TP Réseaux
Le code de Hamming

Ballot Corentin, Simon Camille 09/04/2017

Ce compte-rendu présente l'implémentation d'un programme JAVA permettant de coder et de vérifier une chaîne binaire à l'aide de la méthode de Hamming.

L'archive Reseau-TP2-CorentinBallot-CamilleSimon.tar.gz. contient :

- Les fichiers JAVA permettant l'exécution du programme
- Les graphiques utilisés pour ce compte-rendu

Le sujet du TP est accessible ici : TP Réseaux - Le code de Hamming.

1 Analyse du sujet

Dans ce TP, nous devons réaliser un programme qui code et décode une chaîne binaire. La méthode utiliser est celle de Hamming, cette méthode ajoute de nouveaux bits au message afin de permettre de le vérifier. La chaîne binaire que l'on souhaite codé est appelé *mot* et la nouvelle chaîne après codage est appelée *mot de Hamming*. Dans la suite de cette section, nous allons présenter les méthodes de codage et de décodage.

1.1 Principe du code de Hamming

La code de Hamming va ajouter *des bits de contrôle* à un mot pour vérifier que le mot a été correctement transmit à travers le réseau. À un mot de taille *m* on va ajouter *n* bits de contrôle, on obtient un mot de Hamming de longueur $2^n - 1$ bits. On décrit un mot de Hamming par deux nombres séparé par un tiret par exemple : 31 - 26, où le premier nombre correspond à la taille du mot de Hamming, et le deuxième au nombre de bits du message à transmettre. Passons maintenant à la méthode de génération d'un mot de Hamming.

1.2 Génération du code de Hamming

La première étape consiste à déterminer la taille du mot de Hamming ainsi que le nombre de bots de contrôle. Soit *m* et *n* sont déjà donnée, dans ce cas la taille du mot de Hamming est *m*+*n*. Dans le cas contraire, on compte le nombre de bits du mot. Puis pour déterminer le nombre de bit de parité on calcule *n* tel que $2^n > m$, avec *n* le plus petit possible.
Par exemple pour le mot 101110010, *m* vaut 9, la puissance de 2 supérieur et de valeur minimal est $2^4 = 16$, on obtient donc *n* égal 4. Le mots de Hamming correspondant aura une longueur de 15 bits.

Une fois la taille du mot de Hamming définit, il reste à déterminer la valeur de chaque bits. On numérote les bits du mot de Hamming de droite à gauche à partir de 1. Sur la figure ci-dessous, les cases avec des tirets représentes les bits dont on ignore la valeur. Les valeurs correspondent aux indices des cases.

-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Chaque indice égal à 2^k va être occupé par un bit de parité noté C_k. Les autres emplacements sont rempli de gauche à droite avec les zéro supplémentaire puis de droite à gauche avec les bits du mot d'origine. On obtient la figure suivante :

0	0	1	0	1	1	1	C_3	0	0	1	C_2	0	C_1	C_0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Il ne reste plus qu'à calculer les valeurs des bits de parité. Pour le bit de parité d'indice *i*, on fait la somme des bits par la gauche en alternant *i* bits ajouter à la somme puis *i* bits exclu. La somme doit contenir $\lfloor \frac{n+m}{2} \rfloor$ terme et ne doit pas contenir les bits de contrôle.

Calculons les bits de parité de l'exemple :

- $C_0 = \sum$ *des bits d'indice 15, 13, 11, 9, 7, 5, 3*, donc $C_0 = 0+1+1+1+0+1+0 = 0$

0	0	1	0	1	1	1	C_3	0	0	1	C_2	0	C_1	**0**
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

- $C_1 = \sum$ *des bits d'indice 15, 14, 11, 10, 7, 6, 3*, donc $C_1 = 0+0+1+1+0+0+0 = 0$

0	0	1	0	1	1	1	C_3	0	0	1	C_2	0	**0**	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

- $C_2 = \sum$ *des bits d'indice 15, 14, 13, 12, 7, 6, 5*, donc $C_2 = 0+0+1+0+0+0+1 = 0$

0	0	1	0	1	1	1	C_3	0	0	1	**0**	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

- $C_3 = \sum$ *des bits d'indice 15, 14, 13, 12, 11, 10, 9*, donc $C_3 = 0+0+1+0+1+1+1 = 0$

0	0	1	0	1	1	1	**0**	0	0	1	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

On obtient le mot de Hamming : 001011100010000.

