**By Chems Eddine Lahbabi et Vincent-Nam Dang**

**Projet d’informatique**

**Cryptographie et Compression**

L2 PCP Option Informatique - Paul Sabatier III

Définition du projet de base:

Le projet de base consiste à prendre l'entrée dans le terminal, la stocker dans un tableau de caractère, convertir ce tableau de caractère en tableau de décimal base 10 pour ensuite transformer ce nouveau tableau de décimal en tableau de binaire, puis coder le binaire dans un autre tableau.

Ainsi en prenant une suite de caractère en entrée, on pourra coder, grâce à un code choisi au préalable, cette entrée et d’afficher sur le terminal cette entrée codée.

Grâce à l'implémentation de l’extension entrées/sorties, il sera possible pour l’utilisateur de pouvoir choisir entre une entrée dans le terminal ou une entrée se trouvant dans un fichier. Cette entrée sera lue par le programme et codée avec le programme. L’utilisateur pourra aussi choisir d’afficher la sortie du programme sur le terminal ou de l’écrire dans un fichier.  
De plus, le code choisi pour coder cette entrée pourra être lu dans un fichier grâce à l’implémentation de l’extension de lecture du code dans un fichier. Il s’agira, comme pour la lecture de l’entrée dans un fichier, de lire un code dans un fichier et de s’en servir.

Nous avons choisi d’implémenter l’extension du codage de Huffman. Cette extension consiste à reprendre la sortie du programme de base comme entrée et de créer un arbre permettant de créer un nouveau code, basé sur la fréquence de chacun des caractères, et de le coder à nouveau. Le code sera fait de tel que les caractères seront associés à un code d’une taille qui est en fonction de la fréquence apparition du caractère permettant d’avoir un codage non ambiguë le plus court possible permettant une compression de la chaîne de caractère de telle sorte qu’il n’y ai pas de perte d’information.

**Le programme de base** se basera sur plusieurs fonction pour être opérationnel:

#### FONCTION LECTURE ENTRÉE :

Tout d’abord, il faudra pour le programme pouvoir lire une entrée. nous allons implémenter une fonction qui lit caractère par caractère l’entrée et rentre chaque caractère dans un tableau qui nous permettra de pouvoir travailler dessus. L’entrée pouvant être de taille variable, il est donc nécessaire d’allouer la mémoire du tableau qui contiendra l’entrée au fur et à mesure de la lecture de l’entrée. Nous avons choisi de faire une boucle qui lit chaque caractère jusqu’à l’apparition d’un passage à la ligne par l’utilisateur et alloue la mémoire nécessaire pour un caractère avant de l’y rentrer dedans.

#### FONCTION ALLOCATION DE MÉMOIRE:

Pour allouer cette mémoire, nous allons créer une fonction qui permettra d’ajouter une “case” dans le tableau de la taille d’un caractère. En effet, cette fonction ajoute de la mémoire de la taille d’une fois la taille du type des données rentrées dans le tableau pris en paramètre. Elle renvoie un tableau.

De plus, nous aurons besoin en plus d’une fonction d’allocation de mémoire qui prendra en paramètre une structure et lui allouera la mémoire nécessaire.  
Enfin il nous faudra aussi une fonction d’allocation de mémoire qui prendra en paramètre un tableau et une taille de tableau en entier et qui lui allouera la mémoire nécessaire pour qu’il ait la taille voulu. La fonction renverra un tableau.

#### FONCTION BINAIRE:

Une fois l’entrée lue, une fonction permettra de transformer chaque caractère dans sa valeur binaire. En effet, le tableau créé après la lecture sera repris et un nouveau tableau sera créé contenant 8 fois plus de cases que le tableau précédent. Ainsi, il sera possible de conserver le code binaire de chacun des caractère. Pour cela, la fonction prendra la valeur entière de chaque caractère. Une boucle servira à prendre un caractère et une boucle à l’intérieur de la première permettra de remplir le tableau de valeurs binaires. Ainsi, pour chaque caractère, il sera soustrait si possible la valeur de chaque bit jusqu’à atteindre 0. Ainsi ce tableau contiendra 8 fois le nombre de caractères en 0 et en 1. La fonction renvoie ce dernier tableau.

#### FONCTION CODAGE

Il faudra, pour la fonction de codage, prendre le tableau des caractères en binaire et le code en paramètre et associer à chaque groupe d’entier (de 0 ou 1) un nouveau caractère défini par notre code. Ainsi, il s’agira de prendre un tableau et de le remplir dans une boucle qui vérifie s’il existe un code associé au groupe de caractères lu dans un ordre croissant de taille de groupe et remplira le tableau. Cette fonction renvoie ce nouveau tableau.

#### FONCTION D’AFFICHAGE

Enfin, il faudra afficher ce message codé. Pour cela sera pris en paramètre par une nouvelle fonction le tableau renvoyé par la fonction codage et affichera grâce à une boucle chaque caractère dans le terminal.

