DURAND-LALOY
Rapport des TP1 parties A et B et TP2 de Codage

TP1 partie A

Simulation de plusieurs utilisateurs communiquant sur un même signal avec encodage et décodage du signal.

Programme exécutable : **./prog <nombre utilisateurs> <verbose>**

- **nombre utilisateurs :** le nombre d'utilisateurs qui vont communiquer simultanément. Ce nombre doit être positif et non nul.
- **verbose :** 0 ou 1, affiche les opérations intermédiaires. 0 pour ne pas afficher.

Le déroulement du programme se fait en 3 étapes :

1. Génération du signal
2. Transmission
3. Décodage du signal.

Génération du signal

Génération du signal associé aux n utilisateurs de la simulation.

1. Une matrice de Hadamard est générée suivant le nombre d'utilisateurs. La taille de la matrice est la puissance de 2 supérieur ou égale au nombre d'utilisateurs.

Exemple :

Pour 2 utilisateurs, la matrice sera de taille 2.
Pour 3 et 4, elle sera de taille 4…

2. Chaque utilisateur est associé à une ligne de la matrice, c'est ce que l'on a appelé un canal. Les n utilisateurs sont associés aux n premiers canaux de la matrice.

3. Génère le signal de l'utilisateur. Le message de chaque utilisateur est prédéfini à « Je suis l'utilisateur <n> ! ». Chaque bit du message est associé à un code d'étalement généré suivant le canal utilisé. Le tout est alors stocké dans un tableau d'éléments binaires : c'est le signal émis par l'utilisateur.

4. La dernière étape consiste à regrouper les différents signaux en un seul : une simple somme bit à bit des signaux.

Transmission

Transmission du signal en vu du décodage.

Nous utilisons ici un canal dit « idéal ». Il s'agit d'une fonction qui ne fait aucune opération sur le signal. Elle se contente de renvoie le signal qu'elle a reçu.

Décodage du signal

Décodage du signal reçu.

1. Pour pouvoir décoder ce signal, il faut générer la matrice de Hadamard et pour cela il faut connaître le nombre d'utilisateur. La première étape consiste donc à chercher le maximum (ou le minimum du signal), indiquant alors le nombre d'utilisateurs sur ce signal. On peut alors générer la matrice de Hadamard.

2. Décode le signal avec chaque canal possible. Il s'agit ici de faire l'opération inverse de l'étalement et de retrouver chaque bit du signal original de chaque utilisateur.

2.1. Il se peut que le signal reçu est était modifié lors de la transmission (canal non idéal), alors, lors du décodage de chaque bit, on ne retrouve pas les valeurs 0 ou 1 mais une valeur entre deux. Il faut donc être capable de corriger cette erreur, c'est ce qui est fait grâce au niveau de décision.

3. Le signal de chaque utilisateur (tableau d'éléments binaires) et convertis en une chaîne de caractères. Si tout se passe correctement, on retrouve les messages « Je suis l'utilisateur <n> ! » pour le n premiers canaux. Les autres canaux, non utilisés, retourne une chaîne vide.

Point sur les structures de données utilisées

Deux structures de données sont principalement utilisées : les matrices et les vecteurs.

La structure de type matrice (*struct matrice*, *ptrMatrice_t* : pointeur sur *struct matrice*) consiste en un tableau de *char* bi-dimensionnel carré dont la taille (largeur, hauteur) sont connus et fixe. Elle est utilisée pour représenter les matrices de Hadamard. Les valeur de la matrice sont de type *char* mais sont interprétées comme des entiers.

La structure de type vecteur (*struct vecteur*, *ptrVecteur_t* : pointeur sur *struct vecteur*) consiste en un tableau de *char* de taille connue et fixe. Elle est utilisée pour stocker les signaux ou plus généralement les tableau d'éléments binaires (signaux de chaque utilisateur, lors de l'étalement...). Ici aussi les valeurs du vecteur sont de type *char* mais sont interprétées comme des entiers.

TP1 partie B

Mise en pratique d'un codeur / décodeur HDBn et d'un codeur / décodeur Arithmétique.

