International Journal of Innovation and Applied Studies
ISSN 2028-9324 Vol. 28 No. 2 Jan. 2020, pp. 478-486

http://www.ijias.issr-journals.org/

Implémentation en Java du système de codage d'informations basé sur le code de Hamming (7,4)

[Java Implementation of Information Coding System Based on Hamming Code (7.4)]

KABEYA TSHISEBA Cedric

Département de Mathématique et Informatique, Faculté de Sciences, Université Pédagogique Nationale (UPN), Ngaliema, Kinshasa, RD Congo

ABSTRACT: The idea of linear coding is simple: by an injective linear application we send a space of binary words in a larger space, hoping that the redundancy introduced helps us to detect and correct the transmission errors. Among the so-called linear codes, we consider in the context of this article the Hamming code which is a perfect code, because for a given code length there is no other more compact code having the same capacity correction. In this sense its yield is maximum. In this work, we have proposed an algorithm based on the above characteristics of the Hamming code, which we can implement in a given programming language.

KEYWORDS: Error correcting codes, linear codes, cyclic codes, Hamming codes, Hamming weight, Hamming distance.

RÉSUMÉ: L'idée du codage linéaire est simple: par une application linéaire injective on envoie un espace de mots binaires dans un espace plus grand, en espérant que la redondance introduite nous aide à détecter et corriger les erreurs de transmission. Parmi les codes dits linéaires, nous considérons dans le cadre de cet article le code de Hamming qui en soit est un code parfait, du fait que pour une longueur de code donnée, il n'existe pas d'autre code plus compact ayant la même capacité de correction. En ce sens, son rendement est dit maximal. Dans ce travail, nous avons su proposer un algorithme basé sur les caractéristiques ci-dessus du code de Hamming, que nous avons ensuite bien pu implémenter ici en Java.

MOTS-CLEFS: Codes correcteurs d'erreurs, codes linéaires, codes cycliques, codes de Hamming, poids de Hamming, distance de Hamming.

1 INTRODUCTION

Communiquer à grande distance tel qu'avec les sondes spatiales, à l'autre bout du système solaire pose le problème de la fiabilité du message transmis. A cause de diverses sources de perturbations électromagnétiques, une transmission sur une telle distance est obligatoirement parasitée. Pourtant, dans ce domaine et dans bien d'autres, il est primordial que les informations collectées par les sondes soient bien reçues. D'où, le recours dans le cadre de cet article scientifique aux codes correcteurs d'erreurs, en vue de sécuriser la transmission, en rajoutant au message à transmettre des informations supplémentaires, qui permettent de reconstituer le message au niveau du récepteur.

A travers cet article nous proposons un algorithme pour le codage et le décodage des messages à caractère confidentiel, tout en se basant sur un système de codage linéaire de Hamming (7,4).

Ce qui a orienté notre choix vers ce système de codage, c'est le fait qu'il soit éminemment utilisé, dans le cadre de cryptage de données critiques, censé n'être connu que de l'expéditeur et du destinataire.

Corresponding Author: KABEYA TSHISEBA Cedric

Bien long à décoder, l'objectif de ce code est la transmission d'un message de quatre bits avec suffisamment de redondances pour que, même si une altération se produit, le récepteur soit capable de corriger automatiquement l'erreur.

1.1 INTRODUCTION À LA THÉORIE DE CODES CORRECTEURS D'ERREURS

1.1.1 CODES LINÉAIRES

En théorie des codes, un code linéaire est un code correcteur ayant une certaine propriété de linéarité.

Les codes linéaires sont utilisés pour la détection d'altérations [1], avec comme méthode de correction associée une demande de retransmission, la technique utilisée la plus usuelle est alors la somme de contrôle. Elles sont utilisées dans une multitude de situations, depuis quelques erreurs isolées à de vastes altérations ou des phénomènes d'effacements.

1.1.1.1 DÉFINITIONS [3][4][4]

Soit p un nombre premier, d'une puissance de p, n un entier strictement positif et k un entier plus petit que n. Un code linéaire C de dimension k et de longueur n est un sous-espace vectoriel de F_d^n de dimension k. Si d est égal à deux, le code est dit binaire. Sinon, on parle de code linéaire de base d.

Au regard de notre définition, F_d désigne l'unique corps à d éléments. On remarque que l'espace vectoriel des suites à valeurs dans F_d est identifié à F_d^n. L'espace vectoriel F_d^n est muni de la distance de Hamming.

