V - Étude de capteur

Un signal transmis par un capteur peut être perturbé par toutes sortes de facteurs pouvant produire des erreurs dans les données : masses métalliques, température, imperfections du matériel... En pratique, il est donc nécessaire de pouvoir détecter ces erreurs et, dans la mesure du possible, les corriger.

V.1 Bit de parité

une technique simple et très répandue pour s'assurer qu'une donnée binaire sera lue correctement par son recepteur est de lui adjoindre un bit de parité, c'est-à-dire la somme binaire des bits de données. Par exemples : 110111_2 a comme bit de parité 1 car il possède un nombre impaire de 1 et 110011_2 a comme bit de parité 0 car il possède un nombre paire de 1

Q46. Donner les bits de parité associées aux représentations binaires des entiers $5, 16, 37$.

Q47. Ecrire une fonction `parite(bits)` prenant pour argument une liste `bits` contituée d'entiers valant 0 ou 1 et retournant l'entier 0 ou 1 correspondant à son bit de parité.

Les techniques de vérification les plus simples consistent à découyper la donnée en blocs et àjoindre un bit de parité à chaque bloc. Par exemple, certains protocoles transmettent 7 bits de données pour un bit de parité.

Q48. Donner un exemple d'erreur n'étant pas détecter par cette technique.

V.2 Code de Hamming :

Le code de Hamming (7,4) est un exemple d'utilisation des bits de parité pour détecter et corriger des erreurs. Il consiste à adjoindre 3 bits de parité à 4 bits de données, soit un message d'une longueur totale de 7 bits. Ces trois bits sont définis aubsu : si la donnée s'écrit (d_1, d_2, d_3, d_4) avec $d_i = 0$ ou 1, alors :

- p_1 est le bit de parité du triplet (d_1, d_2, d_4) ;
- p_2 est le bit de parité du triplet (d_1, d_3, d_4) ;
- p_3 est le bit de parité du triplet (d_2, d_3, d_4).

Le message encodé, que l'on transmet, s'écrit alors comme suit : $(p_1, p_2, d_1, p_3, d_2, d_3, d_4)$.

Q49. Ecrire une fonction `encode_hamming(donnes)` prenant pour argument une liste `donnes` contituée d'entiers valant 0 ou 1 et retournant une lsite de bits contenant le message encodé. On pourra appeler la fonction `partie(bits)` précédemment définie.

Le contrôle après réception d'un message ainsi encodé est relativement simple. La technique proposée par Hamming est de calculer les bits de contrôle suivants, notés (c_1, c_2, c_3), à partir d'un message complet (données et bits supplémentaires), noté $(m_1, m_2, \ldots, m_7)$:

- c_1 est le bit de parité du triplet (m_4, m_5, m_6, m_7) ;
- c_2 est le bit de parité du triplet (m_2, m_3, m_6, m_7) ;
- c_3 est le bit de parité du triplet (m_1, m_3, m_5, m_7) ;

On montre que si le message est bien encodé selon les règles précédentes, et n'a pas été altéré, alors les trois bits de contrôle doivent être à 0. Si ce n'est pas le cas, alors il y a une erreur ; l'intérêt de la technique de Hamming est que dans le cas particulier où l'erreur est unique, le mot de contrôle donne la représentation binaire de la position de cette erreur en numérotant les indices à partir de 1. Par exemple, si $(c_1, c_2, c_3) = (0, 1, 1)$, alors l'erreur porte sur le troisième bit du message (m_3). Il suffit d'inverser ce bit (le mettre à 1 s'il est à 0, et inversement) pour corriger l'erreur.

La donnée décodée est alors constituée des quatres bits (m_3, m_5, m_6, m_7) du message complet $(m_1, m_2, \ldots, m_7)$.

Q50. On reçoit la donnée (0110111). Vérifier qu'il y a bien une erreur et déterminer la donnée décodée.

Q51. Ecrire une fonction `decode_hamming(message)` prenant pour argument une liste `message` contituée de 7 bits et retournant une lsite de 4 bits contenant la donnée décodée. En cas d'erreur, on affichera à l'écran un avertissement indiquant la position du bit affecté et on effectuera la correstion. On supposera que s'il y a une erreur, elle est unique.