1.3 Correction d'un mot de Hamming

Une fois le message codé et transmis, le destinataire doit vérifier que le message obtenu ne contient pas d'erreur. Un mot de Hamming permet de détecter et de corriger une erreur.

La méthode permettant de vérifier le mot est similaire à la méthode de codage. On appelle C'_k le bit correcteur, celui-ci est calculer grâce à la somme permettant d'obtenir C_k plus la valeur de C_k lui-même. Voici une exemple avec le mot de Hamming 101010010010 :

0	0	0	1	0	1	0	1	0	0	1	0	0	1	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

$C'_0 = \sum$ *des bits d'indice 15, 13, 11, 9, 7, 5, 3, 1*, donc $C'_0 = 0+0+0+0+0+1+0+0 = 1$

0	0	0	1	0	1	0	1	0	0	1	0	0	1	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

$C'_1 = \sum$ *des bits d'indice 15, 14, 11, 10, 7, 6, 3, 2*, donc $C_1 = 0+0+0+1+0+0+0+1 = 0$

0	0	0	1	0	1	0	1	0	0	1	0	0	1	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

$C'_2 = \sum$ *des bits d'indice 15, 14, 13, 12, 7, 6, 5, 4*, donc $C'_2 = 0+0+0+1+0+0+1+0 = 0$

0	0	0	1	0	1	0	1	0	0	1	0	0	1	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

$C'_3 = \sum$ *des bits d'indice 15, 14, 13, 12, 11, 10, 9, 8*, donc $C'_3 = 0+0+0+1+0+1+0+1 = 1$

Une fois la valeur des C'_k obtenus, on peut connaître l'emplacement de l'erreur en plaçant les C'_k de la manière suivante : $C'_k C'_{k-1} ... C'_2 C'_1$. Pour l'exemple, cela donne le code d'erreur : 1001. Une fois ce nombre binaire convertie en décimal, on trouve l'emplacement de l'erreur. Ici, il s'agit du bit d'indice 9.

Remarque : Le codage de Hamming ne permet pas de détecter et de corriger plus d'une erreur. Plus le message est long, plus Hamming est efficace, en effet le coefficient d'efficacité est $\frac{m}{m+n}$. Ainsi pour un message 31 - 26 le coefficient d'efficacité est de 83%.

2 Choix techniques & mise en œuvre

2.1 Choix techniques

Dans ce TP nous devons réaliser une interface graphique. Nous avons choisi d'utiliser Swing. Bien que cette technologie soit ancienne, elle reste largement utilisée et enseignée.

2.2 Mise en œuvre

Le travail demandé peut être séparé en deux volets, d'une part la conception des algorithmes de codage et de décodage et d'autre part, l'aspect graphique de l'application.

2.2.1 Algorithmes

Les algorithmes permettant la génération et la vérification des mots de Hamming ce trouvent dans le fichier *resultat.java*. Nous allons maintenant décrire le fonctionnement de ses deux algorithmes :

- CODAGE

  On commence par déterminer la taille de la trame, c'est à dire la taille du tableau sur lequel on va travailler, que l'on note *n*. Tant que $2^n <$ *la longueur du mot à codé*, on augmente la valeur de *n*.

  Dans la partie 1.2, les bits sont numéroté de droite à gauche, hors les indices de tableau en *Java* vont de gauche à droite. On crée donc une nouvelle chaîne de caractère de taille *n* et contenant le mot inversé. On ajoute des zéro à l'emplacement des caractères entre les indices *longueur du code - 1* et *n - 1*.

  On transfert les valeurs des bits de la chaîne de caractères dans un tableau de caractères en laissant des espaces libre aux indices égaux à une puissance de 2. L'utilisation d'un tableau sera plus simple pour le calcul des bits de parité comparé à une chaîne de caractères.

  Il reste maintenant à calculer la valeur des bits de parité. On parcoure le tableau dans l'ordre de ses indices. Si l'indice n'est pas une puissance de deux, on passe à la case suivante. En revanche, si l'indice est une puissance de 2, on parcours le tableau en commençant par l'indice le plus élevé et on somme *i+1* bits puis on en saute *i+1* et ainsi de suite. Si la somme est paire, on inscrit 0 pour la valeur du bit de contrôle correspondant, sinon on inscrit 1.

  Une fois tout les indices passé en revu, en transforme le tableau de caractères en une chaîne que l'on transmet à la méthode d'affichage.