Comme pour les autres codes correcteurs, la notion de paramètres s'applique. Cependant, pour tenir compte de la structure d'espace vectoriel, elle est un peu modifiée :

Les paramètres d'un code sont notés $[n, k, \delta]$ ou δ désigne la distance minimale entre deux points du code.

La définition de paramètre pour les codes linéaires n'est donc pas compatible avec celle, plus générique utilisées pour les codes correcteurs. Pour cette raison, traditionnellement les paramètres d'un code linéaire sont notés $[n, k, \delta]$ et ceux d'un code correcteur général $\{n, M, \delta\}$.

Il faudra noté l'existence ici d'une application d'encodage : φ.

L'application d'encodage φ d'un code linéaire est une application linéaire injective de F_d^k dans F_d^n.

La matrice G de F_d^n dans les bases canoniques est dite matrice génératrice du code C. Elle vérifie l'égalité suivante :

$$\forall x \in F_d^k \; x \,.\, G = \varphi(x) \in C \subset F_d^n$$

Le contrôle permettant la vérification et les éventuelles corrections est donné par une application linéaire h de F_d^n dans F_d^{n-k} ayant pour noyau C. La théorie de l'algèbre linéaire montre qu'une telle application existe, il suffit par exemple de considérer un projecteur sur un sous-espace supplémentaire de C parallèlement à C.

La matrice H de h dans les bases canoniques est dite matrice de contrôle du code C. Elle vérifie les propriétés suivantes:

$$\forall x \in F_d^n \; H^t \,.\, x = 0 \Leftrightarrow x \in C$$

Le terme de matrice de parité est aussi utilisé pour désigner la matrice de contrôle.

1.1.1.2 PROPRIÉTÉS

Toutes les propriétés de l'algèbre linéaire s'appliquent aux codes linéaires. Le code est ainsi à la fois plus facile à implémenter et à décoder. De plus les outils de génération d'espace vectoriel comme l'espace dual ou le produit tensoriel permettent de concevoir des nouveaux codes, parfois plus adaptés aux contraintes industrielles.

1.1.1.3 MATRICE GÉNÉRATRICE

L'application d'encodage est linéaire, elle se représente donc et se calcule grâce à sa matrice génératrice. Un code est entièrement défini par sa matrice génératrice, de dimension $k \times n$. De plus comme les propriétés de son code ne dépendent que de la géométrie $\varphi(E)$. Si f est un isomorphisme de E, le code défini par l'application $\varphi o f$ est le même que celui de φ. Ce qui donne lieu à la définition suivante :

Deux codes sur un même alphabet F_d de longueur k définis par deux matrices génératrices G et G' tel qu'il existe une matrice carrée inversible P d'ordre k vérifiant $G = G'.P$ sont dits équivalents.

Il existe une forme particulièrement simple pour la matrice G :

Un code linéaire dont la matrice génératrice possède pour k premières lignes une matrice identité d'ordre k est dit code systématique.

Les coordonnées de la matrice C correspondent à la redondance, leur objectif est la détection et la correction d'erreurs éventuelles:

Les $n - k$ dernières coordonnées d'un mot du code systématique sont dites bits de contrôle ou parfois somme de contrôle.

1.1.1.4 MATRICE DE CONTRÔLE

Dans le cas linéaire, le code est un sous-espace vectoriel de dimension k. Il existe alors une application linéaire surjective de F dans un espace de dimension $n - k$ ayant pour noyau exactement le code :

$$x \in \varphi \Leftrightarrow H^t . x = 0$$

Une matrice de contrôle d'un code $\varphi(E)$ est une matrice H de dimension $n \times n - k$ tel que :

$$Si\ G = (c^{Ik})\ alors\ H = (-C\ I_{n-k})$$

1.1.1.5 DISTANCE DE HAMMING [5]

Dans le cas d'un code linéaire, la distance de Hamming s'exprime comme une distance issue d'une pseudo-norme. Le poids de Hamming, qui a un code associe le nombre de coordonnées non nulles, joue ici le rôle de pseudo-norme.

Si ω désigne le poids de Hamming pour un code linéaire C, alors la distance de Hamming d est définie par la formule suivante :

$$\forall x, y \in C\ d(x, y) = \omega(x - y)$$

La linéarité de la structure sous-jacente introduit une propriété directe :

La distance minimale δ entre deux points du code est égale au minimum du poids des mots du code non nuls.

Pour s'en convaincre, il suffit de remarquer que si x et y sont deux mots du code, alors leur différence est aussi un mot du code.