VI Production des amortisseurs

Pour améliorer la qualité des amortisseurs mises en vente, différents tests de fin de chaîne ont été mis en place pour valider l'assemblage du produit final. Différentes mesures sont alors exécutées. Ces mesures sont envoyées comme une suite de caractères ASCII vers un ordinateur. Cet ordinateur va effectuer des analyses pour valider le fonctionnement de l'amortisseur. En particulier, on mesure le temps de réponse à un échelon ($T_{r_{5\%}}$) en ms, puis on calcule la valeur moyenne T_{moy} du temps sur la durée d'acquisition ainsi que l'écart-type T_{ec}. Ainsi on peut avoir l'intervalle de confiance à 95% qui est $[T_{moy} - 1,96T_{ec}; T_{moy} + 1,96T_{ec}]$ (c'est-à-dire que la probabilité que le temps de réponse de l'amortisseur est dans cet intervalle est de 95%).
L'ensemble des mesures et des analyses est sauvegardé dans un fichier texte.
Cette sauvegarde s'effectue dans une base de données.
Après son assemblage et avant les différents tests de validation, un numéro de série unique est attribué à chaque amortisseur. À la fin des tests, les résultats d'analyse ainsi que le fichier contenant l'ensemble des mesures réalisées sont rangés dans la table $testfin$.
Lorsqu'un amortisseur satisfait les critères de validation, il est enregistré dans la table *production* avec son numéro de série, la date et l'heure de sortie de production ainsi que son modèle. Une représentation simplifiée des deux tables de la base de données que l'on souhaite utiliser est donnée ci-dessous :

$testfin$				
nSerie	dateTest	T_{moy}	T_{ec}	fichierMes
588ZX2547	2017-02-07 :17-25-45	28	3	mes31025.csv
588ZX2548	2017-02-07 :17-26-57	30	6	mes41026.csv
⋮	⋮	⋮	⋮	⋮

production			
Num	nSerie	dateProd	Modele
20	588ZX2547	2017-02-07 :18-29-12	B4-P205
21	588ZX2549	2017-02-07 :18-31-53	B6-Rclio1
⋮	⋮	⋮	⋮

Les attributs T_{moy} et T_{ec} sont des flottants. L'attribut Num est un entier. Les autres attributs sont de type chaîne de caractères.

Q52. Rédiger une requête SQL permettant d'obtenir les numéros de série des amortisseurs répondant aux critères d'exigences des constructeurs (**tableau 1**) à 95%.

Q53. Rédiger une requête SQL permettant d'obtenir les numéros de série, la valeur de l'écart-type T_{ec} et le fichier de mesures des amortisseurs ayant une valeur T_{ec} strictement inférieure à la moyenne de la colonne T_{ec}.

Q54. Extraire à partir de la table $testfin$ les numéros de série et le fichier de mesures des amortisseurs qui n'ont pas été validées en sortie de production.

FIN

Correction

Q46. $5 = 101_2$, le bit de parité vaut 0
$16 = 10000_2$, le bit de parité vaut 1
$37 = 100101_2$, le bit de parité vaut 1

Q47.

```
def parite(bits):
    s=0
    for i in bits:
        s=s+i
    return s%2
```

Q48. Si un nombre pair d'erreurs est commis, on ne peut pas les détecter
Par exemple, pour 37 transmis comme 101001_2 n'est pas détectable.

Q49.

```
def encode_hamming(donnee):
    d1,d2,d3,d4=donnee
    p1=parite([d1,d2,d4])
    p2=parite([d1,d3,d4])
    p3=parite([d2,d3,d4])
    return [p1,p2,d1,p3,d2,d3,d4]
```

Q50. Pour 0110111_2
La parité de 0111 est 1
La parité de 1111 est 0
La parité de 0111 est 1
Donc on a une erreur en position 4+1=5
Le message est 0110011_2 et la donnée : 1011_2

Q51.

```
def decode_hamming(message):
    m1,m2,m3,m4,m5,m6,m7=message
    c1=parite([m4,m5,m6,m7])
    c2=parite([m2,m3,m6,m7])
    c3=parite([m1,m3,m5,m7])
    n=4*c1+2*c2+c3
    if n!=0:
        print("Erreur transmise en ",n)
        message[n-1]=1-message[n-1]
        m1,m2,m3,m4,m5,m6,m7=message
    return [m3,m5,m6,m7]
```

Q52. select nSerie from testfin where Tmoy-1.96*Tec<35 and Tmoy+1.96*Tec>35

Q53. select nSerie,Tec,fichierMes from testfin where Tec < (select avg(Tec) from testfin)

Q54. select nSerie, fichieerMes from testfin where nSerie not in (select nSerie from production)