TP Codage Huffman

Le codage de Huffman est une méthode de compression statistique de données qui permet de réduire la longueur du codage d'un alphabet.

LE PRINCIPE

Le principe de l'algorithme de Huffman consiste à recoder les octets rencontrés dans un ensemble de données source avec des valeurs de longueur binaire variables.

L'unité de traitement est ramenée au bit. Huffman propose de recoder les données qui ont une occurrence très faible sur une longueur binaire supérieure à la moyenne, et recoder les données très fréquente sur une longueur binaire très courte.

Ainsi, pour les données rares, nous perdons quelques bits regagnés pour les données répétitives. Par exemple, dans un fichier ASCII le "w" apparaissant 10 fois aura un code très long: 0101000001000. Ici la perte est de 40 bits (10 x 4 bits), car sans compression, il serait codé sur 8 bits au lieu de 12. Par contre, le caractère le plus fréquent comme le "e" avec 200 apparitions aura un code à 1. Le gain sera de 1400 bits (7 x 200 bits). On comprend l'intérêt d'une telle méthode.

De plus, le codage de Huffman a une propriété de préfixe : une séquence binaire ne peut jamais être à la fois représentative d'un élément codé et constituer le début du code d'un autre élément.

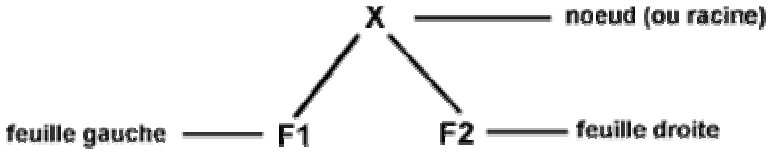
Si un caractère est représenté par la combinaison binaire 100 alors la combinaison 10001 ne peut être le code d'aucune autre information. Dans ce cas, l'algorithme de décodage interpréterait les 5 bits comme une succession du caractère codé 100 puis du caractère codé 01. Cette caractéristique du codage de Huffman permet une codification à l'aide d'une structure d'arbre binaire.

LA MÉTHODE

L'algorithme se décompose en deux phases. La première consiste à lire entièrement l'ensemble des données source et à comptabiliser le nombre d'apparitions de chaque information : on bâtit ainsi une table des occurrences. Cette table est ordonnée suivant l'ordre décroissant des fréquences.

Ensuite, on procède à des réductions successives jusqu'à la fin de la table. Une réduction consiste à prendre les deux éléments de la table ayant les plus petites fréquences (ou probabilités) et à les assembler. À ce nouvel élément ainsi constitué, on associe une fréquence résultant de la somme des deux fréquences associées. Puis il est reclassé dans la table; bien entendu, les deux membres initiaux en ont été éliminés.

L'assemblage se fait par l'intermédiaire d'un arbre binaire. On construit un noeud (ou racine pour le premier) avec deux branches. L'élément de plus faible fréquence est placé sur la branche de droite (appelée feuille droite), l'autre sur la branche de gauche (appelée feuille gauche). On pourra montrer que l'on peut inverser la répétition des deux feuilles.

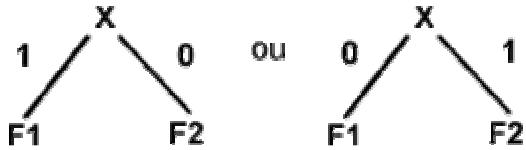


Ainsi, deux éléments n'en font plus qu'un, et le nombre d'éléments total de la liste diminue de un à chaque itération. Cette opération est répétée jusqu'à ce qu'il n'y ait plus qu'un seul élément. Ce dernier noeud sera précisément la racine de l'arbre final.

Une fois l'arbre terminé, nous pouvons attribuer aux feuilles (donc les informations source) des valeurs dictées par la structure de l'arbre.

Soit, pour chaque élément, on veut atteindre sa feuille correspondante, tout en commençant la recherche à partir de la racine de l'arbre. À chaque pas à droite, on ajoute "0" à une chaîne représentative du codage qui sera initialement nulle; à chaque pas à gauche, on ajoute "1" et cela jusqu'à ce qu'on atteigne la feuille.

Sur la figure ci-dessous, on remarque que l'attribution d'un "0" à la branche de droite d'un noeud (respectivement "1" pour la branche gauche) est arbitraire; on pourrait attribuer 1 à la branche gauche (respectivement "0" à la branche droite).



Seul le codage sera différent avec des "0" à la place des "1" et des "1" à la place des "0", mais la longueur du code sera la même, ce qui, est le but de la compression.

Il est évident qu'une fois le codage effectué, il faut relire l'ensemble des données pour les coder. Aussi, pour que le codeur et le décodeur soient en phase, il faut transmettre la table codage/décodage avant de coder les informations.

