ASR1 – TD1 : Codage et calcul

{ Andreea.Chis, Matthieu.Gallet, Bogdan.Pasca } @ens-lyon.fr

16 et 17 septembre 2008

1 Codage...

Étant donné un ensemble B de mots de n bits et un ensemble I d'informations différentes, on appelle *codage* de I sur n bits toute fonction de B dans I. Le codage est *total* si la fonction est surjective. Si la fonction n'est pas injective, le codage est dit *redondant*.

1. Quel est le nombre minimum de bits nécessaire pour un codage total de N informations ?
Réponse : Il faut $\lceil \log_2 N \rceil$ *bits.*

2 Les entiers

2.1 Comme au néolithique

1. Comment appelle-t-on un petit caillou en latin ?
Réponse : Calculus.

Les bergers de l'antiquité, qui ne savaient ni compter ni poser une addition, avaient une méthode utilisant cette technologie (et un système de numération unaire) pour faire la somme des effectifs de deux troupeaux.

2. Retrouvez-la.
Réponse : Un petit caillou par mouton. On fait la somme en mélangeant deux tas.

3. Comment pose-t-on une multiplication en cailloux ?
Réponse : Un tas pour le multiplicande et un tas pour le multiplieur. On construit un tas de même taille que celui du multiplicande, on l'accumule puis on retire un caillou du tas du multiplieur. Ainsi de suite jusqu'à ce que le tas du multiplieur soit vide.

2.2 Comme maintenant

1. Rappelez les principes du codage des entiers positifs en numération simple de position en base β.
Réponse :

$$(x_{n-1} \cdots x_1 x_0)_\beta = \sum_{i=0}^{n-1} x_i \cdot \beta^i.$$

2. Faites quelques additions d'entiers positifs sur 8 bits pour vous faire la main.
Réponse : $00011111_2\ (31) + 00010010_2\ (20) = 00110011_2\ (51)$.

3. Proposez une manière de coder les entiers relatifs.
Réponse :
Codage signe et valeur absolue : un nombre est codé sur n bits par son signe s sur 1 bit et sa valeur absolue v en binaire classique sur $n-1$ bits :

$$x = (-1)^x \cdot v.$$

On peut ainsi représenter les nombres de $-2^{n-1}+1$ à $2^{n-1}-1$.

4. Essayez de faire quelques additions. Par exemple $16+64$ et $42+(-17)$. Quels sont les problèmes ?
Réponse :
Pour le second calcul, il faut faire une soustraction, et c'est difficile.
On a un autre souci car zéro a deux représentations : $+0$ et -0. Donc ça rend les comparaisons difficiles aussi.

5. Trouvez une autre manière de coder les entiers relatifs qui soit plus pratique.
Astuce : pensez au complément à la base.
Réponse :

$$(x_{n-1}\cdots x_1x_0)_2 = -x_{n-1}\cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i.$$

6. Faites quelques additions et quelques soustractions d'entiers relatifs codés en complément à 2 sur 8 bits.
Réponse :
$11001101_2\ (-51) + 0100011_2\ (67) = 00010000_2\ (16)$,
$00101010_2\ (42) - 01101010_2\ (106) = 11000000_2\ (-64)$.

7. Posez quelques multiplications de nombres de 4 bits en binaire.
Réponse : $0101_2\ (5) \times 1101_2\ (13) = 01000001_2\ (65)$.

2.3 Comme chez les souris

Le *code de Gray* sur n bits permet de coder les nombres entre 0 et 2^n-1 de telle sorte que le codage de deux nombres consécutifs ne diffère que d'un seul bit. Par exemple un codage de Gray sur trois bits donne les codes : $000, 001, 011, 010, 110, 111, 101, 100$.

1. Un tel codage s'appelle aussi binaire réfléchi, pourquoi ?
Réponse : Si on note $G_n(i)$ le codage de Gray sur n bits du nombre i, le codage de Gray sur $n+1$ bits devient :

$$G_{n+1} = 0G_n(0), 0G_n(1), \cdots, 0G_n(2^n-1), 1G_n(2^n-1), \cdots, 1G_n(1), 1G_n(0).$$

2. À quoi sert-il ?
Réponse : À éviter des problèmes d'asynchronisme par exemple, ou bien plus faible consommation (conteur, addresses dans une mémoire).

3. Écrivez une procédure récursive donnant un codage de Gray d'un nombre quelconque de bits.
Réponse : Procédure récursive, en se servant de la symétrie.

