

Compte-Rendu TP1

MALOD Victor - LANQUETIN Alexis - BLANQUET Antoine

10/02/20

Table des matières

1	README.txt	2
2	Codage et Entropie	3
2.1	Entropie de X	3
2.2	Taille d'un code à longueur fixe	3
2.3	Comparaison avec l'optimum	3
2.4	entropie.c	3
2.5	Test du programme sur X	3
2.6	Test du programme sur X'	3
3	Code de Huffman	4
3.1	Arbre de Huffman	4
3.2	Table de codage découlant de l'arbre	4
3.3	Longueur moyenne du code de Huffman obtenu	4
3.4	Borne inférieure	4
3.5	Surcoût	5
3.6	Taux de compression	5
3.7	Rentabilité de Huffman	5
3.8	huffman.c pour des événements simples	6
3.9	huffman.c avec X'	6
3.10	Un pas vers la borne inférieur	7
3.11	Les événements double	7
3.12	Résultats des événements double	7
3.13	Des événements multiples de taille n	7
3.14	Limites physiques	7
3.15	Complexité algorithmique	8
3.16	Complexité mémoire	8
3.17	Mesures de performances	8
4	Codage Arithmétique	9
4.1	Le codage décodage arithmétique	9
4.2	Étape de programmation	9
4.3	Code de WIKI	9
4.4	Code de KIWI	9
4.5	Code de KIKIWIWI	9
4.6	Etape de programmation	9
4.7	Questions 7, 8, 9 et 10	9

4.8	Si n est plus petit que nécessaire	10
4.9	Si n est plus grand que nécessaire	10
4.10	Si n n'est pas fourni	10
4.11	Optimisation de l'algorithme	10

1 README.txt

Avant de vous lancer dans la lecture de ce Compte-Rendu de TP, veuillez noter que pour chaque programme il existe une commande make associée dans le répertoire courant où vous avez extrait l'archive, ce dossier contiendra un fichier README.txt résumant de façon précise chaque chose à savoir pour exécuter les programmes codés en C.

Bonne lecture.

2 Codage et Entropie

2.1 Entropie de X

L'entropie de $X = \{e1, e2, e3, e4, e5, e6\}$ est noté $H(X)$ et vaut 2.32278844363.

2.2 Taille d'un code à longueur fixe

La taille d'un code à longueur fixe est défini par $\lceil \log_2 m \rceil$, avec m le nombre de symboles de l'alphabet. Ici $\lceil \log_2 6 \rceil = 3$.

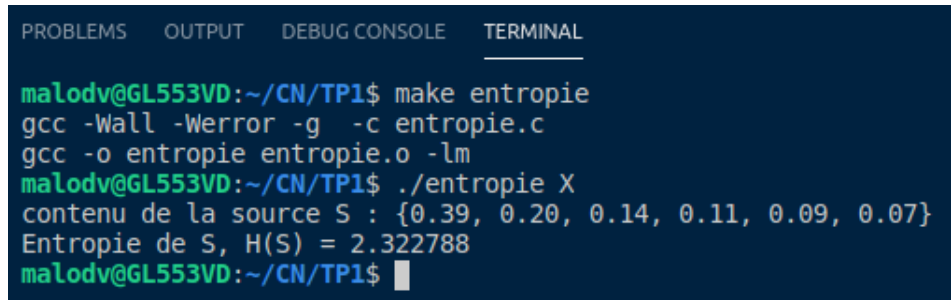
2.3 Comparaison avec l'optimum

Notre optimum valant 2.32, 3 est strictement supérieur, on aura donc des bits inutiles dans certains cas (car $2^3 = 8$, et que nous avons seulement 6 événements à coder), cependant si l'on descend à 2 bits on perd de l'information et nous ne pouvons plus que coder $2^2 = 4$ événements. Dans le cas d'un code à longueur fixe le mieux que l'on puisse faire et de prendre 3 bits, même si l'on s'éloigne beaucoup de l'optimum.

2.4 entropie.c

Le code est disponible dans le fichier source entropie.c, compilable avec `make entropie`, et lançable avec `./entropie <nom du fichier de données>`. Le fichier possède une gestion d'erreurs minimaliste comme demandé.