1.1.1.6 CODES DE HAMMING [5]

Un code de Hamming est un code correcteur linéaire. Il permet la détection et la correction automatique d'une erreur si elle ne porte que sur une lettre du message.

Pour une longueur de code donnée il n'existe pas d'autre code plus compact ayant la même capacité de correction. En ce sens son rendement est maximal. D'où, la perfection du code de Hamming.

Il existe une famille de codes de Hamming ; le plus célèbre et le plus simple après le code de répétition binaire de dimension trois et de longueur un est sans doute le code binaire de paramètres [*7,4,3*]. Pour chaque alphabet ayant pour nombre de lettres une puissance d'un nombre premier et pour chaque longueur l de code il existe un code de Hamming utilisant cet alphabet et de longueur au moins égal à l.

Plusieurs méthodes permettent de construire un code de Hamming. Une approche consiste à rechercher les codes cycliques de distance minimale égale à trois, le code apparait alors comme un cas particulier de code BCH (*Bose, Ray-Chaudhuri et Hocquenghem*). Il est aussi possible d'utiliser uniquement les outils de l'algèbre linéaire et particulièrement la théorie des matrices.

1.1.1.7 CODE DE HAMMING (7,4) [6]

En théorie des codes, le Code de Hamming (*7,4*) est un code correcteur linéaire binaire de la famille des codes de Hamming. À travers un message de sept bits, il transfère quatre bits de données et trois bits de parité. Il permet la correction d'un bit

erroné. Autrement dit, si, sur les sept bits transmis, l'un d'eux au plus est altéré (un « zéro » devient un « un » ou l'inverse), alors il existe un algorithme permettant de corriger l'erreur.

Le code de Hamming(*7,4*) a pour objectif la transmission d'un message de quatre bits avec suffisamment de redondances pour que, même si une altération se produit, le récepteur soit capable de corriger automatiquement l'erreur. Le message envoyé est en conséquence plus long. Dans la pratique il contient sept bits, quatre composent le message et les trois autres servent à détecter et à corriger l'erreur, si nécessaire.

L'erreur que protège ce code est nécessairement limitée. Si les sept bits transmis sont tous modifiés, il est vain d'espérer retrouver le message originel. La protection contre la modification d'un unique bit est parfois largement suffisant, c'était le cas de la situation d'Hamming. Une modification d'un unique bit, c'est-à-dire d'un trou non lu sur la carte perforée ou d'une absence de trou considéré comme un trou est relativement rare. Supposons que cette situation se produit sur une série de quatre en moyenne une fois sur mille. Alors cinquante heures d'exécution de programme, qui utilisent par exemple de l'ordre de mille séries voit sa probabilité d'arrêt supérieure à soixante pour cent. La probabilité de trouver deux erreurs sur une série de sept est inférieur à un sur cinq cent mille. L'utilisation de mille séries de longueur sept possède une probabilité d'arrêt inférieure à deux pour mille dans le cas d'une utilisation de cinquante heures si les messages sont protégés par le code. C'était largement suffisant pour éradiquer la frustration d'Hamming.

2 ALGORITHME BASÉ SUR LES CODES DE HAMMING (7,4)

Nous présentons dans cette partie de notre article un algorithme lié au codage et décodage de messages à caractère confidentiel, tout en se basant sur un système de codage linéaire, particulièrement le code de Hamming (*7,4*).

Nous avons porté notre choix sur ce système de codage pour le fait qu'il soit de plus en plus utilisé, et plus encore par les experts de la sécurité informatique, dans le cadre de cryptage de données critiques, censé n'être connu que de l'expéditeur et du destinataire.

L'objectif du code est la transmission d'un message de quatre bits avec suffisamment de redondances pour que, même si une altération se produit, le récepteur soit capable de corriger automatiquement l'erreur. Le message envoyé est en conséquence plus long. Dans la pratique il contient sept bits, quatre composent le message et les trois autres servent à détecter et à corriger l'erreur, si nécessaire.

2.1 ALGORITHMES PROPOSÉ

Coder et décoder un message tout en se basant sur le code de Hamming, particulièrement le Hamming(*7,4*), revient respecter les étapes ci-dessous:

2.1.1 LE CODAGE

Cette étape consiste à la construction du code, identifiant ainsi les quatre bits de données notés d1, d2, d3 et d4 (4)et des trois bits de parité notés p1, p2, p3 (3 ➔ 4+3 = 7Bits).

