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

Chaque personne en France dispose d'un numéro INSEE qui est un identifiant unique composé de 13 chiffres et d'une clé de contrôle à deux chiffres. Ce numéro figure par exemple sur les cartes de sécurité sociale (carte vitale). Si n est le nombre formé par les treize premiers chiffres et r le reste de la division euclidienne de n par 97, alors la clé de contrôle est le nombre $k = 97 - r$.

1. Ecrire une fonction qui prend en argument une chaîne de caractères représentant les treize chiffres, calcule et renvoie la clé.
2. Ecrire une fonction qui prend en argument une chaîne de caractères représentant les quinze chiffres (avec la clé), et vérifie si la clé correspond bien aux treize premiers chiffres. (Si ce n'est pas le cas, c'est qu'il y a une erreur.)
3. Les dix premiers chiffres du numéro 2960406327536 sont exacts, les trois derniers chiffres, 5, 3 et 6, sont peut-être dans un ordre différent ; sachant que la clé est 76, trouver le bon numéro INSEE.

Le principe est le même pour la clé sur un relevé d'identité bancaire sauf que le nombre formé par les 21 chiffres est multiplié par 3 avant la division par 97. S'il y a une lettre, elle est remplacée par un chiffre de 1 à 9 pour les lettres de A à I ou de J à R et de 2 à 9 pour les lettres de S à Z.

Exemple 2

Clé de Luhn : carte bancaire à 16 chiffres. On prend les 15 premiers chiffres et on remplace les chiffres d'ordre impairs (le premier, le troisième, ...) par leur double ou la somme des chiffres de leur double s'il est supérieur ou égal à 10 (cela revient à soustraire 9 ou à prendre le reste modulo 9). La somme des seize chiffres doit être divisible par dix, le seizième chiffre étant la clé.

Ecrire une fonction qui prend en argument une chaîne de caractères représentant les seize chiffres et qui renvoie False si une erreur est détectée, et True sinon.

Exemple 3

Le "bit de parité" : le principe est d'adjoindre à une donnée binaire un bit égal à :
soit 0 si la donnée contient un nombre pair de 1 (donc si ses bits sont de somme paire) ;
soit 1 si la donnée contient un nombre impair de 1 (donc, si ses bits sont de somme impaire).

Par exemple, si nous transmettons sur sept bits un entier ou un caractère codé avec le code ASCII, nous pouvons ajouté devant un bit de parité.

1. Sachant que le caractère 'A' est codé par 65, que vaut le bit de parité ?
2. Quels sont les bits de parité associés aux représentations binaires des entiers 7, 8 et 15 ?
3. Quels types d'erreurs permet de détecter cette méthode ?
4. Ecrire une fonction **parite** prenant pour argument une liste constituée d'entiers valant 0 ou 1 et retournant l'entier 0 ou 1 correspondant à son bit de parité.

Exercice 2

Afin de comparer deux fichiers (pour déterminer par exemple s'ils sont identiques) nous allons comparer les tailles de ces fichiers, c'est-à-dire le nombre d'octets composant chaque fichier. Mais si les deux tailles sont égales, il n'est pas possible d'en déduire que les fichiers sont identiques.

Nous allons donc compléter cette comparaison par deux sommes : la somme des valeurs décimales des octets et la somme modulo 127 des valeurs décimales des octets où pour cette deuxième somme, la valeur décimale de chaque octet est pondérée par son rang dans le fichier modulo 7.

Chaque octet d'un fichier peut être interprété comme le codage ASCII d'un caractère. Il suffit donc d'ouvrir le fichier en lecture, de lire caractère par caractère et d'utiliser la fonction **ord** qui renvoie la valeur décimale.

