Sujet Mines-Ponts 2015 réécrit en conformité avec le programme.
Durée nécessaire : 4h

Introduction

Les imprimantes sont des systèmes mécatroniques (combinaison synergique et systémique de la mécanique, de l'électronique et de l'informatique en temps réel) 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 transmises sous la forme d'une suite de caractères 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. Afin de minimiser l'espace occupé, les fichiers sont compressés. Une base de données stocke les informations concernant les mesures et les imprimantes sur lesquelles elles portent, et permet à l'entreprise d'améliorer la qualité de la production après diverses études statistiques.

Le problème comporte quatre grandes parties indépendantes, sauf par le thème traité.

- Dans la partie 1, on s'intéresse au processus de réception par l'ordinateur des données transmises par le capteur.
- Dans la partie 2, on procède au traitement des données numériques récupérées à fin de validation de l'imprimante assemblée.
- Dans la partie 3, on étudie quelques aspects de la base de données constituée.
- Dans la partie 4, on propose une introduction à la problématique de la compression de données.

Le sujet comporte des questions de programmation. Le langage à utiliser est Python sauf dans la partie 2 où le candidat est libre de choisir le langage Python ou le langage Scilab. Le candidat indique en tête de sa première copie s'il choisit le langage Python ou le langage Scilab pour les questions de programmation de la deuxième partie.

1 Réception des données issues de la carte d'acquisition

1.1 Le capteur

Le capteur utilisé est analogique, et comporte un convertisseur (appelé convertisseur analogique numérique) qui traduit les mesures effectuées sous forme numérique. Chaque donnée est ainsi codée comme un entier.

Q1. Rappeler un principe de représentation en machine des entiers signés sur 10 bits. Préciser la plage des valeurs entières représentables selon ce principe.

Q2. On considère que les valeurs analogiques s'étendent en pleine échelle de -5 V à 5 V et sont converties en entiers signés représentés sur 10 bits. Donner une valeur approchée de la résolution de la mesure en volts.

Dans la suite du problème, la question du codage des entiers en binaire n'intervient plus.

1.2 Liaison avec l'ordinateur

Une liaison série asynchrone permet la communication entre la carte de commande/acquisition et le PC. Les informations correspondant à une mesure sont envoyées par la carte électronique sous la forme d'une suite de caractères, appelée trame. Plusieurs trames consécutives correspondant à différents jeux de

mesures peuvent être transmises sans interruption, le format même des trames devant permettre de les délimiter. Un exemple de trame est :

'U'	'0'	'0'	'3'	'+'	'0'	'1'	'2'	'+'	'0'	'0'	'4'	'-'	'0'	'2'	'3'	'9'	'9'	'9'	'3'

Une trame est donc constituée de la manière suivante :

- un caractère d'en-tête, qui est une des trois lettres `'U'`, `'I'`, `'P'`, permettant d'identifier la nature de la mesure effectuée (`'U'` : tension moteur, `'I'` : courant moteur, `'P'` : position absolue) ;
- trois chiffres constituant une valeur entière précisant le nombre N de mesures qui constituent la suite de la trame ;
- un ensemble de N blocs consécutifs de quatre caractères. Chacun des blocs est constitué d'un caractère de signe `'+'` ou `'-'`, suivi de trois chiffres donnant une valeur absolue. L'entier signé ainsi codé sur quatre caractères donne une valeur issue de la conversion analogique/numérique d'une mesure ;
- un dernier bloc de quatre chiffres constituant ce qu'on appelle une somme de contrôle (*checksum*). Celle-ci est calculée en formant la somme des valeurs des données précédentes de la trame, et en prenant le reste de la division euclidienne par 10000 de cette somme.

La somme de contrôle (*checksum*) permet de vérifier si la trame a été correctement transmise. On peut, à réception d'une trame, calculer la somme de contrôle associée aux données reçues, et la comparer à la somme de contrôle contenue dans la trame. Si ces deux valeurs ne coïncident pas, les données ont été altérées lors de la transmission, et il convient de ne pas les exploiter.