2.4 Pour ceux qui sont en avance

1. Comment coder un entier naturel non borné ? Trouvez un codage de n sur $n+1$ bits, et un autre sur $2\lfloor \log_2 n \rfloor + 3$ bits. Peut-on faire encore mieux ?
Réponse :
En unaire : 0^n1 *($n+1$ bits).*
En binaire : on code d'abord en unaire la taille du codage binaire, puis le codage binaire proprement dit (sur $\lfloor \log_2 n \rfloor + 1$ *bits).*
On peut rajouter encore des niveaux d'encodage de la taille, voire même avoir une procédure adaptative, où le nombre de niveaux utilisés est aussi codé en unaire au début du nombre.

2. Généralisez le tout au cas où, au lieu de coder vers des bits, on code vers l'alphabet $\{0, 1, 2\}$.
Réponse : On peut coder la taille (c'est-à-dire le nombre de trits*) en binaire avec les chiffres* 0 *et* 1*, jusqu'au marqueur* 2*, puis coder le nombre proprement dit en base* 3*.*

3 Détection et correction d'erreurs

3.1 Généralités

Étant donnée une source émettant une information codée sur n bits reçue par un récepteur, on définit les termes suivants :
- un codage est *auto-vérificateur* s'il permet de déceler une erreur de transmission, et
- un codage est *auto-correcteur* s'il permet de reconstituer l'information.

1. Trouvez des exemples simples de codages auto-vérificateurs ou auto-correcteurs.
Réponse :
Auto-vérificateur : on calcule la somme (sur 8 *bits par exemple) de tous les octets transmis. Cependant le nombre d'erreurs détectables est toujours limité.*
Auto-correcteur : on envoie chaque information en double.

Le *contrôle de parité* utilise $n-1$ bits pour l'information et un bit de contrôle dit bit de parité. Celui-ci est positionné de telle sorte que le nombre total de bits à 1 de la configuration soit pair (parité paire) ou impair (parité impaire). On retrouve cette méthode dans le protocole de communication du lien série (UART/RS-232) par exemple.

2. Quelles sont les possibilités et les limites d'un tel codage ?
Réponse : Seul un nombre impair d'erreurs peut être détecté, car un nombre pair d'erreurs ne change pas la parité du mot.

3.2 Codes de Hamming

Le *code auto-correcteur de Hamming* permet de corriger un bit erroné dans une information de 4 bits (i_3, i_2, i_1, i_0). On code cette information au moyen de 3 bits supplémentaires p_2, p_1 et p_0 : le codage est $(i_3, i_2, i_1, p_2, i_0, p_1, p_0)$, où :
- p_0 est le bit de parité paire du sous-mot (i_3, i_1, i_0, p_0),
- p_1 est celui du sous-mot (i_3, i_2, i_0, p_1), et
- p_2 est celui de (i_3, i_2, i_1, p_2).

À la réception de $(i_3, i_2, i_1, p_2, i_0, p_1, p_0)$, on vérifie ces trois parités, et on définit t_k par : $t_k = 0$ si p_k est correct et $t_k = 1$ sinon.

Par magie, (t_2, t_1, t_0) est alors le rang dans $(i_3, i_2, i_1, p_2, i_0, p_1, p_0)$ du bit erroné, écrit en binaire : 000 si l'information est correcte, 001 si p_0 est erroné, 010 si p_1 est erroné, 011 si i_0 est erroné, 100 si p_2 est erroné, 101 si i_1 est erroné, 110 si i_2 est erroné ou 111 si i_3 est erroné.

On parle ici du code $(7, 4)$: 7 bits transmis pour 4 bits d'information effective.

1. Vérifiez le fonctionnement correct de l'algorithme de correction sur quelques exemples.
Réponse : On prend $(i_3, i_2, i_1, i_0) = 0110$. Comme $(i_3, i_1, i_0) = 010$, on a $p_0 = 1$. De même, $p_1 = 1$ et $p_2 = 0$. Le mot transmis est donc 0110011.
Si le mot est transmis sans erreur, les valeurs des p_k sont correctes, et ainsi le mot de correction est $(t_2, t_1, t_0) = 000$: pas d'erreur. Jusqu'ici tout va bien.
Supposons qu'une erreur est faite sur le bit i_1 : on reçoit un 0 au lieu d'un 1. Cela fausse les bits de parité p_0 et p_2. Le mot de correction devient alors 101, signalant une erreur sur le $5^{ème}$ bit, soit... i_1. Youpi !

