Détection et résolution d'erreurs

https://www.youtube-nocookie.com/embed/ecw8AbBKZCM

Maintenant que l'on sait représenter des nombre et des caractères en binaire, on peut maintenant voir comment faire dans les cas où il pourrait y avoir des erreurs.

Que ce soit dans la mémoire ou lors de communication, ce n'est pas infaillible et 1 seul bit qui change pourrait changer complètement l'information.

Le bit de parité (code auto-verificateur)

Supposons que notre code est le suivant : `01001000` pour calculer la parité paire, il suffit de compter le nombre de `1`. Dans ce cas il y a 2 fois `1`. Donc le bit de parité est `0`.

Si on compte un nombre impair de `1`, alors le bit de parité est `1`. En bref, il faut que le nombre total de `1` (en comptant à la fois le message et le bit de parité) soit **pair**.

Et pour la parité impaire, il suffit de faire l'inverse. On s'assure donc que le nombre total de `1` (en comptant message + bit de partité) soit **impair**.

Ainsi maintenant on peut vérifier un code et savoir si il a été altéré ou non. En revanche cette méthode est limitée, car si il y a un nombre pair d'erreurs, alors le bit de parité ne dédectera rien d'anormal. Aussi, on n'a aucune idée de où se trouve l'erreur.

La double parité (code auto-correcteur)

Maintenant imaginons que l'on a un message plus long. Comme `SHUMI` par exemple. On va donc le transcrire en binaire (avec le code UTF-8)

Car	b7	b6	b5	b4	b3	b2	b1	b0
S	0	1	0	1	0	0	1	1
H	0	1	0	0	1	0	0	0
U	0	1	0	1	0	1	0	1
M	0	1	0	0	1	1	0	1
I	0	1	0	0	1	0	0	1

Maintenant on va faire une parité *paire* horizontalement et une parité *impaire* verticalement comme vu précédemment..

Car	b7	b6	b5	b4	b3	b2	b1	b0	PP
S	0	1	0	1	0	0	1	1	0
H	0	1	0	0	1	0	0	0	0
U	0	1	0	1	0	1	0	1	0
M	0	1	0	0	1	1	0	1	0
I	0	1	0	0	1	0	0	1	1
PI	1	0	1	1	0	1	0	1	

Maintenant en vérifiant tous les bits de parités on peut retrouver une erreur facilement et la corriger

Car	b7	b6	b5	b4	b3	b2	b1	b0	PP
S	0	1	0	1	0	0	1	1	0
H	0	1	0	0	1	0	0	0	0
E	0	1	0	**0**	0	1	0	1	**0**
M	0	1	0	0	1	1	0	1	0
I	0	1	0	0	1	0	0	1	1
PI	1	0	1	**1**	0	1	0	1	

On peut voir ici qu'il y a eu une erreur au niveau du bit 4 sur le 3e caractère car la parité paire de la ligne et la parité impaire de la colonne ne sont plus correct. A cause de ça notre `U` est devenu un `E`.

Mais on sait donc précisément quel bit est la cause de l'erreur et comment le résoudre. En revanche cette méthode nécessite beaucoup de bits de vérification.

Le code de Hamming

Ici on se concentre sur le code de Hamming qui permet de corriger 1 bit en erreur, il existe des variantes plus complexes mais que l'on apprend pas ici.

Création du code de Hamming

Disons que l'on veut stoquer le code *S*, soit `1010011` en binaire.

Tout d'abord on va faire un tableau pour contenir notre code, mais qui va être un peu plus long pour contenir nos bits de parités

11	10	9	8	7	6	5	4	3	2	1

Maintenant on va bloquer tous les emplacements qui sont des puissances de 2 (qui sont nos bits de parités)

11	10	9	8	7	6	5	4	3	2	1
			k4				*k3*		*k2*	*k1*

Ensuite on peut écrire le message à coder dans les bits restants :

11	10	9	8	7	6	5	4	3	2	1
1	0	1	*k4*	0	0	1	*k3*	1	*k2*	*k1*

Maintenant on liste tous les numéros de bits qui valent `1` dans le tableau et on les convertis en binaire

```
11 → 1011
9  → 1001
5  → 0101
3  → 0011
```

Ensuite pour chaque colonne en binaire on va faire une parité de la colonne. Chaque bit de parité trouvé corresponds aux bits de parités dans le tableau.

```
11 → 1011
9  → 1001
5  → 0101
3  → 0011
     ----
     0100 (k4 = 0, k3 = 1, k2 = 0, k1 = 0)
```

On peut donc remplacer ces derniers dans le tableau

11	10	9	8	7	6	5	4	3	2	1
1	0	1	**0**	0	0	1	**1**	1	**0**	**0**

Notre code de Hamming est donc : `10100011100`

Vérification et correction dans un code de Hamming

Maintenant quand on veut vérifier un code reçu, voici comment faire. On va partir du code `10101011100` et on va le vérifier (et si besoin corriger)

Tout d'abord on va placer tout ceci dans un tableau avec les bits numérotés :

11	10	9	8	7	6	5	4	3	2	1
1	0	1	0	1	0	1	1	1	0	0

Maintenant on liste tous les bits qui sont a 1 et on convertis leur position en binaire :

```
11 → 1011
9  → 1001
7  → 0111
5  → 0101
4  → 0100
3  → 0011
```

Ensuite, on va faire la parité de chaque colonne. Le résultat corresponds à la position du bit en erreur en binaire. Si le résultat done `0` partout, alors il n'y a pas d'erreur détectée.

```
11 → 1011
9  → 1001
7  → 0111
5  → 0101
4  → 0100
3  → 0011
     ----
     0111 → 7 (le bit 7 doit donc être inversé)
```

Nous pouvons donc corriger notre code :

11	10	9	8	7	6	5	4	3	2	1
1	0	1	0	**0**	0	1	1	1	0	0

Pour obtenir le message d'origine il suffit de retirer tous les bits qui sont à des positions qui correspondent à des puissances de 2

11	10	9	8	7	6	5	4	3	2	1

1	0	1		0	0	1		1		

Ce qui nous donne donc le message suivant : `1010011` soit `S` en ASCII. Nous retrouvons donc le même résultat que le message encodé au départ □□

Revision #1
Created 27 April 2023 04:15:05 by SnowCode
Updated 27 April 2023 04:27:26 by SnowCode