Informatique en CPGE (2015-2016)
TD 9 : transmission fiable de données

Exercice 1 : clé de contrôle et somme de contrôle ("cheksum" en anglais)

Une clé de contrôle ou une somme de contrôle est un nombre qui est ajouté à un message pour permettre au récepteur de vérifier si le message reçu est sans erreur. Mais cette méthode ne permet pas de corriger une éventuelle erreur.

Exemple 1

```
def cle(c):
    n=int(c)
    k=97-n%97
    return k

def verif(insee):
    c=insee[:13]
    print(c)
    k=insee[13:]
    print(k)
    if int(k)==cle(c):
        return True
    else:
        return False

print(verif('296040632753676'))
print(verif('296040632756376'))
print(verif('296040632735676'))
print(verif('296040632736576'))
print(verif('296040632765376'))
print(verif('296040632763576'))
```

Exemple 2

```
# le seizième chiffre est la clé
def verif(carte):
    somme=0
    for i in range(len(carte)):
        p=int(carte[i])
        if i%2==0:
            n=2*p
            if n>9:
                n=n-9
            somme+=n
        else:
            somme+=p
        print(somme)
    if somme%10==0:
        return True
    else:
        return False
```

Exemple 3

```
# 65 en base 2 : 1000001
#donc le bit de parité vaut 0
# 7 en base 2 : 111
#donc le bit de parité vaut 1
# 8 en base 2 : 1000
#donc le bit de parité vaut 1
# 15 en base 2 : 1111
#donc le bit de parité vaut 0

# une erreur sur un bit est détectable
# s'il y a deux erreurs, la correction rajoute
# une troisième erreur sur un autre bit

def parite(liste):
    return sum(liste)%2

# exemple pour un caractère:
liste=[int(i) for i in bin(ord('A'))[2:]]
print(parite(liste))

# ou bien pour un entier
liste=[int(i) for i in bin(8)[2:]]
print(parite(liste))
```

Exercice 2

1.

```
def taille(nomFichier):
    fic=open(nomFchier,'r')
    test=True
    cpt=0
    while test==True:
        u=fic.read(1)
        if u!=str():
            cpt+=1
        else:
            test=False
    fic.close()
    return cpt
```

2.

```
def calcul(nomFichier):
    f=open(nomFichier,"r")
    test=True
    cpt=0
    somme=0
    somme2=0
    while test ==True:
        u=f.read(1)
        if u!=str():
```

```
            cpt+=1
            somme+=ord(u)
            somme2=(somme2+ord(u)*(1+cpt%7))%127
        else:
            test=False
    f.close()
    return cpt, somme, somme2
```

3.

```
def compare(nomFichier1,nomFichier2):
    if calcul(nomFichier1)==calcul(nomFichier2):
        return True
    else:
        return False
```

Exercice 3 : code correcteur

1.

```
def parite(liste):
    return sum(liste)%2

# par exemple:
liste=[int(i) for i in bin(ord('A'))[2:]]
print(parite(liste))
# ou bien
liste=[int(i) for i in bin(8)[2:]]
print(parite(liste))

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]
```

2.

```
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])
    test=c1*c2*c3
    if test!=0:
        p=c1*2*2+c2*2+c3
        print("erreur sur le bit",p)
        message[p-1]=1-message[p-1]
    return [message[2],message[4],message[5],message[6]]
```

3. La donnée est 1011 : p1=0, p2=1, p3=0 donc le message codé est 0110011 ;

s'il y a une erreur sur chacun des deux premiers bits, cela donne 1010011 ; et au décodage, on obtient 1000011 soit le message décodé 0011.

La correction ajoute une troisième erreur sur un autre bit.

Vérification :

```
d=[1,0,1,1]
dc=encode_hamming(d)
print(dc)
er=[1-dc[0]]+[1-dc[1]]+dc[2:]
print(er)
erdec=decode_hamming(er)
print(erdec)
```

4. On ajoute un bit de parité p4 pour m1m2m3m4m5m6m7 ; s'il y a une erreur p4=1 et s'il y a 2 erreurs p4=0.