2. Justifiez la propriété auto-correctrice d'un tel codage.
Réponse : Les bits p_0, p_1 et p_2 sont des bits de parité : ils peuvent ainsi détecter une erreur (un bit faux) dans leur sous-mot.
Or, on peut voir que les bits du sous-mot de p_0 sont les bits de rang 1, 3, 5 et 7 respectivement. Ainsi, t_0 est à 1 s'il y a eu au une erreur sur un de ces bits. En écrivant les rangs de ces bits en binaire, on obtient dans l'ordre 001, 011, 101 et 111 : le bit de poids faible de tous ces rangs est à 1.
De même, les bits du sous-mot de p_1 occupent les positions 2, 3, 6 et 7, soit respectivement 010, 011, 110 et 111. Ce coup-ci, toutes ces positions ont leur bit de poids 2^1 à 1. C'est louche, quelque chose se trame derrière tout ça...
Et il en va de même aussi pour p_2, dont les positions des bits (4, 5, 6 et 7) s'écrivent en binaire 100, 101, 110 et 111 : leur bit de poids fort est toujours à 1.
Si l'on résume, t_0 vaut 1 s'il y a eu une erreur sur un des bits de rang $??1$. Tout pareil pour t_1 qui passe à 1 en cas d'erreur sur un des bits de poids $?1?$, et enfin t_2 pour les bits de rang $1??$. Le mot (t_2, t_1, t_0) désigne alors bien directement le rang du bit erroné.

3. Peut-on définir un codage auto-correcteur de 4 bits utilisant moins de 3 bits supplémentaires ?
Réponse : Notons n le nombre de bits supplémentaires nécessaires au codage. On doit donc transmettre $4 + n$ bits, ce qui demande de pouvoir coder les $4 + n + 1$ cas possibles : aucune erreur, ou bien une erreur sur un des $4 + n$ bits transmis. Ce codage demande donc $n = \log_2\lceil 5 + n\rceil$ bits supplémentaires, soit au moins 3 bits.

4. Tentez de généraliser l'algorithme de Hamming à d'autres tailles de mots.
Réponse : n bits de parité permettent de coder 2^n cas, soit la position d'une erreur sur une transmission de $2^n - 1$ bits. Cela permet ainsi d'avoir $2^n - n - 1$ bits d'information.
Les bits de parités sont sur les rangs des différentes puissances de deux.
Par exemple, le cas $n = 4$ bits, avec 11 bits d'information : le mot transmis est $(i_{10}, i_9, i_8, i_7, i_6, i_5, i_4, p_3, i_3, i_2, i_1, p_2, i_0, p_1, p_0)$, avec les bits de parité suivants :
- *p_0 bit de parité paire sur $(i_{10}, i_8, i_6, i_4, i_3, i_1, i_0, p_0)$,*
- *p_1 sur $(i_{10}, i_9, i_6, i_5, i_3, i_2, i_0, p_1)$,*
- *p_2 sur $(i_{10}, i_9, i_8, i_7, i_3, i_2, i_1, p_2)$, et enfin*
- *p_3 sur $(i_{10}, i_9, i_8, i_7, i_6, i_5, i_4, p_3)$.*

3.3 *Bonus* : Comme à la NASA

Un des codes auto-correcteurs les plus efficaces à l'heure actuelle (tellement efficace qu'il est utilisé pour causer avec les bébêtes de la NASA qui se balladent sur Mars) est le codage Reed-Solomon. Plus proche de vous, il est aussi utilisé dans tout ce qui est télécommunications (ADSL, Usenet, téléphonie mobile, etc...) et dans les CD.

On souhaite donc coder n blocs B_i de k bits chacun. La première étape est de s'assurer que chaque bloc est correct. Ceci est réalisé en rajoutant un CRC (*Cyclic Redundancy Check*, un code auto-vérificateur) à chaque bloc.

L'idée parachutée est de considérer le polynôme $P(X)$ de degré $n-1$ et dont les coefficients sont donnés par les B_i :

$$P(X) = \sum_{i=0}^{n-1} B_i X^i.$$

1. Dans le cas où certains blocs sont faux, quelle propriété des polynômes permet de corriger $P(X)$ et donc de retrouver les valeurs de ces blocs ?
   Réponse : On code en plus des points par lesquels passe le polynôme. Il suffit alors de résoudre le problème d'interpolation pour retrouver les coefficients manquants.
2. Est-ce si facile que ça en a l'air ?
   Réponse : Non ! Mais ce n'est carrément pas l'objet de ce TD.