La fonction `car_read(nbre_carac)` permet d'obtenir des caractères consécutifs issus de la liaison asynchrone. Elle prend en argument un entier `nbre_carac`, et renvoie une chaîne de caractères constituée de `nbre_carac` caractères. Ainsi, dans l'exemple ci-dessus, deux appels consécutifs à `car_read(5)` puis `car_read(3)`, renverront respectivement les chaînes `'U003+'` et `'012'`.

Q3. Toujours à partir de l'exemple donné plus haut, on suppose que les deux appels

`car_read(5)` puis `car_read(3)`

ont été effectués. Quelle valeur est alors renvoyée par l'appel `car_read(4)` ?

Le programme de lecture des données transmises à l'ordinateur lit des caractères jusqu'à obtention d'un caractère `x` d'en-tête égal à `'U'`, `'I'` ou `'P'`. Il fait alors appel à une fonction de lecture de trame `lecture_trame(x)` lisant les données numériques contenues dans la trame qui commence par `x`.

Q4. Écrire la fonction `lecture_trame(x)` prenant en argument le caractère d'en-tête `x` qui vient d'être lu, et renvoyant une liste `[x,L,S]`, où `L` est une liste d'entiers représentant les mesures transmises dans la trame commençant par `x`, et `S` est la valeur de la somme de contrôle transmise dans la trame commençant par `x`.

Dans la question suivante, on utilise une valeur issue d'un appel à la fonction

`lecture_trame(x)`.

Une telle valeur sera stockée dans une variable `trame` *via* une affectation

`trame = lecture_trame(x)`.

Q5. Écrire une fonction `checkSum(trame)` prenant en argument une liste `trame`, et renvoyant un booléen indiquant si le troisième élément de la liste `trame` est égal à la somme de contrôle calculée à partir des valeurs stockées dans le deuxième élément de cette liste.

Q6. Écrire une fonction `lecture_intensite()`, sans argument, et procédant au traitement suivant :

① obtenir un caractère jusqu'à ce que ce soit le caractère `'I'` ;
② lire la trame dont on vient de lire le caractère d'en-tête et stocker les informations dans une variable `trame` ;
③ si la variable `trame` ne présente pas une somme de contrôle valide, recommencer à ① ;
④ renvoyer la liste des mesures numériques alors contenues dans la variable `trame`.

On fait l'hypothèse que le canal de transmission est de qualité suffisante pour assurer qu'on finira par obtenir une trame dont la somme de contrôle est correcte.

2 Analyse des mesures

Les fonctions demandées pourront être écrites dans le langage `Python` (éventuellement à l'aide de sa bibliothèque `numpy`) ou dans le langage `Scilab`.
On exclut dans cette partie tout appel à une fonction de calcul approché d'intégrale ou de résolution approchée d'équations différentielles qui serait préprogrammée dans le langage choisi ou dans l'une de ses bibliothèques.

2.1 Traitement numérique

Q7. Rappeler le principe de la méthode des trapèzes permettant de calculer une valeur approchée de l'intégrale d'une fonction f sur un intervalle $[a, b]$. On ne demande pas ici de coder de fonction informatique permettant ce calcul.

Pour valider le fonctionnement de l'imprimante, on considère une suite de valeurs de mesures de l'intensité du courant (en ampères) du moteur de la tête d'impression, stockée dans un tableau `mesures`.
On s'intéresse alors à la valeur moyenne et à l'écart-type de cette distribution de valeurs.
Pour une fonction f définie (et continue) sur un segment $[0, t_{\text{final}}]$, la valeur moyenne de la fonction sur ce segment est définie par :

$$f_{\text{moy}} = \frac{1}{t_{\text{final}}} \int_0^{t_{\text{final}}} f(t)\mathrm{d}t.$$

On ne dispose ici des valeurs de l'intensité qu'en des instants prédéfinis : les mesures ont été effectuées toutes les 2 millisecondes.

