Protocoles réseaux
TD n° 1 : Codes détecteurs et correcteurs

Un **code** est une fonction C qui a tout mot de m bits associe un mot de n bits, $n > m$. Souvent, mais pas toujours, le code consiste à rajouter r bits au mots m et à copier les autres tels quels. Dans tous les cas, on notera $r = n - m$ la **redondance** du code. Pour qu'un code soit utile, il faut que deux mots différents soient toujours codés différement. Le **dictionnaire** du code est l'ensemble des 2^m mots de n bits codés.

Lors du décodage,
- Si le mot y de n bits reçu est dans le dictionnaire, alors on décode avec l'unique x à m bits tel que $C(x) = y$;
- Sinon, le code a **détecté** une erreur.

Si le mot envoyé est y et le mot reçu est y', le nombre d'**erreurs sur un bit** qui ont lieu est le nombre de positions où les bits de y et y' différent. Par exemple, 10110 et 00111 diffèrent en 2 positions (deux erreurs sur un bit).

Un code **détecte** k **erreurs** si, pour tout y' reçu avec au moins une erreur et au plus k par-rapport au y envoyé, le code signale une erreur.

Exercice 1 : bit de parité

Les anciens modems RTC utilisaient un système de contrôle d'erreurs basé sur le bit de parité : à la fin de chaque octet, on rajoute un 9e bit de sorte que le nombre de 1 dans l'octet soit pair.

1. L'utiliser pour coder 01100100 puis 0111010001111010
2. Donner l'algorithme de détection d'erreur.
3. Combien d'erreurs sur un bit ce code détecte-t-il ? Donner un exemple d'erreur non détectée.

Un code **corrige** k **erreurs** si, pour tout mot y' reçu avec au moins une erreur et au plus k par-rapport au mot y envoyé, y est le seul mot du dictionnaire tel que l'on peut obtenir y' en changeant au plus k bits dans un mot du dictionnaire. y' et k étant connus, le décodeur peut retrouver l'unique y et le produire, corrigeant ainsi l'erreur.

Exercice 2 : détection versus correction

1. Construire un code qui détecte 1 erreur. L'étendre pour détecter k erreurs (k donné).
2. Expliquez pourquoi tout code qui corrige k erreurs permet de détecter au moins k erreurs.
3. Montrez qu'un code qui détecte une erreur peut ne pas être capable de corriger une erreur.
4. On considère le code où chaque bit est répété $2k + 1$ fois. Expliquez comment détecter et corriger les erreurs avec ce code. Combien d'erreurs ce code peut-il détecter ? Combien d'erreurs peut-il corriger ?

Exercice 3 : matrice de parité

Une variante du bit de parité est la *matrice de parité*. On écrit 8 octets de données sur une matrice 8x8, puis on rajoute une 9e ligne et une 9ème colonne. Le bit de la ie colonne (resp. ligne) est choisi pour que le nombre de 1 dans cette colonne (resp. ligne) soit pair.

1. Codez la matrice ci-contre
2. Que vaut le bit en position (9,9) ?
3. Justifiez que ce code corrige une erreur.
4. Donnez un exemple de deux erreurs non corrigeable.
5. Combien d'erreurs ce code détecte-t-il ?
6. Donnez un exemple d'erreurs non détectées.
7. Expliquez comment étendre ce code à un message de taille $n \times n$. Comparez l'efficacité de ce code par rapport à celui de l'exercice 2.

1	0	0	0	1	1	0	1	
0	1	1	1	0	0	1	0	
1	0	0	0	0	0	0	1	
0	1	1	0	1	1	0	1	
0	1	0	0	0	0	0	0	
1	0	0	0	1	1	0	1	
0	1	0	1	0	1	0	0	
1	1	0	0	0	0	0	1	

Exercice 4 : Code de contrôle de IPv4

Dans la RFC 791, datant de 1981 et qui sert toujours de base pour la définition des entêtes de paquet IPv4, on trouve le texte suivant :

> `The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.`

Le but de cet exercice est d'analyser ce protocole de détection d'erreur, très largement utilisé actuellement. Pour comprendre ce charabia il faut savoir

- que `checksum` se traduit par *somme de contrôle*
- qu'une entête IPv4 est un ensemble de mots de 16 bits, l'un de ces mots étant le `checksum`
- que `one's complement` se traduit par *complément à 1*, et revient à inverser tous les bits d'un mot (opérateur NOT)
- et le `one's complement sum` de a et b est : si $a + b < 2^{16}$ alors $a + b$ sinon $a + b + 1 - 2^{16}$.

1. Calculez le checksum quand les mots sur 16 bits sont `0fa1 0070 fab1 0667 0000` (le dernier `0000` étant pour le checksum).

2. Maintenant, si on remplace le 0000 par la valeur du `checksum` que l'on vient de calculer, et que l'on refait le calcul, qu'obtient-on ?

3. En déduire l'algorithme de vérification.

4. Que se passe-t-il si une erreur pendant la transmission modifie un seul des mots 16 bits du message ?

5. Montrez que le code ne peut pas détecter un échange entre deux mots du message. Déduisez-en un exemple d'erreur non détectée en modifiant seulement deux bits dans le message.

6. La RFC se poursuit par

> `This is a simple to compute checksum and experimental evidence indicates it is adequate, but it is provisional and may be replaced by a CRC procedure, depending on further experience.`

Discutez.