TP ISBS 2005
Compression

Hugues Talbot (talbot@esiee.fr)

20 mai 2005

1 Introduction

Ce TP est une introduction aux algorithmes de compression sans pertes (lossless). Ces méthodes sont essentielles de nos jours et forment la bases des logiciels de compresssions de fichiers tel que `gzip`, `zip`, `compress`, etc. On les retrouve également dans toutes les méthodes de compression d'images, qu'elle soient avec ou sans perte (formats GIF, PNG pour les première, JPEG pour les secondes).

2 Compression sans perte

Dans cette section on étudie la compression de Huffman, qui est optimale sous certaines conditions, et qui illustre relativement bien la méthodologie générale de compression sans perte de fichiers généraux.

La compression (encore appelé codage) sans perte est un moyen de représenter des données arbitraires d'une manière telle qu'elles occupent moins d'espace. Cela est rendu possible en exploitant les redondances inhérentes à certains types de données. Par exemple en français ou en anglais, il y une prépondérance d'espaces et de la lettre "e".

2.1 Méthodologie générale

Le codage de Huffman consiste en les étapes suivantes. On suppose les données encodées sur 8 bits (ASCII, ISO Latin-1, etc).

1. Construire un histogramme des fréquence des données d'entrées
2. Grouper récursivement les données par paires, en commencant par les octets les moins fréquents, et accumuler leur fréquence. On construit ainsi une structure d'arbre binaire.
3. On associe chaque donnée avec un code d'un nombre faible de bits dérivés de la structure d'arbre. On construit ainsi un dictionnaire.
4. On remplace les données d'entrées par leur code en utilisant le dictionnaire.

Pour réaliser l'opération inverse de décodage, en supposant le dictionnaire connu, on remplace simplement les données compressées par leur entrée dans le dictionnaire.

2.2 Construction de l'histogramme des données

On illustre la technique sur une chaîne de caractères en majuscule et sans accents. On ignore les guillements mais pas les espaces. On dispose aussi d'un texte plus long en anglais sous forme de fichier appelé `sample.txt`.

```
"A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"
```

On construit l'histogramme de la distribution des caractères :

h	1	1	1	2	2	2	2	2	3	3	3	3	4
alpha	C	F	P	D	G	L	R	U	A	B	O	T	M
h	4	5	5	6	11								
alpha	S	E	N	I	" "								

Dans ce tableau, les fréquences des lettres sont situées immédiatement en dessus d'elles. l'histogramme du texte `sample.txt` est lui illustré par la figure. 1.

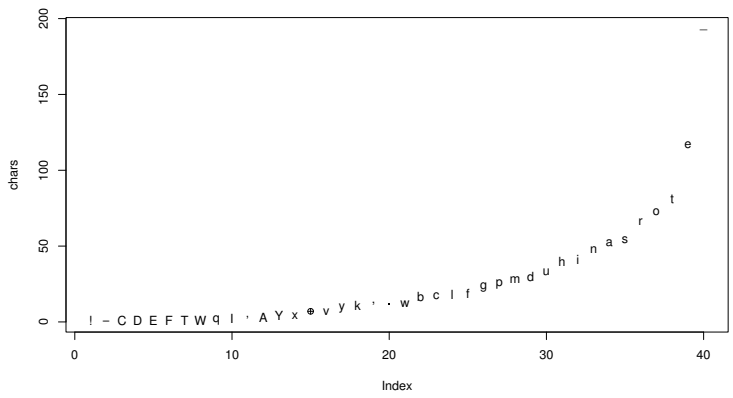

FIG. 1 – Distribution des fréquence des Lettres du fichier `sample.txt`. Le souligné est en fait le symbole pour la lettre espace

.

2.3 Construire la structure d'arbre de codage

L'idée principale est de construire une structure d'abre binaire telle que les lettres les plus fréquentes sont proches de la racine de l'arbre et les lettres les moins fréquentes sont plus bas dans la hierarchie.

