Compte-Rendu TP1

PARA Yaël - TEYSSIER Théo - GONZALEZ Jules

13/03/20

Table des matières

1 Codage et Entropie

1.1 Entropie de X

L'entropie de X = $\{e1, e2, e3, e4, e5, e6\}$ notée $H(X)$ vaut 2.323.

1.2 Taille d'un code à longueur fixe

La taille d'un code à longueur fixe se retrouve en faisant $\lceil \log_2 m \rceil$, avec m le nombre de symboles de l'alphabet. Ici il nous faudrait des mots de $\lceil \log_2 6 \rceil = 3$ bit.

1.3 Comparaison avec l'optimum

Ce codage a une longueur moyenne de 3, bien supérieur à l'optimum valant 2.323.

1.4 entropie.c

Le code est disponible dans le fichierentropie.c. Pout le compiler : `make entropie` Pour le lancer : `./entropie <nom du fichier contenant les données>`.

1.5 Test du programme sur X

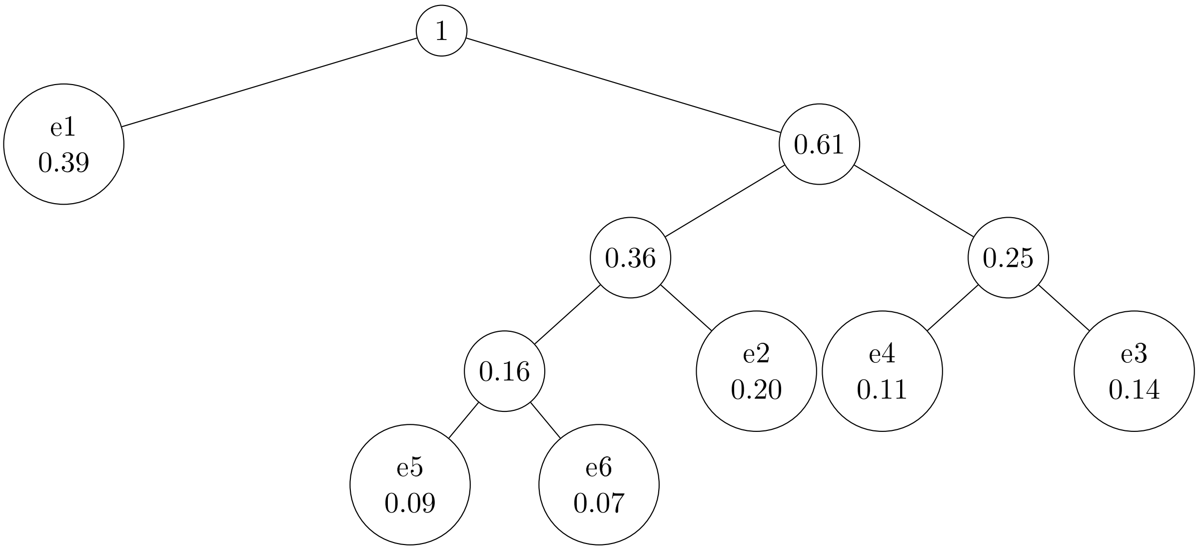
Figure 1 – Entropie de X

1.6 Test du programme sur X'

Figure 2 – Entropie de X'

2 Code de Huffman

2.1 Arbre de Huffman



2.2 Table de codage découlant de l'arbre

Événement	Code
e1	0
e2	101
e3	111
e4	110
e5	1000
e6	1001

Table 1 – Codes des élements de X

2.3 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 :

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

On trouve alors : $L(c, X) = 2.38$

2.4 Borne inférieure

La distance à la borne inférieure est donnée par $L(c, X) - H(X)$, on obtient ici : $2.38 - 2.32 = 0.06$.

2.5 Surcoût

Le surcoût de la transmission dépend du format établi pour la table de décodage. Dans notre cas, on a un surcoût de 66 bits.

2.6 Taux de compression

Le taux de compression de ce code par rapport au code de taille 3 est donné par :

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

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

2.7 Rentabilité de Huffman

Le code de Huffman est rentable lorsque n est tel que :

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

où t est 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 : $66 + (n * 2.38) < 3 * n \iff 66 < n(3 - 2.38)$, et on trouve $n > 107$.
Donc à partir de 107 événements envoyés, le code de Huffman est rentable.

2.8 huffman.c pour des événements simples

Le programme permettant de construire le code de Huffman pour des événements simples se trouve dans le fichier huffman.c

2.9 Un pas vers la borne inférieur

Le cardinal de l'ensemble X^2 est : $card(X^2) = card(X)^2$, soit 36 événements possibles pour X^2.

2.10 Les événements double

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

2.11 Résultats des événements double

Longueur moyenne pour X avec des événements doubles : 2.340 Longueur moyenne pour X' avec des événements doubles : 2.553

2.12 Des événements multiples de taille n

Le programme huffmanMult.c permet de traîter des événements multiples.
Plus la taille est élevée, plus la longueur moyenne du code se rapproche de l'entropie.
On note les longueurs moyennes obtenues en fonction de la taille de regroupement n :

n	$L(c, X^n)$
1	2.380
2	2.340
3	2.332
4	2.331
5	2.3283
6	2.3276

Table 2 – Codes c de l'ensemble X

2.13 Limites physiques

Nos machines de posséde pas de processeurs assez puissant pour réaliser ces calculs dans des délais raisonnables et la taille de la mémoire peut également être un facteur bloquant le traitement d'évenements aussi gros.