Programme exécutable : **./prog**

Le déroulement du programme se fait en 2 grandes parties :

1. Le codage / décodage arithmétique
2. Le codage / décodage HDBn

Pour gérer l'ensemble des signaux, nous avons utilisé un switch case allant de 1 à 4 permettant de sélectionner le codage arithmétique, HDB2, HDB3 et HDB4.
Le traitement effectué par défaut est HDB3.

Codage / décodage arithmétique

Point sur les structures de données utilisées

Nous avons utilisé trois structures nous permettant de stocker les données dont nous avions besoins.

- La structure inventaire, elle contient le nombre de lettres différentes, le nombre de lettres au total ainsi qu'un tableau qui associe à chaque lettres son nombre d'apparition.
- La structure lettreIntervalle, elle contient le nombre de lettres différentes et un tableau contenant la borne inférieure et supérieure pour chaque caractère.
- La structure messageIntervalle, elle contient la borne inférieure et supérieure du message codé.

Explication des grandes lignes du code

Tout d'abord, on initialise la structure inventaire avec la chaîne de caractère que l'on traiter. Puis grâce à cette dernière, on peut initialiser la structure lettreIntervalle qui nous permettra d'appliquer le codage arithmétique.

Puis en connaissant l'intervalle de chaque lettre, on prend l'intervalle du premier caractère de la chaîne de caractère.
Pour ensuite calculer la nouvelle borne inférieure et supérieure, on prend le caractère suivant, et on applique le calcul suivant pour trouver la nouvelle borne inférieure :

(ABI + (ABS - ABI) * BIA)

et le calcul suivant pour trouver la nouvelle borne supérieure :

(ABI + (ABS - ABI) * BSA).

ABI étant l'ancienne borne inférieure.
ABS étant l'ancienne borne supérieure.
BIA étant la borne inférieure actuelle.
BSA étant la borne supérieure actuelle.

Codage / décodage HDBn

Point sur les structures de données utilisées

Nous avons utilisé une seule structure.

- La structure hdbn, elle contient la valeur du dernier viol et celle du dernier "1", le message codé positif et négatif ainsi que la valeur de "n".

Explication des grandes lignes du code

Tout d'abord, on initialise la valeur de "n" dans la structure par la valeur choisit par l'utilisateur. Puis on met par défaut la valeur du dernier viol et du dernier "1" à -1.
On parcourt tout le message non codé, on regarde les n+1 valeurs, si il s'agit d'une suite de zéro, on effectue un certain traitement, sinon on regarde la 1ère valeur parmi les n+1 et si c'est un 1 on le remplace par l'inverse du dernier "1", et si c'est un zéro, on réécrit la valeur zéro.

Le traitement cité précédemment est le traitement HDBn, qui prend en compte la valeur du dernier viol et la valeur du dernier "1" et qui remplace la suite de zéro par la valeur suivant :(x n-2 y).
x étant 0 / 1 / -1 suivant le dernier "1".
y étant 1 / -1 suivant le dernier viol.
n-2 étant le nombre de zéro.

A la fin, on obtient un message codé ternaire. Or on souhaite manipuler du binaire, c'est pourquoi on modifie le message codé ternaire en 2 messages codés binaires. Pour ce faire, nous parcourons le message codé et lorsque l'on rencontre un :

- "1", on note un "1" pour le message codé positif et un "0" pour le message codé négatif.
- "-1", on note un "1" pour le message codé négatif et un "0" pour le message codé positif.
- "0", on note un "0" pour le message codé positif et un "0" pour le message codé négatif.

Pour ce qui est du décodage, nous devons d'abord réunifier les deux messages codés binaires, donc de la manière inverse de celle citée précédemment. On met la valeur du dernier "1" à -1 par défaut.
Puis on parcourt le message, lorsque l'on tombe sur un :

- "0", on recopie la valeur "0".

- "1" ou "-1", on regarde si la polarité est la même avec celle du dernier "1", si c'est le cas, on met un "0" à la valeur actuelle ainsi qu'à la valeur actuelle-n, sinon on met un "1" à la valeur actuelle.

On retrouve à la fin le message non codé.

