Codage de l'Information
Compte rendu de TP

Aurélien TUDORET, Axel JOURRY

Novembre 2020

1 Code d'étalement

Afin de parvenir à écrire le code d'étalement nous avons fait le choix de diviser la tache en quatre parties. La première partie à consisté en l'écriture des fonctions relatives a la création de la matrice de Hadamard. La seconde à été de coder le message en utilisant la matrice. Puis il a fallut créer les fonctions relatives à l'étalement des messages envoyés par les utilisateurs. Enfin, la dernière partie à été de créer les fonction relatives au désétalement du message étalé reçu.

1.1 Matrice de Hadamard

Afin de créer la matrice de Hadamard, nous avons eu besoin de 2 fonctions. La première permet de créer cette matrice et la seconde permet de l'afficher.

Pour créer la matrice, nous initialisons celle ci a l'aide du nombre d'utilisateurs envoyant des messages, c'est a dire : "Matrice[User][User]", ce qui permet de ne pas avoir un tableau qui serait plus grand que ce dont nous avons besoin. Ensuite ce tableau est passé en paramètre de la fonction de création, avec en second argument le nombre d'utilisateurs. La fonction consiste en deux boucles for imbriquées dans une boucle while afin de pouvoir parcourir chaque case de la matrice et ce en fonction du nombre d'utilisateurs (correspondant à la boucle while).

Ensuite, la fonction de lecture ne consiste qu'en deux boucles for imbriquées permettant le parcours entier de la matrice pour l'afficher.

1.2 Étalement

Afin d'effectuer l'étalement des messages saisis par les utilisateurs, nous nous sommes servis de 5 fonctions La première n'est autre que la fonction "allocation tab message" permettant d'allouer à une matrice le nombre d'utilisateurs et pour chacun, la longueur du message une fois que celui ci sera codé.

La seconde, "mise a zéro" initialise l'entièreté du tableau précédent à 0 afin de pouvoir effectuer des additions dessus.

La troisième, "codage utilisateur", permet d'appeler la fonction "codage" pour chacun des utilisateurs.

Cette même fonction codage, elle, réalise pour chaque bit du message utilisateur le codage de Hadamard.

Enfin, une fonction "affichage codage" parcours toute la matrice afin de l'afficher.

Une fois le message codé, la dernière fonction "étalage" utilisera la formule d'étalage afin de transformer tous les messages utilisateurs en un seul et unique message prêt à être envoyé.

1.3 Désétalement

Le désétalement fonctionne avec deux fonctions.

La première effectue le désétalement de la chaîne étalée précédemment et la seconde l'affiche.

2 Codeur HDBn / Codeur Arithmétique

Ces deux codages partagent une même fonction afin d'être effectué, il est nécessaire de préciser dans les paramètres de cette fonction le type de codage souhaité.

2.1 Codeur / Décodeur HDBn

Afin de mettre en place le codeur/décodeur HDBn, il a été mit en place deux structures

La première, hdbnIn, va servir a accueillir le message initial, alors qu'une seconde instance de cette structure récupérera le message une fois qu'il aura été décodé à la fin des traitements (variable message). La dernière, de type hdbnOut, permet d'accueillir le message une fois qu'il aura été codé avec le Pulse+ (dans la variable P) et le Pulse- (dans la variable N).

Ces deux structures contiennent aussi un dernier champ taille correspondant a la taille du message.

2.1.1 Codeur HDBn

Le codeur HDBn va être composé d'une boucle for générale qui parcours toute la chaîne. A chaque fois que le parcours trouvera un bit 1, il le stockera a l'emplacement prévu alternativement dans les variables P et N. Pour garder une trace de la bonne chaîne, on utilisera une variable "dernier choix". Si on trouve un bit à 0, alors on regarde si les N bits suivants sont eux aussi à 0 (le n correspond à la version de HDBn choisi). Si on trouve un bit a 1 dans ces bits suivants, alors on écrit 0 dans les deux variables P et N. En revanche, si les n bits suivants sont à 0, alors on place le bit nMax a 1 dans la même chaîne que la dernière utilisée. Si cette action a déjà été effectuée une fois, alors en plus de faire cette action, on placera a n=0 un bit 1 dans la chaîne alternée par rapport a "dernier choix".

Le tout donnera les deux chaînes codées du message.

2.1.2 Décodeur HDBn

Le décodeur HDBn est composé de trois boucles for.

La première va utiliser la formule P-N et placer le résultat dans la structure qui récupère le message décodé. Cela va permettre de transformer les deux messages codés en un seul, composé de 1, 0 et -1.

Le but de la seconde boucle va être de repérer dans cette nouvelle chaîne les alternances de -1 et de 1. Si il y avait un manque d'alternance (violation pendant le codage), alors les n bits précédents sont placés à 0 afin d'annuler la violation.

Une fois cette action terminée, il ne reste plus qu'à parcourir une dernière fois la chaîne afin de remplacer toutes les occurrences de -1 par 1 et nous retrouvons la chaîne envoyée au début du programme.

2.2 Codeur / Décodeur Arithmétique

Pour encoder un message avec le codeur arithmétique, il est nécessaire de passer à la fonction "codeur" une structure, arithIn, à deux champs. Le premier est un tableau de caractères, et le second, la taille du message dans ce tableau. Le message une fois codé, sera stocké dans la

structure arithOut, contenant un champs F représentant le message codé, ainsi qu'un structure nommée matrice, qui est composé du tableau des fréquences des lettres dans le message originel.