Q8. Écrire une fonction `trapezes(L)` prenant en argument un tableau `L` de valeurs numériques, et renvoyant une valeur approchée, calculée par la méthode des trapèzes, de la valeur moyenne d'une grandeur variant durant un certain intervalle de temps $[0, t_{\text{final}}]$, en supposant que le tableau `L` contient les valeurs de cette grandeur mesurées toutes les 2 millisecondes.

On définit l'écart-type d'une fonction sur un segment $[0, t_{\text{final}}]$ par :

$$f_{\text{ec}} = \sqrt{\frac{1}{t_{\text{final}}} \int_0^{t_{\text{final}}} (f(t) - f_{\text{moy}})^2 \mathrm{d}t}.$$

Q9. Écrire une fonction `indicateurs(mesures)` prenant en argument un tableau `mesures` de flottants codant les valeurs de l'intensité mesurées toutes les 2 millisecondes sur un intervalle de temps $[0, t_{\text{final}}]$, et renvoyant la valeur de la moyenne et de l'écart-type des valeurs de ce tableau. On privilégiera l'utilisation de la fonction `trapezes`.

2.2 Validation des mesures

On sait que chaque moteur de l'imprimante, soumis à un signal d'entrée e, doit fournir un signal de sortie s lié au signal d'entrée par l'équation différentielle linéaire du premier ordre :

$$\frac{\mathrm{d}s}{\mathrm{d}t} = -\frac{k}{10}(s - e), \tag{1}$$

où k est un nombre strictement positif. Le bon fonctionnement d'un tel moteur est testé en vérifiant l'adéquation des valeurs mesurées avec des valeurs issues d'une résolution numérique de cette équation.

Q10. On considère le signal d'entrée $e(t) = \sin(\omega\, t)$ sur l'intervalle de temps $[0, 1]$ (en secondes), où ω est une constante numérique stockée dans une variable `omega`.
Écrire une fonction `simulation(...)` renvoyant un tableau de valeurs approchées, calculées à l'aide de la méthode d'Euler, d'une solution de l'équation (1) aux instants $[0, 0.002, 0.004, \ldots, 1]$ (en secondes). Le candidat sera amené à choisir un ou des arguments pour la fonction programmée, et commentera le choix effectué.
On rappelle que tout appel à une fonction de résolution approchée d'équations différentielles préprogrammée est exclu.

On suppose désormais qu'à l'instant $t = 0$, la valeur $s(0)$ du signal de sortie doit être nul. Le signal de sortie réel, acquis par le capteur, est stocké dans une variable `mesures` de type tableau, contenant la valeur du signal aux instants $[0, 0.002, 0.004, \ldots, 1]$ (en secondes). L'imprimante doit avoir un comportement conforme à la solution de l'équation (1) avec $s(0) = 0$. En raison de différences de fabrication, k peut prendre les valeurs 0.5, 1.1 ou 2, et seulement celles-ci. Le comportement du moteur est considéré valide si l'écart maximal entre valeurs observées et valeurs simulées est inférieur à un certain flottant positif `eps`, pour au moins une valeur de k parmi 0.5, 1.1 ou 2.

Q11. Écrire une fonction `validation(mesures,eps)` prenant en arguments un tableau `mesures` de valeurs observées du signal de sortie et un flottant positif `eps`, et renvoyant un booléen indiquant si le comportement du moteur peut être validé au seuil `eps`.

3 Bases de données

On suppose que les résultats des analyses précédentes ont été stockés dans une base de données, constituée de deux tables, dont on donne une représentation simplifiée dans l'annexe de la page 8.
Après son assemblage et avant les différents tests de validation, un numéro de série unique est attribué à chaque imprimante. À la fin des tests de chaque imprimante, les résultats d'analyse ainsi que le nom du fichier contenant l'ensemble des mesures réalisées sur l'imprimante sont stockés 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 le nom du modèle.