1. Ecrire une fonction **taille** qui prend en argument le nom d'un fichier avec son extension sous forme d'une chaîne de caractères et qui renvoie la taille de ce fichier.
2. Ecrire une fonction qui prend en argument le nom d'un fichier avec son extension sous forme d'une chaîne de caractères et qui renvoie la taille de ce fichier ainsi que les deux sommes décrites ci-dessus.
3. Ecrire une fonction **compare** qui prend en argument deux chaînes de caractères représentant les noms de deux fichiers avec leur extension et qui renvoie True ou False suivant que les tailles et les deux sommes sont identiques ou pas.

Exercice 3 : code correcteur

Extrait du sujet Banque PT 2015

Le signal transmis par une liaison RFID 13,56 MHz peut être perturbé par toutes sortes de facteurs pouvant provoquer des erreurs dans les données : autres signaux électromagnétiques, masses métalliques, imperfections du matériel électronique... En pratique, il est donc indispensable de pouvoir détecter ces erreurs et, dans la mesure du possible, les corriger sans que cela ne nécessite une nouvelle transmission ; l'objet de cette partie est de mettre en place quelques algorithmes dans ce but.

Le code de Hamming est un exemple d'utilisation des bits de parité pour détecter et corriger des erreurs. Nous nous intéressons ici au code dit (7,4), ainsi appelé car il consiste à joindre trois bits de parité à quatre bits de données, ce qui donne un message d'une longueur totale de sept bits. Ces trois bits de parité sont définis ainsi :

si la donnée s'écrit (d1, d2, d3, d4) avec di = 0 ou l, alors :
p1 est le bit de parité du triplet (d1, d2, d4),
p2 est le bit de parité du triplet (d1, d3, d4),
p3 est le bit de parité du triplet (d2, d3, d4).
Le message encodé, que l'on transmet, s'écrit alors comme suit : (p1, p2, d1, p3, d2, d3, d4).

1. Ecrire une fonction **encode_hamming (donnee)** prenant pour argument une liste **donnee** de quatre bits (représentés par des entiers valant 0 ou 1) et retournant une liste de bits contenant le message encodé. On pourra appeler la fonction **parite** définie dans l'exercice précédent.

   Le contrôle après réception d'un message ainsi encodé est relativement simple. On pourrait naturellement recalculer les trois bits de parité de la donnée et les comparer aux valeurs transmises, mais la technique proposée par Hamming est de calculer les trois bits de contrôle suivants, notés (c1, c2, c3), à partir du message complet (données et bits supplémentaires), noté (m1,..., m7) :

   c1 est le bit de parité de l'ensemble (m4, m5, m6, m7),

   c2 est le bit de parité de l'ensemble (m2, m3, m6, m7),

   c3 est le bit de parité de l'ensemble (m1, m3, m5, m7).

   On montre que si le message a bien été 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 eu 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 à partir de 1. Par exemple, si (c1, c2, c3) : (0,1,1), alors l'erreur porte sur le troisième bit du message. Il suffit ainsi d'inverser ce bit (le mettre à 1 s'il est à 0, et inversement) pour corriger I'erreur. La donnée décodée est alors constituée des quatre bits (d1, d2, d3, d4) qui se trouvent respectivement en positions 3, 5, 6 et 7 (toujours en numérotant à partir de 1) conformément à la description de l'encodage donnée ci-dessus.

2. Ecrire une fonction **decode_hamming(message)** prenant pour argument une liste de sept bits et retournant une liste de quatre 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 correction. On supposera dans cette question que s'il y a une erreur, alors elle est unique.
3. Déterminer le codage de Hamming de la donnée 1011, puis la donnée décodée par l'algorithme dans l'hypothèse où les deux premiers bits du message codé ont été incorrectement transmis. Quel a été l'effet de la "correction" sur la donnée dans ce cas ?
4. Sans coder, proposer un moyen simple de différencier une double erreur d'une erreur unique au moyen d'un bit de parité supplémentaire et expliquer comment cela permet d'éviter le problème mis en évidence à la question précédente. On s'appuiera sur les techniques introduites dans cette partie. On ne demande pas d'essayer de corriger la double erreur.