2.2.1 Codeur Arithmétique

Il est nécessaire dans un premier temps de rechercher les occurrences de chaque lettre composant le message. Une fois ce tableau d'occurrences rempli, il faut établir pour chacune des lettres un intervalle représentant sa fréquence dans le message. Tous les intervalles doivent tenir entre 0 et 1. L'intervalle attribué à une lettre correspond est donné selon sa première apparition et non l'ordre alphabétique. Une fois ces deux actions effectuées, on entame la dernière phase du codage, le calcul de F qui représentera le message codé. Pour se faire, on prend pour base l'intervalle le plus proche de 0, et on remonte jusqu'au plus proche de 1.

2.2.2 Décodeur Arithmétique

Décoder le message est encore plus simple, il suffit pour chaque tour de boucle, de regarder à quel intervalle F peut être associé. Ainsi, une lettre E, ayant un intervalle de [0.5;0.75[et un F égal à 0.5236 permettent de définir que la lettre E est le caractère que nous recherchons. Suite à cela, F se voit attribué une nouvelle valeur calculée grâce aux bornes de la lettre trouvée.

Lors de la conception de cette fonction nous nous très rapidement aperçu que la précision des floats de nos systèmes d'exploitations est très rapidement dépassée par les résultats des calculs des bornes. Ainsi, si une chaîne est trop longue, ou qu'elle contient trop de lettres différentes, il est très probable que le programme ne soit pas capable de retrouver la bonne lettre, ce qui peut donner lieu à de grosses erreurs de décodage.

3 Générateur Pseudo Aléatoire

Dans le but de réaliser les codeurs de Gold et JPL, nous avons décidés de les mettre en place à l'aide de deux mains différents, soit deux programmes différents. Nous utilisons une structure nommée "sequence", elle contient toutes les informations nécessaires à la création et l'utilisation d'une séquence. On y trouve notamment la séquence de l'utilisateur sous forme d'un tableau d'entier, ainsi qu'une variable nommée "taille" qui représente la taille de ce tableau. Il y également une variable etage qui correspond à la longueur d'une ligne et etageMax au nombre de lignes.

3.1 Codeur de Gold

Afin de pouvoir délivrer une séquence de Gold, nous commençons par initialiser deux matrices, une par séquence, puis nous les remplissons grâce au codage à longueur maximale. Si les tailles (nombre de colonnes par ligne) sont identiques pour chaque séquence, nous pouvons alors commencer le codage de la séquence Gold. Pour ce faire, nous prenons la dernière colonne de chacune des deux matrices de séquence, puis pour chacune des valeurs, nous effectuons un XOR dont le résultat est un des éléments de la séquence de Gold. Si l'utilisateur demande une séquence finale plus longue que le résultat, nous n'allongeons pas le résultats pour qu'il reste cohérent. En revanche, si la séquence demandée est plus courte que le résultat, nous ne conservons que les valeurs demandées et ne nous préoccupons pas du reste.

3.2 Codeur JPL

Pour pouvoir effectuer le codage JPL de séquences, nous effectuons le même fonctionnement que pour le codage JPL a quelques nuances près.

La première nuance est que l'on peut avoir autant de séquences que nous le voulons. La seconde nuance requiert que les nombre d'étages pour chaque séquences soit premiers entre eux. Cela implique que chaque séquence aura une longueur unique, a contrario du codeur de Gold qui impliqué d'avoir les mêmes longueurs pour les deux séquences.

4 Agent Cycle Life

Nous n'avons eu le temps que de commencer ce dernier TP. En conséquences, nous expliquerons dans cette section de quelle manière nous l'aurions traité. Nous avons envisagé l'utilisation des threads afin de pouvoir traiter parallèlement plusieurs agents.

4.1 Traitement du TP

Afin de traiter ce TP3, nous avons envisagé de découper notre code en plusieurs fonctions clefs pour gérer ce programme. Nous réutiliserions les structures et fonctions déjà implémentées lors des TP précédents pour les différents codages, simplement en les incluant dans les fichiers et à la compilation.

Dès que cela s'avérerait nécessaire, nous créerions un nouvel agent via la fonction "create_thread", cet agent sera initialisé dans la foulée par la fonction "init_thread" et passé à un état "initialisé". Une fois ces étapes accomplies, l'agent pourra alors passer en état "actif" grâce à la fonction "activate_thread", invoquant ce nouvel agent. Fonction qui permettra de lancer le thread avec les paramètres nécessaires pour exécuter la fonction souhaitée avec les paramètres nécessaires à cette fonction.

Afin de permettre la mise en pause, la suspension, ou l'arrêt des threads, nous serions obligés d'ajouter la gestion des signaux (interruptions) à nos fonctions des TP précédents.

Ainsi, au signal SIGUSR1, le thread serait placé dans une état suspendu, via la fonction "sleep" de "unistd.h" et notre fonction "suspend", l'envoi du signal SIGUSR2 de part la fonction "resume" sortant l'agent de l'état suspendu.

Les signaux SIGINT et SIGBUS, permettront de passer l'agent dans l'état d'attente grâces aux fonctions correspondantes "wait" et "wait_up".

Au signal SIGQUIT le thread serait simplement détruit, ce qui correspond au traitement de la fonction "destroy".Au signal SIGTERM correspond la fonction "quit", demandant à l'agent de se terminer.

Une fois le cycle de vie de l'agent terminé, le résultat de son action est retournée au programme principal et l'agent est désalloué afin sauver des ressources.