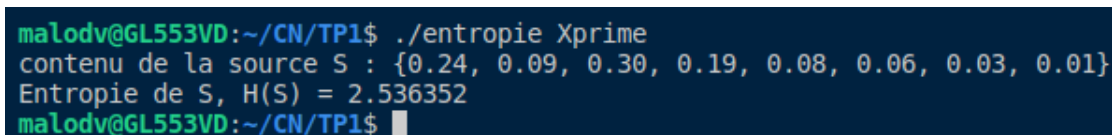
2.5 Test du programme sur X



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
malodv@GL553VD:~/CN/TP1$ make entropie
gcc -Wall -Werror -g -c entropie.c
gcc -o entropie entropie.o -lm
malodv@GL553VD:~/CN/TP1$ ./entropie X
contenu de la source S : {0.39, 0.20, 0.14, 0.11, 0.09, 0.07}
Entropie de S, H(S) = 2.322788
malodv@GL553VD:~/CN/TP1$
```

FIGURE 1 – Entropie de X

2.6 Test du programme sur X'

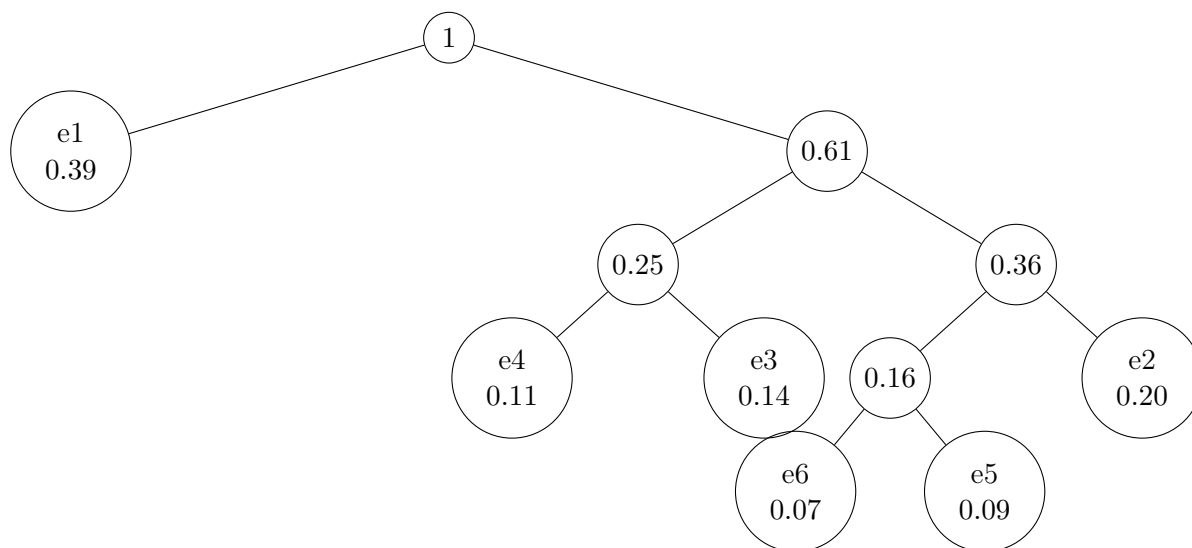


```
malodv@GL553VD:~/CN/TP1$ ./entropie Xprime
contenu de la source S : {0.24, 0.09, 0.30, 0.19, 0.08, 0.06, 0.03, 0.01}
Entropie de S, H(S) = 2.536352
malodv@GL553VD:~/CN/TP1$
```

FIGURE 2 – Entropie de X'

3 Code de Huffman

3.1 Arbre de Huffman



3.2 Table de codage découlant de l'arbre

Événement	Code
e1	0
e2	111
e3	101
e4	100
e5	1101
e6	1100

TABLE 1 – Codes c de l'ensemble X

3.3 Longueur moyenne du code de Huffman obtenu

La longueur moyenne d'un codage " c " pour un événement " e " dans un alphabet X , est donnée par la formule :

$$L(c, X) = \sum p(e) * |c(e)|$$

Ici on obtient, $L(c, X) = 2.38$

3.4 Borne inférieure

La distance à la borne inférieure étant calculée avec $L(c, X) - H(X)$, l'Entropie étant l'optimum, on obtient : $2.38 - 2.32 = 0.06$.

3.5 Surcoût

Pour simplifier le calcul du surcoût en bits de la transmission du dictionnaire des correspondances, on peut simplement sommer la taille des bits de chaque événement, ce qui dans ce cas nous donne un surcoût de **18 bits**.

Après questions posées au professeur lors du TP, nous sommes restés sur cette initiative. Cependant, le receveur du dictionnaire ne pourra pas l'utiliser tel quel, puisque il ne saura pas comment interpréter les données et retrouver les correspondances (pour cela, il faudrait pouvoir transmettre l'arbre avec une convention pour le lire).

Pour conclure, on peut donc dire que la taille du dictionnaire à transmettre est proportionnelle à la somme calculée ci-dessus, mais pas égale.

3.6 Taux de compression

Le taux de compression de ce code par rapport à celui de taille 3 se résume par l'équation suivante :

$$(1 - \frac{L(c,X)}{3}) * 100$$

On obtient donc $(1 - \frac{2.38}{3}) * 100 \approx 21\%$

3.7 Rentabilité de Huffman

Le code de Huffman devient rentable à partir du moment où :

$$t + (L(c, X) * n) < 3 * n$$

avec t la taille du dictionnaire, $L(c, X)$ la longueur moyenne du code de Huffman, et n la longueur de la suite d'événements.

On a donc : $18 + (2.38 * n) < 3 * n \iff 18 < n(3 - 2.38)$, et on trouve $n > 30$, ce qui signifie qu'à partir d'une suite de 30 événements envoyés, un code de Huffman utilisera moins de bits qu'un code de taille fixe durant l'envoi pour notre ensemble X.

3.8 huffman.c pour des événements simples

Le screenshot des questions suivantes ont été réalisés avec la version finale du programme

