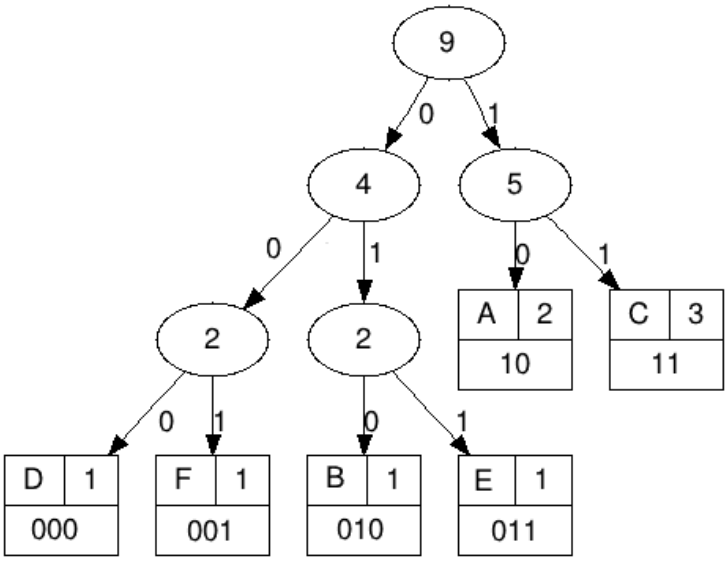


Figure 2

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 _a 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 noeuds de plus faibles poids pour créer un noeud 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).

4.2. Fonctions make_huffman_tree() et freq_table()

Q8. Analyser le code du programme `TP8_Huffman.py`. Puis 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)).
```

Q9. De même, donner le contenu de la variable `node3` suite à l'exécution de la commande `node3=make_huffman_tree(node1,node2)`.

Q10. Écrire une fonction `table_freq(txt)` qui étant donné une chaîne de caractère texte retourne un dictionnaire qui associe à chaque caractère son nombre d'occurrences dans texte.

L'appel de `table_freq("ABRACADABRA")` devra retourner un dictionnaire de la forme `{'A': 5, 'R': 2, 'B': 2, 'C': 1, 'D': 1}` (pas forcément dans cet ordre)

Afin de comprendre les dictionnaires voilà quelques petits exemples que vous pouvez utiliser :

```
##comprendre les dicos
mondico={'premier':1,'deuxième':2}
mondico2={1:'premier',2:'deuxième'}
print(mondico['premier'])
print(mondico['deuxième'])
print(mondico2[1])
print(mondico2[2])

#creation d'un dictionnaire
nouveau={}
nouveau['cle']=3
print(nouveau)
for b in mondico2:
    print(b) #retourne les clès
for b in mondico2:
    print(mondico2[b]) #retourne les valeurs associées aux clés

##La méthode get permet de récupérer une valeur dans un dictionnaire et si la
clé est introuvable, vous pouvez donner une valeur à retourner par défaut.
La méthode get prend donc deux arguments.
Si le premier argument est une clé de notre dictionnaire, alors elle renvoie
la valeur associée sinon elle renvoie le deuxième argument :

a={'nom': 'engel', 'prenom': 'olivier'}
a.get("prenom", "inconnue")
#'Olivier'
a.get("adresse", "Adresse inconnue")
#'Adresse inconnue'
```

4.3. Construction de l'arbre

Q11. Commentez les étapes de la fonction `encode_Huffman(dico)`

```
#test
T=freq_table("ABRACADABRA")
C = encode_Huffman(T)
print(C)
```

4.4. Encodage et décodage

Q12. Écrire une fonction `encodage(code,texte)` qui étant donné code de Huffman construit par la fonction précédente et le texte initial retourne la chaîne de bits produite par le codage de Huffman

L'appel de `encodage(C,"ABRACADABRA")`
*devra retourner un dictionnaire de la forme "**01111001100011010111100**"*

Q13. Écrire une fonction `decodage(code, texte_binaire)` qui étant donné code de Huffman un texte binaire compressé retourne le texte initial

On utilisera `code_inv = dict((code[str(bits)], bits) for bits in code)` pour construire le dictionnaire inverse.
L'appel de `decodage(C, "01111001100011010111100")`
*devra retourner un dictionnaire de la forme "**ABRACADABRA**"*

5. TRI SUR LE FICHIER DECOMPRESSE

Après avoir décompressé le fichier, l'imprimante effectue souvent un tri sur les listes de processus à effectuer. Ce tri est effectué sur la liste des taches sachant que chacune possède un numéro correspondant à son ordre de traitement dans la file d'attente. On propose ici d'utiliser l'algorithme de tri fusion, plus rapide sur des listes de taille importante, pour effectuer un tri.

*Q14. Créer la fonction **récursive** fusion (L1,L2) qui permet de faire la fusion de deux listes triées (L1 et L2) et de faire ainsi le tri fusion proposé dans la fonction suivante :*

```
def trifusion(L):   #Tri fusion
        if len(L)==1:
              return L
        else:
              return fusion( trifusion(L[:len(L)//2]) ,
trifusion(L[len(L)//2:]) )
```