2.1.1.1 ALGO

Variable

*d1, d2, d3, d4, p1, p2, p3 Binaire /*Déclaration des variables liées aux bits de données (d1, d2, d3, d4) et des variables liées au bits de parités(p1, p2, p3).*/*

Début

*/*Entré du message, constitué de quatre bits de données*/*
Saisir d1
Affecter d1 : d1
Saisir d2
Affecter d2 : d2
Saisir d3
Affecter d3 : d3

Saisir d4
Affecter d4 : d4
Afficher «le message clair est : »d1d2d3d4
*/*Codage du message et ajout des bits de parités*/*
Saisir p1
Affecter p1 : p1
Saisir p2
Affecter p2: p2
Saisir p3
Affecter p3 : p3
Afficher «le message codé est : » p1p2d1p3d2d3d4

Fin

2.1.2 LE DECODAGE

Le précèdent point va nous permettre de coder un message tout en se basant sur le code de hamming(7,4), tandis que cette étape va consister au décodage du code codé sur base du code de hamming(7,4)

2.1.2.1 ALGO

Variable

*b1, b2, b3, b4, b5, b6, b7 Binaire /*Déclaration des variables liées aux bits codés constituant donc le message non clair , qu'il faudra par la suite vérifier pour y identifier des probables erreurs */*

Début

*/*Entré des bits du message codé, constitué de quatre bits de données*/*
Saisir b1
Affecter b1 : b1
Saisir b2
Affecter b2 : b2
Saisir b3
Affecter b3 : b3
Saisir b4
Affecter b4 : b4
Saisir b5
Affecter b5 : b5
Saisir b6
Affecter b6 : b6
Saisir b7
Affecter b7 : b7
Afficher «le message codé reçu est : » b1b2b3b4b5b6b7
*/*Détection et correction d'erreurs dans le message codé reçu */*
Si b1+b3+b5+b7 = 0 alors
    Afficher «Pas d'erreurs pour le premier bit de parité(p1) »
Sinon
    p1=b1
    Si p1 = 1 alors
        Affecter à p1 : 0
    Sinon
        Affecter à p1 :1
    Fin si
Fin si
Si b2+b3+b6+b7 = 0 alors

```
        Afficher «Pas d'erreurs pour le premier bit de parité(p2) »
Sinon
        P2=b2
        Si p2 = 1 alors
                Affecter à p2 : 0
        Sinon
                Affecter à p2 :1
        Fin si
Fin si
Si b4+b5+b6+b7 = 0 alors
        Afficher «Pas d'erreurs pour le premier bit de parité(p3) »
Sinon
        P3=b4
        Si p3 = 1 alors
                Affecter à p3: 0
        Sinon
                Affecter à p3 :1
        Fin si
Fin si

Afficher «le bon message codé est : » p1p2d1p3d2d3d4
Afficher «le bon message clair est : » p1p2d1p3d2d3d4
```

Fin

3 LE SYSTÈME DE CODAGE «HAMMING(7,4)» EN JAVA

Dans cette partie de notre article nous présentons une possible implémentation de notre algorithme ci-dessus basé sur le code correcteurs d'erreur de Hamming(7,4), particulièrement ici en Java.

Code source Hamming (7,4) en Java :

```
import java.util.*;
class Code_Hamming {
    public static void main(String args[]) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Nombre de bits du Message:");
        int numero1 = scan.nextInt();
        int numero2[] = new int[numero1];
        for(int i=0 ; i < numero1 ; i++) {
            System.out.println("Inserer le bit N°. " + (numero1-i) + ":");
            numero2[numero1-i-1] = scan.nextInt();
        }
        System.out.println("Vous avez inseré:");
        for(int i=0 ; i < numero1 ; i++) {
            System.out.print(numero2[numero1-i-1]);
        }
        System.out.println();

        int numero3[] = generation_code(numero2);

        System.out.println("Voici le code géneré:");
        for(int i=0 ; i < numero3.length ; i++) {
            System.out.print(numero3[numero3.length-i-1]);
        }
        System.out.println();
```

// La différence entre les tailles du tableau d'origine et du nouveau tableau nous donnera le nombre de bits de parité ajoutés.

```
        System.out.println ("Entrez la position d'un bit à modifier pour vérifier la détection d'erreur au niveau du destinataire (0 pour l'absence d'erreur):");
        int erreur = scan.nextInt();
        if(erreur != 0) {
            numero3[erreur-1] = (numero3[erreur-1]+1)%2;
        }
        System.out.println("Code envoyé:");
        for(int i=0 ; i < numero3.length ; i++) {
            System.out.print(numero3[numero3.length-i-1]);
        }
        System.out.println();
        reception(numero3, numero3.length - numero2.length);
    }

    static int[] generation_code(int numero2[]) {
        int numero3[];
```