```
malodv@GL553VD:~/CN/TP1$ huffman X 1
0.200000 [111] e2
0.360000 [11]
0.090000 [1101] e5
0.160000 [110]
0.070000 [1100] e6
0.610000 [1]
0.140000 [101] e3
0.250000 [10]
0.110000 [100] e4
1.000000
0.390000 [0] e1

Contenu de la table de codage :
e2 | 111
e5 | 1101
e6 | 1100
e3 | 101
e4 | 100
e1 | 0

Longueur moyenne du code = 2.380000
malodv@GL553VD:~/CN/TP1$
```

FIGURE 3 – Résultat de huffman.c avec X

Et on peut s'apercevoir qu'on retrouve exactement les résultats ci-dessus. On aurait pu aussi trouver un autre codage car il n'est pas unique mais la méthode de construction à partir des données est la même qu'on a utilisé sur papier et dans l'algorithme.

3.9 huffman.c avec X'

Voici le résultat obtenu avec X' :

```
malodv@GL553VD:~/CN/TP1$ huffman Xprime 1
0.300000 [11] e3
0.570000 [1]
0.090000 [1011] e2
0.170000 [101]
0.080000 [1010] e5
0.270000 [10]
0.060000 [1001] e6
0.100000 [100]
0.030000 [10001] e7
0.040000 [1000]
0.010000 [10000] e8
1.000000
0.240000 [01] e1
0.430000 [0]
0.190000 [00] e4

Contenu de la table de codage :
e3 | 11
e2 | 1011
e5 | 1010
e6 | 1001
e7 | 10001
e8 | 10000
e1 | 01
e4 | 00