Q12. Écrire une requête `SQL` permettant d'obtenir les numéros de série des imprimantes ayant une valeur de `Imoy` comprise strictement entre 0.4 et 0.5.

Q13. Écrire une requête `SQL` permettant d'obtenir les numéros de série, les valeurs de `Imoy` et `Iec` ainsi que le modèle des imprimantes ayant été validées et dont la valeur de `Imoy` est comprise strictement entre 0.4 et 0.5.

Q14. Écrire une requête `SQL` permettant d'obtenir les modèles des imprimantes validées et pour chacun de ces modèles, le nombre d'imprimantes validées.

Q15. Écrire une requête `SQL` renvoyant le numéro de série et le nom du fichier de mesures des imprimantes qui n'ont pas été validées en sortie de production. Commenter très brièvement l'intérêt de cette information pour cette ligne de production.

4 Compression d'un fichier texte

Le fichier de résultat va être compressé sous la forme d'un fichier binaire. Pour cela, on code chaque caractère en une séquence de bits, de longueur variable en fonction du caractère. Le taux de compression est le rapport entre la taille en bits du fichier compressé et celle du fichier initial. Pour optimiser ce taux de compression, on réserve les séquences les plus courtes pour les caractères qui apparaissent le plus souvent dans le fichier, en laissant les séquences longues pour ceux qui sont les plus rares. Le codage du fichier est alors obtenu en juxtaposant les codes de chacun de ses caractères.

Cette partie est constituée de quatre sous-parties : on propose d'abord quelques codages explicites pour de petites chaînes de caractères afin de se familiariser avec la notion de codage par des séquences de longueur variable ; on écrit ensuite des fonctions permettant, à partir d'un texte à coder, d'extraire et d'organiser les informations concernant les caractères qui apparaissent et leurs fréquences ; on présente dans les troisième et quatrième sous-parties un algorithme de compression appelé « codage de Huffman », et un algorithme de décompression associé.

4.1 Exemples de codage

Q16. On choisit les codes suivants pour les quatre lettres `E`, `S`, `A` et `T` :

`E` : 1 `S` : 0 `A` : 10 `T` : 01.

Quels sont les codes des deux chaînes de caractères `EST` et `ASE` ? Quel inconvénient majeur possède ce codage ?

Q17. On choisit maintenant les codes suivants pour les six lettres `E`, `R`, `U`, `O` , `N` et `T` :

`E` : 11 `R` : 10 `U` : 000 `O` : 001 `N` : 010 `T` : 011.

Quel mot est codé par la séquence :

10110110000100101111?

Sachant qu'un caractère se code habituellement sur un octet, quel taux de compression obtient-on avec ce codage ? Que doit-on transmettre au destinataire pour qu'il puisse décoder le fichier binaire obtenu ? Commenter brièvement l'intérêt d'un tel codage.

Q18. Proposer une propriété du codage permettant d'éviter l'inconvénient de la question 16.

4.2 Tri des caractères selon leur fréquence

Q19. Écrire une fonction `caracteres_presents(donnees)` prenant en paramètre une chaîne de caractères `donnees` et retournant une liste des caractères présents dans cette chaîne, chaque caractère apparaissant dans le paramètre ne devant figurer qu'une seule fois dans la valeur de retour. Estimer la complexité temporelle de la fonction ainsi codée en fonction de la longueur de la chaîne `donnees`.

Q20. Modifier la fonction précédente pour qu'elle retourne deux listes `cars` et `freq` telles que :
- la liste `cars` est la liste des caractères apparaissant dans l'argument `donnees`, comme à la question précédente ;
- la liste `freq` est une liste de même longueur que la liste `cars` telle que, pour chaque position `i`, `freq[i]` est un entier donnant le nombre d'apparitions du caractère `cars[i]` dans la chaîne `donnees`.

