# Protocoles réseaux
# TD n° 2 : Codes correcteurs
## (Correction)

## Code de Hamming

Attention, pour Hamming on numérote les bits d'un mot en partant de 1. Coder un mot $A$ de $m$ bits se fait en deux étapes :

D'abord, on insère un blanc (en décalant les autres bits vers la droite) à toutes les positions de bits dont le numéro est une puissance de 2. Ainsi :

A=011101000111101 devient : __0_111_0100011_1101.

Puis on remplit le blanc en position $2^i$ comme bits de parité de certain bits du mot. Plus précisément, le bit numero $2^i$ code alors la parité des bits de numéro $j$ tels que $j$ modulo $2^{i+1}$ est supérieur ou égal à $2^i$. Ainsi 1 contrôle ceux de numéro 1,3,5,7... ; 2 contrôle 2,3,6,7,10,11... ; 4 contrôle 4,5,6,7,12 etc etc. Visuellement, en gras les bits contôlés par le bit 1 : __0_111_0100011_1101 par 2 : __0_111_0100011_1101 par 4 : __0_111_0100011_1101 par 8 : __0_111_0100011_1101 et enfin par 16 : __0_111_0100011_1101.

On remplace donc chaque blanc par le bit de parité des bits qu'il contrôle, et c'est ce mot qui est transmis.

**Questions :**

1. Transmettez (codez) 0110010 puis 011101000111101 puis 001100001101000111001111

   ▷ *Pour le mot* 0110010*, on insère les bits de parité aux positions* $2^i$ *:* _ _0_110_010.
   *Le bit de partité en position :*
   $2^0$ *est 1 car les bits dépendant de ce bit sont les bits aux positions impaires. Donc on obtient :* 1_0_110_010*,*
   $2^1$ *est 0 car les bits dépendant de ce bit sont les bits 3, 6, 7, 10 et 11. Donc on obtient :* 100_110_010*,*
   $2^2$ *est 0 car les bits dépendant de ce bit sont les bits 5, 6 et 7. Donc on obtient :* 1000110_010*,*
   $2^3$ *est 1 car les bits dépendant de ce bit sont les bits 9, 10 et 11. Donc on obtient :* 10001101010.
   *Le code transmis pour le mot* 0110010 *est donc* 10001101010.

2. Quelle est la redondance $r$ ajoutée ? Donner une relation liant $m$ (taille du mot initiale), $n$ (taille après codage) et $r$.

   ▷ *On a* $n = m + r$.

3. Comment décoder un mot ? Comment vérifier s'il y a eu des erreur ? Comment corriger *une* erreur, le cas échéant ?

   ▷ *On vérifie que tous les bits de parité (ceux en positions* $2^i$*) sont corrects, c'est-à-dire qu'ils donnent la bonne parité des bits qu'ils contrôlent. Si il y a un ou plusieurs bits de parité qui ne donnent pas la bonne parité et que l'on suppose qu'au maximum ne se produit qu'une seule erreur, alors il y a un unique bit qui dépend exactement des bits de parité qui ne donnent pas la bonne parité et c'est donc lui qu'il faut inverser : si les bits de parité aux positions* $2^{i_1}$, $2^{i_2}$, ..., $2^{i_k}$ *ne donnent pas la bonne parité, alors le bit à la position* $\sum_{j=1}^{k} 2^{i_j}$ *est celui qu'il faut inverser pour que le code deviennent correct.*

4. Recevez (décodez) 1010110 puis 1011011111011011 puis 0010010110011100101. Il faut donner le mot décodé, et le mot corrigé en cas d'erreur (en supposant une seule erreur).

   ▷ *Pour le mot* `1010110`*, il y a un bit de parité qui ne donne pas la bonne parité, c'est le premier bit qui devrait valoir 0. C'est donc le bit en position* $2^0$ *qui doit être inversé, soit le bit de parité. Donc le code correct est* `0010110`*.*

   *Pour le mot* `0010010110011100101`*, les bits de parité aux positions* $2^0$ *et* $2^3$ *ne donnent pas la bonne parité. C'est donc le bit en position* $2^0 + 2^3 = 9$ *qui doit être inversé. Le code correct est donc* `0010010100011100101`*.*

5. Combien d'erreurs ce code peut-il détecter ? Produisez un exemple d'erreur non détectée.

   ▷ *deux.*

   *Partant du code* `10001101010`*, si les bits aux positions* $2^0$*,* $2^2$ *et* $2^0 + 2^2 = 5$ *sont inversés, alors le code obtenus* `00010101010` *est correct !*

6. Combien peut-il en corriger ?

   ▷ *une*

7. Montrer que, pour certaines valeurs de $n$, $m$ et $r$ (lesquelles ?) tout mot à $n$ bits est soit un mot du dictionnaire, soit diffère sur un seul bit d'un et d'un seul mot du dictionnaire. En d'autre termes, que tout mot reçu est soit interprêté comme étant correct, soit corrigé. Un code vérifiant cette propriété est dit *parfait*.

   ▷ *Il y a* $2^m$ *messages possibles et* $2^n$ *mots de code possibles. Si on suppose que lorsqu'une erreur sur un bit se produit, le mot de code diffère d'un seul bit d'un mot du dictionnaire, alors pour chaque message correct, il y a* $n$ *mots de codes incorrects (les* $n$ *mots obtenus en modifiant un des* $n$ *bits du mot de code). D'où chaque message peut donner* $n + 1$ *mots (le mot de code correct et les* $n$ *mots qui diffèrent d'un bit avec lui). Donc, comme* $n = m + r$*, on a :*

   $$(n+1)2^m \leqslant 2^n \Leftrightarrow n + 1 \leqslant 2^r \Leftrightarrow \log_2(n+1) \leqslant r.$$

   *D'où il faut au minimum* $\log_2(n+1)$ *bits de redondance pour fabriquer un code parfait. Ce qui prouve que le codage de Hamming est optimal (puisqu'il utilise exactement* $\log_2(n+1)$ *bits de redondance) pour détecter et corriger une erreur sur un bit dès que* $\log_2(n+1)$ *est un entier, c'est-à-dire pour* $n = 2^k - 1$*.*