### EXTENSION : ENTRÉES SORTIES:

Cette extension a pour but de permettre à l’utilisateur de choisir s’il veut utiliser comme entrée un fichier ou de la rentrer directement dans le terminal et de choisir s’il veut avoir la sortie du programme dans un fichier ou directement dans le terminal.

#### FONCTION LECTURE FICHIER

Une fonction qui permettra d’ouvrir et de lire un fichier sera utilisée. Cette fonction est analogue à la fonction de lecture d’entrée à la seule différence qu’elle prendra en paramètre une adresse fichier existant, l’ouvrira et en lira le contenu de la même manière que la fonction de lecture d’entrée du programme de base dans ce fichier. Une fois chaque caractère rentré dans le tableau associé, la fonction s’assurera de la bonne fermeture du fichier. La fonction renverra ce tableau.

#### FONCTION ÉCRITURE FICHIER

En prenant le tableau contenant l’entrée codée, cette fonction s’assurera de l’ouverture d’un fichier, prendra chaque caractère du tableau dans une boucle en l’écrivant dans ce fichier et fermera ce fichier. La sortie pourra être lue dans ce fichier.

### EXTENSION LECTURE CODE DANS FICHIER

Cette extension consiste à lire le code dans un fichier. Il faudra une fonction analogue à celle de lecture d’entrée dans un fichier. En effet, elle prendra en paramètre une structure de liste, ouvrira un fichier pour lire le code, le lira et rentrera chaque caractère et son code dans une liste. La fonction renverra cette liste. Il faudra pour se code qu’il respecte un format / une syntaxe précis(e).

Il faudra modifier la fonction de codage afin qu’elle prenne en paramètre un code et qu’elle l’utilise à la place du codage du programme de base.

### EXTENSION CODAGE DE HUFFMAN:

Cette extension nécessite le tableau de sortie du programme de base. Il faudra trier ce tableau selon les caractères et créera 2 nouveau tableau. Un tableau contiendra un caractère et un autre contiendra, dans la case d’indice identique au caractère, sa fréquence. Une fois ces informations récupérées, il faudra créer un arbre qui rangera chacun des caractères par ordre décroissant des fréquence. Le code de Huffman sera trouvé en fonction du nombre de branche parcouru pour atteindre le caractère et sera conservé dans un fichier afin de pouvoir décrypter le message de sortie du programme (et donc de pouvoir le décompresser). Une fois ce code récupéré, il faudra coder avec ce nouveau code le message. Enfin, ce nouveau message sera renvoyé soit en sortie sur le terminal soit dans un fichier grâce à l’extension d’entrées-sorties.

#### FONCTION TRIAGE ET ÉCHANTILLONAGE :

Cette fonction prend le tableau de sortie de programme de base, c’est à dire déjà codé, trie dans un nouveau tableau cette sortie avec chaque caractère identiques à la suite. Il sera donc plus simple de trouver les fréquences de chaques caractère. En effet, 2 nouveaux tableaux seront créés qui comprendront chaque caractères différents dans l’un et, dans l’autre, dans la case du tableau d’indice identiques à un caractère, sa fréquence grâce à une boucle qui comptera le nombre fois que le caractère est apparu dans le tableau trié. La fonction renverra une structure.

#### FONCTION CRÉATION DE L’ARBRE DE HUFFMAN:

Cette fonction prend la structure renvoyé par la fonction précédente. La fonction utilisera ces tableaux afin de créer un arbre. Chaque noeud de cet arbre comprendra 4 valeurs qui sont un caractère, sa fréquence et 2 fils, un gauche un droit. Chaque caractère sera rangé de cette façon :

-Le caractère dont la fréquence est la plus grande sera pris dans le tableau.

-La fonction ira dans le premier fils libre du premier fils droit.

-Ce caractère et sa fréquence seront mis dedans.

La fonction renverra cet arbre et la taille total de l’arbre.

#### FONCTION RÉCUPÉRATION DU CODE:

Le code de Huffman sera récupéré de la manière suivante :

La fonction gardera le nombre de fils gauches et fils droits qui seront rencontrés jusqu’à l’obtention du caractère dans l’arbre. Ainsi, si 2 fils droits sont rencontrés, la fonction créera un mot comptant le nombre total de fils gauches rencontrés en 0 puis deux 1. Si 1 seul fils droit est rencontré, la fonction créera un mot comprenant le nombre de fils gauches moins un , puis un 1 et fini par 0.

Chaque mot créé sera mis dans une liste dans l’ordre des tailles des codes avec chaque caractère associé. La fonction renverra une liste.

#### FONCTION CODAGE :

La fonction utilisera la fonction de codage du programme de base avec comme paramètre le code de Huffman.

### VARIABLES UTILISES:

Nous utilisons dans ce programme plusieurs types de variables :

Nous utiliserons des structures. Ces structures contiennent soit des tableaux, soit des entiers, soit des caractères.