On se propose maintenant d'ordonner les tableaux `cars` et `freq` selon la fréquence des caractères. On fournit en *annexe* 5.2 trois fonctions nommées `echange`, `aux` et `tri`. On propose dans les questions suivantes d'analyser ces fonctions.

Q21. Décrire le comportement de l'appel à `aux(cars,freq,2,7)` lorsque :

```
freq = [ 4 ,  6 ,  3 ,  4 ,  8 ,  1 ,  7 ,  4 ]
cars = [' ', 'a', 'c', 't', 'e', 'è', 's', 'r'].
```

Q22. Préciser l'action de l'instruction `aux(cars,freq,i,j)` sur les paramètres `cars` et `freq`. On justifiera la réponse en s'aidant d'un invariant de boucle, et on prouvera la terminaison.

Q23. Écrire, à l'aide d'une ou des fonctions données en annexe, une instruction ou une suite d'instructions permettant d'obtenir les tableaux `cars` et `freq` triés par ordre croissant de fréquences et vérifiant toujours la propriété de correspondance énoncée à la question 20. Donner la complexité en temps dans le pire des cas de cette ou ces instructions, en fonction de la longueur commune des deux tableaux `cars` et `freq`.
Existe-t-il un tri ayant une complexité plus faible ? Si oui, donner son nom et sa complexité dans le pire des cas.

4.3 Codage de Huffman.

La partie 4.2 permet d'obtenir deux tableaux `cars` et `freq` vérifiant les conditions de la question 20, et tels que, de plus, le tableau `freq` est trié par ordre croissant. On commence par stocker ces informations sous la forme d'une seule liste `cars_ponderes`, dont chaque élément est un tableau constitué d'un caractère et de sa fréquence dans le fichier, et qui est triée par ordre croissant des fréquences.
Par exemple, pour le texte `'abbaeccedeccedadaeefeee'`, cette variable est initialement :

```
cars_ponderes = [['f',1], ['b',2], ['d',3], ['a',4], ['c',4], ['e',9]]
```

Dans ce qui suit, on appelle « poids », les entiers figurant dans les tableaux contenus dans la variable `cars_ponderes`.

L'algorithme de Huffman consiste alors, jusqu'à ce que la liste `cars_ponderes` ne contienne qu'un élément, à sélectionner les deux tableaux $[s_1,p_1]$ et $[s_2,p_2]$ ayant les poids p_1 et p_2 les plus faibles. On calcule un nouveau tableau `[s,p]` dans lequel `s` est une chaîne obtenue en concaténant s_1 et s_2 et `p` un poids obtenu en additionnant p_1 et p_2. Ce nouveau tableau est inséré dans la liste `cars_ponderes` de manière à conserver la propriété de croissance des poids, tandis que les deux tableaux $[s_1,p_1]$ et $[s_2,p_2]$ sont supprimés. Par exemple, avec les valeurs figurant ci-dessus, la première étape consiste à remplacer `['f',1]` et `['b',2]` par `['fb',3]` (il y a ici égalité entre le poids du tableau à insérer et le poids du tableau `['d',3]` déjà présent dans `cars_ponderes` : l'insertion se fait à gauche dans ce cas) ; puis la seconde étape consiste à remplacer `['fb',3]` et `['d',3]` par `['fbd',6]`.

Parallèlement, on calcule la séquence de bits codant chaque caractère. Les séquences de bits seront représentées ici par des chaînes de caractères formées de `0` et de `1` et appelées des « codes ». Au départ, chaque caractère est associé à une chaîne vide. À chaque étape de l'algorithme, on fusionne deux chaînes de caractères s_1 et s_2 comme on l'a vu précédemment. On modifie alors les codes associés aux caractères présents dans s_1 et s_2 : on ajoute un `0` dans le code de chaque caractère de s_1, et un `1` dans le code de chaque caractère de s_2. Les ajouts de `0` et de `1` se font toujours par la gauche.