LES ALGORITHMES

I- Phase de compression

Algorithme de compression

```
Fonction Huffman → arbre
    données : l'alphabet et les fréquences de chaque lettre
    résultat : l'arbre binaire du codage de Huffman
    Algorithme
        créer une file à priorités
        pour chaque caractère c de l'alphabet faire
            créer un arbre a
            a.caractère ← c
            a.poids ← fréquence de c
            ajouter a à la file à priorités
        tant que la file à priorités contient plus d'un arbre faire
            créer un arbre a
            a.gauche ← l'arbre de poids le plus faible de la file à priorités (que l'on retire de la file)
            a.droite ← l'arbre de poids le plus faible de la file à priorités (que l'on retire de la file)
            a.poids ← a.gauche.poids + a.droite.poids
            ajouter a à la file à priorités
        retourner le seul arbre contenu dans la file à priorités
```

II- Phase de décompression

Une des caractéristiques générales du codage de Huffman est qu'un code ne peut pas être le début d'un autre; aussi, en lisant bit à bit le fichier compressé, on est capable de redescendre l'arbre de codification jusqu'à une feuille, donc un caractère décompressé.

On utilise l'en-tête (table de codification) du fichier compressé pour reconstruire l'arbre original. Cette partie n'est pas détaillée car elle ressemble à la construction de l'arbre dans la phase de compression.

A Faire

Codage

1. Lire le texte contenu dans le fichier à coder, calculer la fréquence d'apparition de chaque caractère et construire une table de fréquences, enregistrer cette table dans le fichier codé de sortie pour permettre de reconstruire l'arbre de Huffmann lors du décodage.
2. Construire l'arbre de Huffman, puis en déduire une table de codage.
3. Lire une deuxième fois le texte, le coder avec la table de codage puis enregistrer le résultat dans le fichier codé de sortie.

Décodage

1. Lire dans le fichier à décoder la table de fréquences.
2. Construire l'arbre de Huffman.
3. Lire le texte codé, le décoder avec l'arbre de Huffman puis enregistrer le résultat dans le fichier de sortie.

Evaluation des performance

1. Calculer le taux de compression (ratio: taille fichier codé/taille fichier non codé) et le nombre de bits moyen par symbole.
2. Calculer la durée des différentes opérations.
3. Calculer l'entropie.
4. Comparer à chaque fois avec un codage à taille fixe en utilisant le fichier **test.txt** sur la plateforme.
5. Utiliser la commande *diff* sur Linux pour comparer le fichier original et le fichier décompressé.

CONSIDÉRATIONS COMPLÉMENTAIRES

Il ne faut pas perdre de vue le fait que le décompresseur doit avoir les informations qui lui permettent de décompresser. C'est-à-dire que l'on devra faire figurer en tête de fichier les clés de décodage des octets. Pour les petits fichiers, cet en-tête prend proportionnellement beaucoup trop de place et le gain peut être négatif.

Malgré son efficacité, cet algorithme comporte quelques inconvénients. Premièrement, afin de constituer notre table des fréquences et le codage de l'information, il faut lire une première fois l'ensemble des données sources. En fonction de sa dimension, on perd plus ou moins de temps. Deuxièmement, lors de la transmission d'information, il faut copier cette table de décodage pour que le décodeur restitue l'ensemble des données sources.

En fait, ces deux défauts sont liés à la méconnaissance de la table de codage/décodage. On peut améliorer cet algorithme en compilant une statistique décrivant la répartition des données susceptibles d'être rencontrées.

Il existe en effet des tables de statistiques pour l'alphabet lexical. Ces probabilités d'apparition changent suivant la langue utilisée dans le fichier source. Par contre, ces tables statistiques intégrées dans un compresseur/décompresseur limitent l'utilisation d'un tel programme à la compression de fichier texte uniquement.

Lettre	Français	Anglais
A	0.1732	0.1859
B	0.0690	0.0642
C	0.0068	0.0127
D	0.0339	0.0317
E	0.1428	0.1031
F	0.0095	0.0208
G	0.0098	0.0152
H	0.0048	0.0467
I	0.0614	0.0575
J	0.0024	0.0008
K	0.0006	0.0049
L	0.0467	0.0321
M	0.0222	0.0198
N	0.0650	0.0574
O	0.0464	0.0632
P	0.0261	0.0152
Q	0.0104	0.0008
R	0.0572	0.0484
S	0.0624	0.0514
T	0.0580	0.0796
U	0.0461	0.0228
V	0.0104	0.0083
W	0.0005	0.0175
X	0.0035	0.0013
Y	0.0018	0.0164
Z	0.0006	0.0005