2.14 Compléxité algorithmique

On note N le nombre d'événements initiaux, et G la taille de regroupement choisie.
La compléxité de notre programme est : $2N + 3N^G + 2^(2G)$

2.15 Compléxité mémoire

La complexité mémoire de notre programme est : $N^G(2N + 1052) + 8 + (N + 8)G + 10N$

2.16 Mesures de performances

Voici nos différentes durées d'exécution en fonction de N :

N	Temps
1	0.002s
2	0.002s
3	0.002s
4	0.013s
5	0.176s
6	10.972s

Table 3 – Temps d'éxecution du programme en fonction de N

La quantité de mémoire alloué en fonction de N :

N	octets alloués
1	6 498
2	38 484
3	230 580
4	1 384 236
5	8 312 652
6	49 922 028

Table 4 – Quantité d'octets alloué dynamiquement en fonction de N

On atteint les limites des capacités de nos machines pour $N = 7$ car le temps d'exécution devient trop important.

3 Codage arithmetique

3.1 Principe des algorithmes

La compression arithmétique permet la compression d'un message ou chaque symbole est représenté sous un nombre flottant appartenant à un intervalle, qui est décrit dans une table de correspondance. En effet chaque symbole est associé dans la table à un sous intervalle de [0,1[, qui est découpé équitablement en fonction du nombres de symboles différents présents dans la table. Chaque symbole possède également une probabilité d'apparition dans le message à coder.

Chaque symbole du message est codé par un flottant, et un symbole est décrit dans la table par un intervalle de flottants. Donc notre compression associe un nombre flottant à chaque symbole du message, tandis que la décompression permet de retrouver les symboles du message en testant l'appartenance des flottants aux différents intervalles décrits dans la table.

Ainsi, l'algorithme de compression nous renvoie un nombre flottant (Vmess) qui est inclus dans l'intervalle associé au premier symbole du message. Ceci est assuré dès la première itération de l'algorithme, où les bornes Binf et Bsup, utilisées pour le calcul de Vmess, sont fixées aux bornes inférieure et supérieure de l'intervalle de ce premier symbole. Par la suite, à chaque itération (qui correspond à l'ajout d'un symbole du message à compresser) , les calculs effectués sur les bornes permettent de faire converger l'intervalle [Binf, Bsup] d'appartenance du Vmess final, qui va représenter le message sous forme compressée.

Lorsqu'on utilise l'algorithme de décompression, Vmess nous permet de retrouver le premier symbole du message, car il est inclus dans l'intervalle correspondant au premier symbole (Vmess appartient à l'intervalle [Binf, Bsup], qui est plus petit que l'intervalle du symbole dans la table).

Le Vmess calculé à chaque itération dans l'algorithme de décompression appartient à l'intervalle du symbole à décompresser, on retrouve cette correspondance en testant l'appartenance du Vmess calculé aux intervalles des symboles dans la table.

3.2 Algorithme de compression

L'algorithme de compression se trouve dans le fichier compression.c

3.3 Test de la compression

Le Vmessage correspondant à « WIKI » est 0.171875
Le Vmessage correspondant à « KIWI » est 0.828125
Le Vmessage correspondant à « KIKIWIWI » est 0.915283
On remarque bien que le Vmessage calculé appartient toujours à l'intervalle correspondant à la première lettre du message.

3.4 Algorithme de décompression

L'algorithme de compression se trouve dans le fichier decompression.c

3.5 Test de la décompression

Le message décompressé dépend de la longueur du message, ici nous avons choisi de nous fixer à une taille de 12 lettres.

Nous avons ainsi les résultats suivants :
Vmessage : 0.008 - > Message : WWWIIIIIKIII
Vmessage : 0.517 - > Message : IIIIKWIWKIWW
Vmessage : 0.164 - > Message : WIKWKKKIIIII
Vmessage : 0.312 - > Message : IWIIIIIIWKKI

3.6 Avec n plus petit

Si le n utilisé lors de la décompression est plus petit que la longueur du message compressé, on obtient le message partiellement décompressé. Par exemple, avec la compression du message « WIKI », nous avons obtenu Vmess = 0.171875. En utilisant l'algorithme de décompression avec ce dernier et avec une taille n égale à 2, nous obtenons le message « WI »

3.7 Avec n plus grand

Si le n utilisé est plus grand que la taille du message, nous obtenons souvent une répétition de la derniere lettre I.
Par exemple, avec la compression du message « KIKIWIWI », nous avons obtenu Vmess = 0.915283. En utilisant l'algorithme de décompression avec cette valeur et avec une taille n égale à 20, nous obtenons le message « KIKIWIWIIIIIIIIIIWII»
De même avec le Vmessage de « WIKI », qui décompressé avec une taille de 20 donne «WI-KIIIIIIIIIIIIIIII ».
Avec un test sur le message « KKKK », nous obtenons KKKKIIIIIIIIIIIIIIKK

Cela est dû au fait que le dernier Vmessage calculé lors de la sortie de l'algorithme de décompression dans le cas d'une taille n correspondant à la taille du message est proche de 0.5, qui appartient à l'intervalle correspondant à la lettre I dans la table.
Comme la lettre I est celle a une probabilité de 0.5, on a alors pour les calculs suivants de Vmess des valeurs s'approchant de :
Vmess = (0,5 (val approximative de Vmess) – 0,25 (borne inférieure de l'intervalle de I)) / 0,5 (proba de I) = 2 * (0,25) = 0,5
donc Vmess continue a avoir des valeurs proches de 0,5 si l'algorithme continue après avoir atteint la fin du message, d'où la répétition du symbole I .

3.8 Avec une taille inconnue

Pour générer un flux de symboles sans connaître sa taille initiale, on peut ajouter un symbole qui marque la fin de la fin du message dans la table, ainsi lorsque l'on décompresse le message, on sait où s'arrêter.