Voici un exemple de déroulement de l'algorithme (fig. 1) : à gauche figure le contenu de la variable `cars_ponderes` et à droite le tableau des codes des différents caractères. Chaque ligne correspond à une étape de l'algorithme. La première ligne indique les valeurs initiales. Lors de la première étape, on traite les tableaux `['f',1]` et `['b',2]` et on ajoute un `0` dans le code de `'f'` et un `1` dans celui de `'b'`. Ensuite, on traite les tableaux `['fb',3]` et `['d',3]` et on ajoute un `0` dans les codes de `'f'` et `'b'` (ajout par la gauche) et un `1` dans le code de `'d'`, etc.

Liste `cars_ponderes`	Codes des caractères `'a' , 'b' , 'c' , 'd' ,'e', 'f'`
`[['f',1], ['b',2], ['d',3], ['a',4], ['c',4], ['e',9]]`	`[ '' , '' , '' , '' , '' , '' ]`
`[['fb',3], ['d',3], ['a',4], ['c',4], ['e',9]]`	`[ '' , '1' , '' , '' , '' , '0' ]`
`[['a',4], ['c',4], ['fbd',6], ['e',9]]`	`[ '' , '01' , '' , '1' , '' , '00' ]`
`[['fbd',6], ['ac',8], ['e',9]]`	`[ '0' , '01' , '1' , '1' , '' , '00' ]`
`[['e',9], ['fbdac',14]]`	`['10' , '001' , '11' , '01' , '' , '000' ]`
`[['efbdac',23]]`	`['110','1001','111','101','0','1000']`

Figure 1. Calcul des codes des caractères du texte `'abbaeccedeccedadaeefeee'`.

Q24. Écrire une fonction `insertion(t,q,cars_ponderes)` prenant en arguments une liste `cars_ponderes` telle que décrite ci-dessus, une chaîne de caractères `t` et un entier `q`, et qui insère le tableau `[t,q]` comme élément de la liste `cars_ponderes` de telle sorte que celle-ci reste triée par ordre croissant des poids. La fonction renvoie la liste obtenue après insertion.

Par exemple, l'appel `insertion('fbd', 6, [['a',4], ['c',4], ['e',9]])` doit renvoyer la liste `[['a',4], ['c',4], ['fbd',6], ['e',9]]`.

Q25. Écrire une fonction `codage(cars,freq)` prenant en argument un tableau de caractères `cars` et le tableau des fréquences correspondantes `freq`, supposé trié en ordre croissant, et calculant le tableau des codes des caractères du tableau `cars` à l'aide de l'algorithme de Huffman. L'appel

```
codage(['f','b','d','a','c','e'], [1, 2, 3, 4, 4, 9])
```

devra renvoyer le tableau de codes figurant en bas à droite de la figure 1.

On supposera disposer d'une fonction `numero(x)` qui renvoie, pour chaque caractère `x` apparaissant dans le tableau `cars`, un entier compris entre 0 et $N-1$ où N est la longueur du tableau `cars`, de sorte que pour deux caractères `x` et `y` distincts, les valeurs renvoyées soient distinctes. On utilisera cette fonction pour déterminer la position, dans le tableau renvoyé, du code du caractère `x`. Dans l'exemple de la figure 1, la fonction `numero` vérifie :

`numero('a') = 0`, `numero('b') = 1`, ..., `numero('f') = 5`.

4.4 Décodage

Pour décoder un texte (variable `txt_bin` ci-dessous) encodé par l'algorithme précédent, on suppose disposer d'un tableau `cars` contenant les caractères du texte et d'un tableau `code`, contenant les codes de ces caractères, tel que rendu par la fonction `codage` de la question 25. Deux conditions sont supposées remplies par ces tableaux : la position d'un caractère dans le tableau `cars` est la même que celle de son code dans le tableau `code` ; et les éléments du tableau `code` sont rangés par ordre dit lexicographique, c'est-à-dire que pour deux codes distincts $a = a_1 \dots a_n$ et $b = b_1 \dots b_m$, a précède b dans le tableau `code` soit s'il existe un indice i inférieur à m et n tel que $a_1 \dots a_{i-1} = b_1 \dots b_{i-1}$ (condition vide lorsque $i = 1$) et $a_i < b_i$, soit si $m > n$ et $a_1 \dots a_n = b_1 \dots b_n$.