TP2

Implémentation de générateurs de nombres pseudo-aléatoires. Mise en place des codeurs à longueur maximale, Gold et JPL.

Programme exécutable : **./prog**

Après que l'utilisateur ait saisie le nombre de nombre pseudo-aléatoire qu'il veut générer, le déroulement du programme se fait en 3 étapes :

1. Génération de n nombres par codeur à longueur maximale.
2. Génération de n nombres par Gold.
3. Génération de n nombres par JPL.

n étant ce qu'a saisie l'utilisateur au début.

Codeur à longueur maximale

La génération de nombre pseudo-aléatoire avec le codeur à longueur maximale se fait en 3 étapes :

1. Initialisation, le codeur à longueur maximale est composé d'un polynôme générateur et de registres. Pour créer un codeur LM il faut donc lui donner un polynôme générateur et une suite d'initialisation. Ici le polynôme utilisé est toujours le même ce qui fait qu'il générera toujours la même suite de nombre à chaque lancement du programme. Le polynôme choisis est [16, 14, 13, 11] et les registres sont initialisés à 1.

2. Génération, retourne un vecteur contenant la séquence binaire générée, la longueur de la séquence doit être donnée. La valeur de sortie des registres est celle du dernier. Les registres sont ensuite décalé. Entre alors une nouvelle valeur qui est le XOR des registres décrit dans le polynôme générateur. L'opération est répétée jusqu'à obtenir une séquence de la longueur voulue.

3. Conversion, la séquence générée est convertis en entier. Chaque élément binaire qui compose la séquence sont 'recoller' et stocker dans une variable.

Codeur de Gold

La génération des nombres pseudo-aléatoires avec codeur de Gold se déroule de la même façon que pour le codeur à longueur maximale : initialisation, génération et conversion.

1. Il s'agit ici d'initialiser les codeurs LM qui composent les 2 codeurs de Gold. Le polynôme générateur des 2 codeurs LM est fixe ([16, 14, 13, 11]), cependant ils ne sont pas initialisés avec la même séquence. Là aussi les séquences sont fixes et sont pour le premier une suite de '1' et '101' pour l'autre. Les codeurs LM ne générerons alors pas les même séquences en même temps.

2. Là encore, on génère une séquence de taille donnée. Cette fois-ci les 2 codeurs LM vont générer chacun une séquence. Puis on effectue un XOR entre chaque éléments des 2 séquences. Le résultat est alors la séquence générée par le codeur de Gold.

3. Il s'agit de la même opération de conversion que pour le codeur LM.

Codeur JPL

Cette fois-ci, le fonctionnement est identique au codeur de Gold à la différence près qu'il y a un nombre variable de codeur LM. Dans le programme, ils sont au nombre de 3 et sont : [2,1] [3, 1] et [5, 1]. Chaque les registres de chaque codeur sont initialisé à la valeur 1.
Lors de la génération (est toujours précisé la taille de la séquence à générer), chaque codeur LM va générer sa séquence (de la taille donnée). L'opération XOR est ensuite opérée entre chaque éléments des séquences. Les résultat est alors la séquence générée par le codeur JPL.

Point sur les structures de données utilisées

Quatre structures de données sont principalement utilisées : vecteur, codeur à longueur maximale, codeur de Gold et code JPL.

La structure vecteur (*struct vecteur*, *ptrVecteur_t* pointeur sur *struct vecteur*) consiste en un tableau d'entier dont la taille est connue et fixe. Elle est principalement utilisée dans le codeur à longueur maximale.

La structure codeur à longueur maximale (*struct codeLongueurMax* et sont pointeur *ptrCodeLongMax*) consiste en 2 vecteurs, l'un contenant les registres du codeur, l'autre le polynome (pour le polynome [5,4,3], le vecteur contient les valeurs 5, 4, 3). Le codeur à longueur maximale est utilisé dans les codeurs de Gold et JPL.

La structure codeur de Gold (*struct gold*) consiste en 2 codeurs à longueur maximale.

La structure codeur JPL (*struct jpl*) consiste en un tableau dont le nombre d'élément stocké en connu. Chaque champs de ce tableau comporte un codeur à longueur maximale et un vecteur.