Longueur moyenne du code = 2.580000
```

FIGURE 4 – Résultat de huffman.c avec Xprime

3.10 Un pas vers la borne inférieur

Si l'on prend les événements de l'ensemble X deux à deux, on cherche à déterminer $\text{card}(X^2)$ pour le nombre d'événements possibles. Or $\text{card}(X^2)$ est égale à $\text{card}(X)^2$, donc 6^2 , on aura donc 36 événements possibles pour X^2 .

3.11 Les événements double

Voici la liste de ces événements :

$X^2 = \{e2e4, e5e4, e4e5, e2e2, e2e1, e1e2, e6e3, e3e6, e3e3, e5e2, e2e5, e1e1, e5e1, e1e5, e5e5, e6e4, e4e3, e3e4, e6e2, e3e2, e4e6, e6e5, e2e6, e2e3, e6e1, e3e1, e1e3, e1e6, e5e3, e3e5, e4e4, e5e6, e6e6, e4e2, e4e1, e1e4\}$

3.12 Résultats des événements double

On rappelle que $L(c, X) = 2.38$ et $L(c, X') = 2.58$, quand on réalise avec le programme les mesures des événements doubles, on obtient $L(c, X^2) = 2.33965$ et $L(c, (X')^2) = 2.55275$. On remarque donc que dans les deux cas on se rapproche de la borne inférieur ($H(X) = 2.322788$ ou de $H(X') = 2.536352$).

3.13 Des événements multiples de taille n

Nous avons donc réaliser le programme `huffman.c`, qui prend en paramètre un jeu de données et une taille de regroupement pour ces dernières.

On remarque que plus la taille est élevée, plus la longueur moyenne du code se rapproche de l'entropie de l'ensemble concerné.

On note les longueurs moyennes obtenues en fonction de la taille de regroupement n :

n	$L(c, X^n)$
1	2.380000
2	2.339650
3	2.332235
4	2.330744
5	2.328274
6	2.339650
7	en attente ...

TABLE 2 – Codes c de l'ensemble X

3.14 Limites physiques

Étant donné que le programme rédigé est `monthread`, on perd déjà beaucoup en terme de performances potentielles, aussi, un programme réalisant le calcul des X^{10} événements possibles requiert une mémoire et un processeur très puissant, que nous ne possédons pas.

REMARQUE : Le programme utilise un tableau statique de taille défini, on a donc ce critère qui joue énormément sur les performances (X^7 prend déjà énormément de temps dans notre cas).

3.15 Complexité algorithmique

On note N le nombre d'événements initiaux, et T la taille de regroupement choisie. La complexité de notre programme s'exprime de la manière suivante :

- $O(N^T)$, pour la création de chaque événements.
- $O(N * N^T)$, pour la construction de l'arbre de huffman.

On a donc au total une complexité algorithmique de $O(N^{3T})$ pour notre programme.

3.16 Complexité mémoire

Nous avons estimé que la complexité mémoire réelle s'approche de la valeur suivante :

- $N^T * \text{sizeof}(\text{struct grp})$.

3.17 Mesures de performances

Avec T , la taille de regroupement des événements, on obtient grace au champs `real` de la commande `time` :

T	Temps
1	0.004s
2	0.015s
3	0.087s
4	0.375s
5	2.155s
6	19.105s
7	en attente ...

TABLE 3 – Temps d'exécution du programme en fonction de T

Deplus, avec la commande `valgrind`, on peut s'apercevoir de la quantité de mémoire allouée **dynamiquement** par le programme, toujours en fonction de T :

T	octets alloués
1	379 666
2	383 568
3	409 412
4	579 392
5	1 683 232
6	8 827 528
7	en attente ...

TABLE 4 – Quantité d'octets alloué dynamiquement en fonction de T

Cependant la taille de mémoire du tableau de données pour l'analyse de fichier en début de programme (dans le fichier `collection.h`) sera toujours de taille 6^6 , car on travail avec X , de cardinal 6, et de taille de regroupement (raisonnable ici) de 6.

En effet avec $T = 7$, on atteint les limites.

4 Codage Arithmétique

4.1 Le codage décodage arithmétique

La compression arithmétique lit les symboles d'un message à coder 1 à 1. Après l'ajout du premier symbole, l'intervalle aura les mêmes limites que l'intervalle du symbole ajouté, signifiant ainsi quelle lettre a été utilisée en première. Puis chaque ajout supplémentaire fera converger ses limites vers un nombre flottant caractéristique du message.

La décompression récupère ce flottant et le nombre de symboles, il ajuste ce nombre par la suite pour lire à partir de la table chaque lettre concerné dans l'ordre jusqu'à que $n = 0$.

4.2 Étape de programmation

Le programme `code_arithmetique.c` (compilable avec `make code_arithmetique`) prend en paramètre le message et retourne la moyenne des deux bornes sous forme de flottant.

4.3 Code de WIKI

Le $V_{message}$ de WIKI a pour valeur 0.171875

4.4 Code de KIWI

Le $V_{message}$ de KIWI a pour valeur 0.828125

4.5 Code de KIKIWIWI

Le $V_{message}$ de KIKIWIWI a pour valeur 0.915283

4.6 Etape de programmation

Le programme `decode_arithmetique.c` (compilable avec `make decode_arithmetique`) prend en paramètre le code d'un message et retourne ce message grâce au table des probabilités et des intervalles.

4.7 Questions 7, 8, 9 et 10

	Taille mot					
Code	1	3	4	6	8	10
0.008	W	WWW	WWWI	WWWIII	WWWIIIII	WWWIIIIKI
0.517	I	III	IIII	IIIIKW	IIIIKWIW	IIIIKWIWKI
0.164	W	WIK	WIKW	WIKWKK	WIKWKKKI	WIKWKKKKIII

TABLE 5 – Decodage arithmétique

4.8 Si n est plus petit que nécessaire

Lors du décodage, si le n utilisé est plus petit que la taille du message, alors il va manquer la fin du message, comme on peut le voir dans le tableau ci-dessous :

Code	1	2	3	4
0.828125	K	KI	KIW	KIWI

TABLE 6 – Decodage de KIWI en fonction de n

4.9 Si n est plus grand que nécessaire

Lors du décodage, si le n utilisé est plus grand que la taille du message, alors la dernière lettre se répétera encore et encore jusqu'à ce que n vaut 0.

Code	4	5	6	...
0.828125	KIWI	KIWII	KIWIII	...

TABLE 7 – Decodage de KIWI en fonction de n

4.10 Si n n'est pas fourni

On peut déterminer quand est-ce que l'algorithme doit s'arrêter de plusieurs manière, et ceci simplement : au moment où l'algorithme est sur une borne (sur 0.00, 0.25, 0.50 ou 0.75), et qu'il est sensé revenir sur cette dernière à la prochaine itération de décodage, alors on l'arrête et on dit que la lecture du message original est terminé. Cette méthode a été testé par notre groupe et un autre et s'est montrée aléatoire dans certains .

4.11 Optimisation de l'algorithme

Une optimisation que nous avons mis en place est de passer dans la partie entière du flottant la taille du message original, on aura donc par ce mécanisme une seule donnée qui en transmet beaucoup plus.