Q26. Ranger les codes `'110'`, `'1001'`, `'111'`, `'101'`, `'0'`, `'1000'` par ordre lexicographique.

En Python, les opérateurs de comparaison usuels permettent directement de comparer les codes (chaînes écrites uniquement à l'aide de 0 et de 1). Par exemple, l'expression `'01' < '11'` est évaluée à `True` tandis que l'expression `'001' < '00'` est évaluée à `False`.

Pour `txt_bin` une chaîne de `0` et de `1` obtenue par compression d'un texte par l'algorithme de Huffman, et `pos` un entier décrivant une position dans cette chaîne telle qu'en cette position commence le code d'un caractère du texte initial, les propriétés du codage de Huffman assurent l'existence d'un unique entier `m` tel que le code `code[m]` soit égal à une sous-chaîne de `txt_bin` commençant à la position `pos`. Par exemple, pour les données suivantes :

```
txt_bin = '0101000000110111011111011110'
code = ['000', '001', '010', '011', '10', '110', '1110', '1111']
cars = [ 'n' ,  'j' ,  'b' ,  ' ' , 'o' ,  '!' ,  'u'  ,  'r'  ]
```

pour `pos=0`, on reconnaît que les trois premiers caractères de `txt_bin` forment un code, et on obtient `m=2`, et on trouve dans la table `cars` le premier caractère du texte initial. On passe alors la variable `pos` à la valeur 3, et on identifie le code `'10'` (`m=4`), correspondant à la lettre `'o'`, etc.

Q27. Continuer les calculs de l'exemple ci-dessus, et préciser le texte qui a été compressé. On attend que figurent les valeurs successivement obtenues pour `pos` et `m`.

Q28. Écrire une fonction `decode_car(txt_bin,pos,code)` prenant les arguments décrits ci-dessus et renvoyant l'unique entier `m` tel qu'il existe une sous-chaîne de `txt_bin` commençant à la position `pos` et égale à la chaîne `code[m]`. On demande que le nombre de comparaisons entre des chaînes de caractères effectuées par cette fonction soit $O\big(\log(\texttt{len(code)})\big)$.

Q29. Écrire une fonction `decode_txt(txt_bin,code,cars)` permettant de décoder un texte `txt_bin` compressé à l'aide des informations de compression stockées dans les variables `code` et `cars` décrites plus haut.

5 Annexe

5.1 Base de données

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	modele
20	230-588ZX2547	2012-04-22 15-52-12	JETDESK-1050
21	230-588ZX2549	2012-04-22 15-53-24	JETDESK-3050
⋮	⋮	⋮	⋮

5.2 Fonctions pour les questions 21 à 23

```
# Lors des appels aux fonctions définies dans ce fichier,
# les paramètres freq et cars sont des tableaux de même taille
# dont les éléments sont respectivement des entiers et des caractères,
# comme obtenus à l'aide de la question 20.
# Les paramètres i et j vérifient 0 ⩽ i ⩽ j ⩽ len(cars) − 1.

def echange(cars,freq,i,j):
    freq[i], freq[j] = freq[j], freq[i]
    cars[i], cars[j] = cars[j], cars[i]

def aux(cars,freq,i,j):
    k = j
    m = j
    while k > i:
        k = k − 1
        if freq[k] > freq[j]:
            m = m − 1
            echange(cars,freq,k,m)
    echange(cars,freq,m,j)
    return m

def tri(cars,freq,i,j):
    if j >= i + 1:
        m = aux(cars,freq,i,j)
        tri(cars,freq,i,m−1)
        tri(cars,freq,m+1,j)
```