// On trouve le nombre de bits de parité requis:

```
        int i=0, compte_parite=0 ,j=0, k=0;
        while(i < numero2.length) {

            if(Math.pow(2,compte_parite) == i+compte_parite + 1) {
                compte_parite++;
            }
            else {
                i++;
            }
        }

        numero3 = new int[numero2.length + compte_parite];

        for(i=1 ; i <= numero3.length ; i++) {
            if(Math.pow(2, j) == i) {

                numero3[i-1] = 2;
                j++;
            }
            else {
                numero3[k+j] = numero2[k++];
            }
        }
        for(i=0 ; i < compte_parite ; i++) {

            numero3[((int) Math.pow(2, i))-1] = recept_parite(numero3, i);
        }
        return numero3;
    }

    static int recept_parite(int numero3[], int puissance) {
        int parite = 0;
        for(int i=0 ; i < numero3.length ; i++) {
```

```
                if(numero3[i] != 2) {
                        int k = i+1;
                        String s = Integer.toBinaryString(k);

                        int x = ((Integer.parseInt(s))/((int) Math.pow(10, puissance)))%10;
                        if(x == 1) {
                                if(numero3[i] == 1) {
                                        parite = (parite+1)%2;
                                }
                        }
                }
        }
        return parite;
}

        int puissance;

        int parite[] = new int[compte_parite];

        String syndrome = new String();

        for(puissance=0 ; puissance < compte_parite ; puissance++) {

                for(int i=0 ; i < numero2.length ; i++) {

                        int k = i+1;
                        String s = Integer.toBinaryString(k);
                        int bit = ((Integer.parseInt(s))/((int) Math.pow(10, puissance)))%10;
                        if(bit == 1) {
                                if(numero2[i] == 1) {
                                        parite[puissance] = (parite[puissance]+1)%2;
                                }
                        }
                }
                syndrome = parite[puissance] + syndrome;
        }

        int location_erreur = Integer.parseInt(syndrome, 2);
        if(location_erreur != 0) {
                System.out.println("erreur is at location " + location_erreur + ".");
                numero2[location_erreur-1] = (numero2[location_erreur-1]+1)%2;
                System.out.println("Code corrigé:");
                for(int i=0 ; i < numero2.length ; i++) {
                        System.out.print(numero2[numero2.length-i-1]);
                }
                System.out.println();
        }
        else {
                System.out.println("Pas d'erreur dans le message.");
        }

        System.out.println("Message envoyé:");
        puissance = compte_parite-1;
        for(int i=numero2.length ; i > 0 ; i--) {
                if(Math.pow(2, puissance) != i) {
                        System.out.print(numero2[i-1]);
```

```
                    }
                    else {
                        puissance--;
                    }
                }
                System.out.println();
            }
        }
```

4 CONCLUSION

Nous avons dans cet article proposé un algorithme basé sur le code linéaire, particulièrement le code de Hamming(7,4), un code très utilisé mais long et pas facile à employer à mesure que le message devient plus complexe.

Nous nous sommes limité à la proposition de cet algorithme dans l'article, ouvrant ainsi comme perspective pour les autres chercheurs de pouvoir sur base de ce même algorithme, implémenter dans un langage de programmation donné, un système de codage et décodage de message basé sur code de Hamming (7,4).

RÉFÉRENCES

[1] Khadidja SERIR, 2011, « Application des codes correcteurs d'erreurs Reed Muller », Université Abou Bakr Belkaid– Tlemcen, Algerie.
[2] A. Spătaru, 1987, « Fondements de la théorie de la transmission de l'information », PPUR,
[3] B. Martin, 2004, « Codage, cryptologie et applications », PPUR,
[4] Jessie MacWilliams et Neil Sloane, 1977, « The Theory of Error-Correcting Codes», North-Holland,
[5] David Kahn (trad. Pierre Baud, Joseph Jedrusek), 1980, « La guerre des codes secrets », InterEditions,
[6] Simon Singh (trad. Catherine Coqueret), 2001, « Histoire des codes secrets », Librairie Générale Française (LFG),
[7] Jacques Stern, 1998, « La science du secret », Odile Jacob, coll.