**Codage arithmétique**

Cette partie va présenter une compression sans perte appelé codage arithmétique. Elle va d'abord définir l'intérêt de l'algorithme avant de présenter la compression et la décompression effectuées par ce codage

**Description**

Le codage arithmétique est un codage statistique, c'est-à-dire que plus un caractère est représenté, moins il faudra de bits pour le coder.

Il s’agit d’un cousin du codage de Huffman qui cependant reste toujours plus efficace que ce dernier (sauf dans le cas particulier où tous les poids des feuilles/nœuds/racines de l’arbre de Huffman sont des puissances de 2). Il est aussi plus simple à implémenter.

L’avantage que possède le codage arithmétique sur le codage de Huffman est que ce dernier va coder un caractère sur un nombre entier de bits (il ne peut coder sur 1.5 bits) là où le codage arithmétique le peut. Par exemple, si un caractère est représenté à 90%, la taille optimale du code du caractère serait de 0.15 bit, alors que Huffman coderait sûrement ce symbole sur 1 bit, soit 6 fois trop.

Ce codage n’est que très peu utilisé en pratique mais elle reste présente, notamment dans le format JPEG2000.

**Compression**

Pour présenter la compression, nous allons utiliser un exemple et nous décrirons chaque étape de compression. Codons le mot "ESIPE" à l’aide du codage arithmétique.

La première étape consiste à décompter chaque lettre du mot. Nous avons donc 2 ‘E’, 1 ‘S’, 1 ‘I’ et 1 ‘P’. Nous en générons alors une probabilité de présence dans le mot soit 40% de chance de trouver un E et 20% de chance pour les autres lettres. Dernière actions à effectuer pour cette première partie, nous affectons à chaque lettre un intervalle entre 0 et 1 de la manière suivante :

* La lettre ‘E’ à une probabilité de 40% (soit 0.4). Son intervalle est donc [0,0.4[
* La lettre ‘P’ a une probabilité de 20% (soit 0.2). Son intervalle est donc [0.4,0.6[
* Etc...

On obtient dès lors le tableau suivant :

|  |  |  |
| --- | --- | --- |
| **Lettre** | **Probabilité** | **Intervalle** |
| E | 4/10 | [0,0.4[ |
| S | 2/10 | [0.4,0.6[ |
| I | 2/10 | [0.6,0.8[ |
| P | 2/10 | [0.8,1.0[ |

Le codage va maintenant consister à remplacer le mot ESIPE par un nombre flottant lui correspondant. Pour cela, le mot va se voir affecter un intervalle compris entre 0 et 1 où chaque nombre compris entre les deux intervalles permettra de retrouver le mot ESIPE.

L’algorithme appliqué est le suivant : le mot commence avec un intervalle de [0,1[. Puis pour chaque lettre croisée, nous appliquons la formule suivante :

* La borne inférieure (BI) du mot est modifiée avec le résultat du calcul "BI + (BS – BI) \* Borne\_Inférieure\_Lettre"
* La borne supérieure (BS) du mot est modifiée avec le résultat du calcul "BI + (BS – BI) \* Borne\_Supérieure\_Lettre"

Le tableau suivant montre les étapes du calcul:

|  |  |  |
| --- | --- | --- |
| **Lettre** | **Borne Inférieure** | **Borne Supérieure** |
|  | 0.0 | 1.0 |
| E | 0.0 | 0.4 |
| S | 0.16 | 0.24 |
| I | 0.208 | 0.224 |
| P | 0.2208 | 0.224 |
| E | 0.2208 | 0.22208 |

Dès lors, tous nombre flottant entre 0.2208 et 0.22208 est le format compressé du mot "ESIPE"

**Décompression**

De la même manière, nous allons présenter la décompression par l’exemple en décompressant notre format compressé.

Prenons le nombre 0.2208 qui code le mot "ESIPE". Le principe de la décompression est très simple. Celle-ci suit les deux étapes suivantes qui se répète jusqu’à l’obtention du mot :

* La prochaine lettre du mot est celle dont l’intervalle contient le nombre du mot actuel (Ex : 0.2208 est dans l’intervalle de E donc la première lettre est E).
* On modifie le nombre représentant le mot à l’aide du calcul « (nombre du mot – borne inférieure de la lettre) / probabilité de la lettre (Ex : nombre du mot = (0.2208 – 0.0) / 0.4 = 0.552)

Le tableau suivant montre les différentes étapes de la décompression :

|  |  |  |
| --- | --- | --- |
| **Mot** | **Lettre** | **Nouveau code** |
|  | E | 0.552 |
| E | S | 0,76 |
| ES | I | 0,8 |
| ESI | P | 0,0 |
| ESIP | E |  |

Ainsi, nous avons récupéré notre premier mot: ESIPE

Algorithme de codage

// Tableau contenant les bornes inferieures des intervalles

 // calcules ci-avant Borne\_Inf[];

// Idem pour la borne sup Borne\_Sup[];

// Initialisation Inf=0.0; Sup=1.0;

 c = lireCaractere(); TantQue (c!= EOF)

 { Sup = Inf + (Sup - Inf) \* Borne\_Sup[c];

 Inf = Inf + (Sup - Inf) \* Borne\_Inf[c];

c = lireCaractere();