Codes détecteur et correcteurs (exemples simples)

Contrôle de parité : détection d'une erreur

L'idée la plus simple consiste à rajouter un **bit de parité**. Si l'on veut transmettre la suite de bits `1011011`, on rajoute un bit de contrôle (par exemple à gauche) de manière à ce que la somme modulo 2 de tous les bits soit nulle. On transmettra donc ici `11011011`. Si la suite reçue présente une erreur, celle-ci sera détectée car la somme modulo 2 des bits ne sera plus nulle. Par contre, on ne peut pas corriger l'erreur car on ne sait pas quel bit est erroné et on ne peut pas détecter deux erreurs.
On souhaite transmettre une chaîne de caractères. Celle-ci doit déjà être transformée en suite de bits. Nous allons pour cela utiliser le code `ASCII` qui fait correspondre à chaque caractère un entier entre 0 et 255.

```
>>> ord('a')
97
>>> chr(65)
'A'
```

Si l'on se restreint à l'alphabet non accentué, on obtient un code ASCII plus petit que 127 ce qui nous permet de coder le caractère sur 7 bits. Il reste alors à ajouter un bit de parité (on obtient donc un octet) pour détecter les erreurs sur chaque caractère.

• Écrire une fonction `entier_vers_binaire(n, N)` qui renvoie sous forme de liste l'écriture en binaire de l'entier n sur N bits (le bit de poids fort est au début de la liste) puis écrire une fonction `binaire_vers_entier(L)` qui réalise l'opération inverse.

On obtient alors les fonctions suivantes

```
def caractere_vers_binaire(c, N):
    return entier_vers_binaire(ord(c), N)
def binaire_vers_caractere(L):
    return chr(binaire_vers_entier(L))
```

• Écrire une fonction `chaine_vers_message1(chaine)` qui transforme une chaîne de caractères en la liste des suites de 7 bits codant chaque caractère de la chaîne puis une fonction `message_vers_chaine1(L)` réalisant l'opération inverse.

• Écrire une fonction `somme_mod_2(L)` qui calcule la somme modulo 2 des éléments de L et en déduire une fonction `ajoute_bit_controle(L)` qui ajoute *en place* en tête de L le bit de contrôle.

• En déduire les fonctions `code1(message)` et `decode1(message)` qui ajoutent ou enlèvent (*en place*) le bit de contrôle sur chaque suite de bits constituant L.

• On rappelle que la fonction `randint(a, b)` du module `random` retourne un entier pseudo-aléatoire de $[\![a, b]\!]$. Écrire une fonction `transmission(L, N)` qui, avec une probabilité $\frac{1}{N}$, modifie *en place* un bit de L (un seul).
On utilisera la fonction

```
def change(b):
    if b == 0:
        return 1
    return 0
```

On en déduit la fonction

```
def transmission_message1(message, N):
    for x in message:
        transmission(x, N)
```

• Écrire une fonction `detection(message)` qui affiche la liste des indices des suites de bits du message qui ont subi une erreur.

• Tester avec le code suivant

```
def simulation1(chaine, N):
    print('a transmettre : ' + chaine)
    message = chaine_vers_message1(chaine)
    code1(message)
    transmission_message1(message, N)
    detection(message)
    decode1(message)
    print('recue : ' + message_vers_chaine1(message))
```

Autre somme de contrôle : ISBN

Chaque livre porte pour l'identifier un code à barres et un code ISBN.
L'International Standard Book Number (ISBN) est un numéro international qui permet d'identifier, de manière unique, chaque livre publié. Le numéro ISBN-10 se compose de trois segments de longueur variable et d'un segment de longueur fixe. La longueur totale de l'ISBN comprend dix chiffres (le 1er janvier 2007, la longueur a été étendue à 13 chiffres en ajoutant un groupe initial de 3 chiffres). Si les quatre segments d'un ancien code ISBN à 10 chiffres sont notés S1 - S2 - S3 - S4 :
— S1 identifie un groupe de codes pour un pays, une zone géographique ou une zone de langue.
— S2 identifie l'éditeur de la publication.
— S3 correspond au numéro d'ordre de l'ouvrage chez l'éditeur.

— S4 est un chiffre-clé calculé à partir des chiffres précédents et qui permet de vérifier qu'il n'y a pas d'erreurs. Outre les chiffres de 0 à 9, cette clé de contrôle peut prendre la valeur X, qui représente le nombre 10.
Exemple l'ISBN 2 - 10 - 004724 - 8

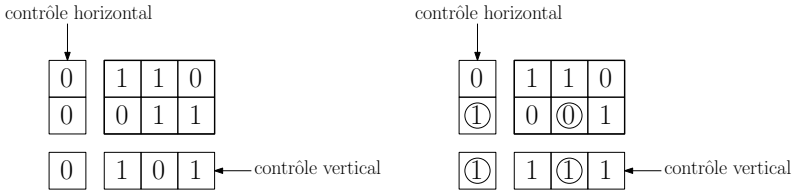
Pour calculer S4, on attribue une pondération à chaque position (de 10 à 2 en allant en sens décroissant) et on fait la somme des produits ainsi obtenus. On conserve le reste de la division euclidienne de ce nombre par 11. La clé s'obtient en retranchant ce nombre à 11. Si se nombre vaut dix, on utilise la lettre X.
En d'autres termes, si on note a_i le i^e chiffre, on doit avoir

$$\sum_{k=1}^{10} k a_k = 0 \ [11]$$

1. Écrire une fonction `ISBN(code)` prenant en argument les 9 premiers chiffres du code `ISBN`, et renvoyant le code complet sous sa forme normalisée.
2. Écrire une fonction `verif_ISBN(code)` prenant en argument un code `ISBN` complet (pour simplifier, sous la forme d'une chaîne de caractères sans tirets) et renvoyant `True` ou `False` selon que le dernier chiffre est cohérent.
Le nombre 11 étant premier, montrer qu'une erreur sur un chiffre ou un échange de deux chiffres sera automatiquement détecté.
Ce type de détection est appelé somme de contrôle (`checksum`), cas particulier de contrôle par redondance, qui permet la **détection** (mais pas la **correction**) des erreurs.

Contrôle de parité croisé : correction d'une erreur

On souhaite maintenant **corriger** une erreur. On utilise pour cela le contrôle de parité croisé qui consiste à ordonner les suites de bits en tableau et à ajouter un bit de contrôle par ligne et par colonne. S'il y a une seule erreur, celle-ci est facilement localisée donc corrigée. Par exemple, pour 3 paquets de 3 bits d'information (9 bits au total) on construit 7 bits de contrôles.



Pour coder nos messages ASCII, on a une information sur 7 bits, on va donc coder un message de 7 caractères en une matrice 8×8 avec 15 bits de contrôle.

1) Écrire les fonctions `code2(message)` et `decode2(message)` où message est une chaîne de 7 caractères `ASCII`.

2) Écrire la fonction `transmission_message2(message, N=8)` qui, avec une probabilité $\frac{1}{N}$, modifie *en place* un bit du message.

3) Écrire la fonction `correction(message)`.

4) Tester avec le code suivant

```
from copy import deepcopy

def simulation2(chaine, N):
    print('a transmettre : ' + chaine)
    message = chaine_vers_message1(chaine)
    code2(message)
    transmission_message2(message, N)
    mc = deepcopy(message)
    decode2(mc)
    print('transmise : ' + message_vers_chaine1(mc))
    correction(message)
    decode2(message)
    print('corrigee : ' + message_vers_chaine1(message))
```