- CORRECTION

  Pour que la correction fonctionne, il faut que le mot saisi soit de taille $2^n - 1$. On commence donc par déterminer la taille du mot et à le transférer dans une tableau de caractère de la même façon que dans la fonction de codage.

  Pour vérifier la valeur du bit de correction C'_k, on somme comme pour le calcul de C_k et ajoutant la valeur de C'_k lui même. On enregistre le résultat dans le tableau *code hamming* dont l'indice correspond au k du C'_k bit de correction.

  Lorsque toutes les sommes ont été enregistrées, on peut maintenant détermine la position de l'erreur. On obtient la position en sommant la valeur de la case multiplié par $2^{\textit{indice de la case}}$.

  Enfin, on transmet la position de l'erreur à la fonction d'affichage.

2.2.2 Design

Une part non négligeable du travail consiste à créer une interface utilisateur simple, efficace et intuitive. Voici à quoi ressemble notre application :

Figure 1: Visuel de l'application

La fenêtre est séparé en deux parties, la partie supérieurs (1) où l'utilisateur va entrer ses informations et la partie inférieur cf. **Figure:** 3 le programme va afficher le résultat. L'ensemble de la fenêtre est générée par le fichier *Fenetre.java*.

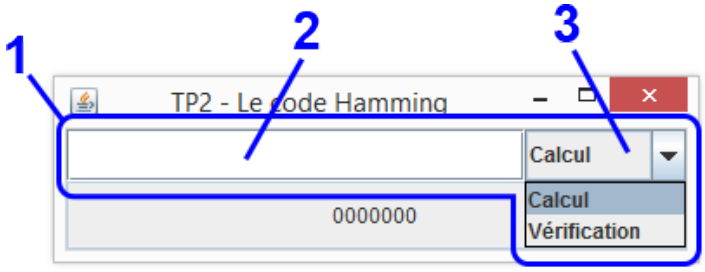

Figure 2: Visuel de l'application. (1) L'espace de saisie des paramètres, (2) JTextField pour le message, (3) JComboBox de sélection de l'opération à effectuer.

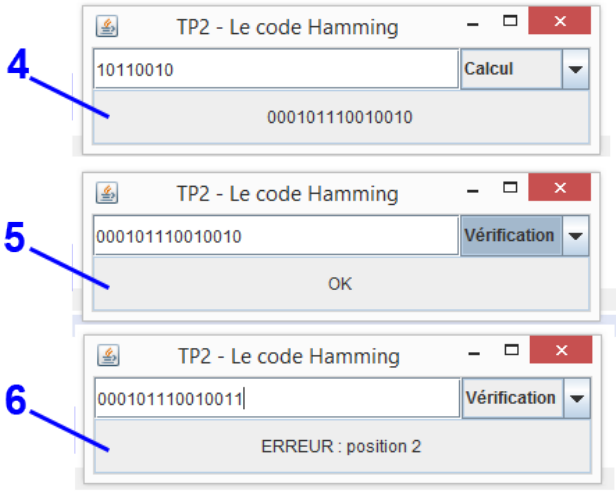

Figure 3: Visuel de l'application. (4) Message après codage, (5) le message ne contient pas d'erreur, (6) le message contient une erreur en position 2.

La partie inférieur va être différente en fonction de se qui est demandé au programme. Pour le calcul, il affichera le mot avec les bits de contrôle (4). Pour la vérification, on aura soit "OK" si le message ne contient pas d'erreur (5), sinon, il affichera la position de l'erreur (6). Les indices pour position des bits incorrecte vont de gauche à droite, dans la figure ci-dessus le bit incorrecte est le deuxième bit en partant de la droite.

3 Tests

Figure 4: Résultat de l'application pour le codage du message 10110010

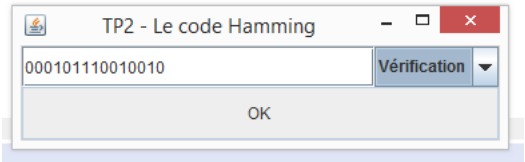

Figure 5: Résultat de l'application pour la vérification d'un message correcte

Figure 6: Résultat de l'application pour la vérification d'un message incorrecte

4 Conclusion

Dans ce TP, nous avons réalisé une application graphique permettant de coder et de vérifier un message de Hamming. Nous avons appliqué la méthode cu en TD. Ce travail nous a permis d'exercer nos compétences en algorithmique mais également de comprendre de manière approfondi le fonctionnement de la méthode de Hamming.