Nous utilisons différentes structures contenant:

- 3 tableau contenant l’entrée lue dans un, la valeur en binaire de chacun des caractères de l’entrée dans un autre et l’entrée codée dans un dernier ainsi que la valeur de la taille du tableau de l’entrée lue.

Ensuite, nous aurons besoin d’une structure d’arbre pour la réalisation de l’extension de Huffman qui comprendra un caractère, un entier pour les fréquences et l’adresse des 2 fils. Enfin il nous faudra une structure de liste pour notre code. Chacune de ces structures auront la mémoire allouée nécessaire avant leur utilisation.

### LES OPTIONS:

L’utilisateur aura le choix pour une entrée rentrée soit dans le terminal soit dans un fichier, pour le code à utiliser pour le programme de base dans un fichier et pour la sortie qui sera soit dans le terminal soit dans un fichier.

### LES MODULES:

Nous avons décidé de diviser nos fonctions en 4 modules distincts.

Un premier module sera consacré aux fonctions d’allocation de mémoire avec une fonction d’allocation de mémoire pour les structures, une fonction d’allocation de mémoire fixe pour un tableau et une fonction d’ajout de mémoire pour les tableaux.

Un second module sera consacré aux fonctions permettant la transformation de données avec la fonction permettant de transformer l’entrée de caractère en binaire et la fonction de codage.

Un troisième sera créé pour les fonctions associés à l’entrée et à la sortie avec une fonction de lecture d’entrée, une fonction de lecture d’entrée dans un fichier, une fonction d’affichage dans le terminal, une fonction d’écriture de la sortie dans un fichier.

Le dernier sera créé pour l’extension de Huffman avec une fonction pour le trie et les fréquences des caractères, une fonction pour la création de l’arbre et une fonction pour la récupération du code à partir de l’arbre.

Les 3 derniers modules dépendent de celui pour les fonction associées à l’allocation de mémoire.

LES FONCTIONS DANS LES MODULES :

#### MODULE MÉMOIRE:

Une fonction d’allocation de mémoire d’une structure qui prend en paramètre une structure et qui renvoie une structure.

Une fonction d’ajout de mémoire à un tableau qui prend en paramètre une structure et qui renvoie une structure.

Une fonction d’allocation de mémoire d’un tableau qui prendra un tableau et sa taille en paramètre et renverra un tableau.

#### MODULE TRANSFORMATIONS :

Une fonction de transformation en binaire qui prend une structure en paramètre et renvoie une structure. Elle dépendra de la fonction d’allocation de mémoire d’un tableau.

Une fonction de codage qui prend une structure et qui renvoie une structure. Elle dépendra des fonctions d’allocation de mémoire.

#### MODULES ENTRÉES-SORTIES :

Une fonction de lecture d’entrée qui ne prend pas de paramètre et renvoie une structure. Elle dépend des fonctions d’allocation de mémoire.

Une fonction de lecture d’entrée dans un fichier qui prend comme paramètre l’adresse d’un fichier et renvoie une structure. Elle dépend des fonctions d’allocation de mémoire.

Une fonction d’affichage dans le terminal qui prend comme paramètre une structure et qui ne renverra rien. (Elle affichera sans rien renvoyer.) Elle dépend des fonctions d’allocation de mémoire.

Une fonction d’écriture dans un fichier qui prendra une adresse de fichier et une structure et qui ne renverra rien. (Elle écrira sans rien renvoyer.)

#### MODULE HUFFMAN :

Une fonction de trie et de récupération de fréquences qui prendre une structure en paramètre et qui renverra une structure. Elle dépendra des fonctions d’allocation de mémoire.

Une fonction de création d’arbre qui prend une structure en paramètre et renvoie une structure (un arbre). Elle dépendra des fonctions d’allocation de mémoire.

Une fonction de récupération du code de Huffman qui prend en paramètre un arbre et renvoie une structure. Elle dépendra des fonctions d’allocation de mémoire.

#### Schéma d’illustration des modules



typedef struct code\_s \*code;

struct code\_s

{

int\* b;

char c;

int sob;

struct code \*suivant;

};

tabs codage(tabs t, const code D, int smin){

int i,j,k,x=0,w=0,e=0;

int\* u;

code C;

for(i=0; i<(t->sot)\*8, i++){

C=D;

u=(int\*)malloc(sizeof(int));

for(j=0,j<smax,j++){

u[j]=t->tb[j+w];

w+=1;

u=(int\*)realloc(sizeof(int));

if (sizeof(u)>=smin){

for(k=0; k < C->sob ; k++){

if ((u[k]==C->b[k])||(sizeof(b)==sizeof(c->b)))

{

x=x+1;

if (x==c->sob)

{

t->ts[e]=C->c;

e=e+1;

free(u);

}

}

else{

if (c->suivant!=NULL){

c=c->suivant;

k=0;

}

else{

printf("Le code n'est pas valable.");

return NULL;

}

}

}

}

}

return t;

}