Pour arriver à ce résultat, on ordonne les lettres de la moins fréquente à la plus fréquente, avec leur fréquence. On choisit les deux moins fréquentes, avec laquelle on construit un noeud d'arbre binaire, avec les deux lettres comme feuilles à droite et à gauche. On additionne leurs fréquences et on insère ce nouvel objet au bon endroit dans l'histogramme, puis on recommence.

En utilisant la chaîne de caractères plus haut comme exemple, les premières étapes de la construction de l'arbre sont les suivantes :

1. L'histogramme de départ :

1	1	1	2	2	2	2	2	3	3	3	3	4	4	5	5	6	11
C	F	P	D	G	L	R	U	A	B	O	T	M	S	E	N	I	_

2. On groupe les lettres `C` et `F` ensemble et on somme leur fréquences en créant un noeud d'arbre binaire. On insère ce noeud au bon endroit dans l'histogramme.

1	2	2	2	2	2	2	3	3	3	3	4	4	5	5	6	11
P	D	G	L	R	U	○	A	B	O	T	M	S	E	N	I	_
						C F										

3. Ensuite on groupe `P` et `D` ensemble, puis `G` et `L`, et ensuite `R` et `U`, toujours en sommant les fréquences. On obtient l'histogramme suivant :

2	3	3	3	3	3	4	4	4	4	5	5	6	11
○	A	B	O	T	○	M	S	○	○	E	N	I	_
C F					P D			G L	R U				

h

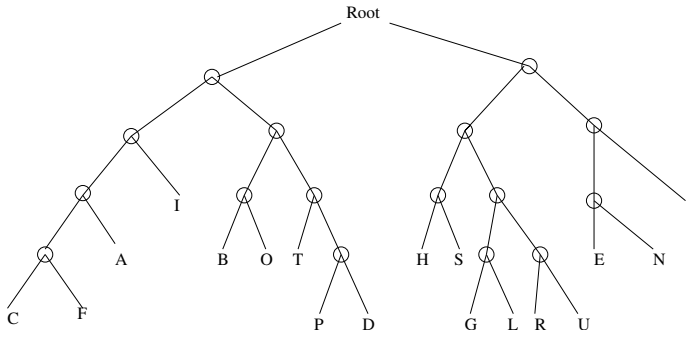

FIG. 2 – Arbre final pour la chaîne donnée en exemple.

4. Les groupes ou noeuds de l'arbre sont traîtés exactement comme des lettres, donc maintenant on groupe le groupe (`C` et `F`) ensemble avec A, toujours en additionnant les fréquences et en insérant dans l'histogramme au bon endroit :

3	3	3	3	4	4	4	4	5	5	5	6	11
B	O	T	∘	M	S	∘	∘	E	N	∘	I	_
			P D			G L	R U			∘ A		
										C F		

On continue de cette manière à construire récursivement un arbre jusqu'à obtenir celui de la figure 2.

La structure d'histogramme finale ne contient au bout du compte qu'un seul noeud, avec 60, le nombre total de lettres dans l'exemple comme fréquence.

2.4 Production du code

Après tout cet effort, une fois l'arbre construit, un code unique peut être associé avec chaque lettre. Pour imprimer le code, on part de la racine et on écrit "0" si on va à gauche et "1" si on va à droite. On stoppe lorsqu'on arrive à une lettre. Cette succession de 1 et de 0 est le code binaire de la lettre. Par exemple le code de `F` est "00001". L'ensemble du code pour toutes les lettres et appelé le dictionnaire.

Once the tree is constructed, a unique code can be associated with each letter. To find the code, start from the root and write "0" if going left, "1" if going right and s stop once at a letter. The succession of 1s and 0s is the binary code of the letter. For example the code of `F` is "00001". The code for all the letters is called the codebook or the dictionary, comme suit :

```
      C       F       A       I       B       O       T       P       D       M
"00000" "00001"  "0001"   "001"  "0100"  "0101"  "0110" "01110" "01111"  "1000"
      S       G       L       R       U       E       N
 "1001" "10100" "10101" "10110" "10111"  "1100"  "1101"   "111"
```

On note que dans cet exemple le code le plus long a 5 bits. Les lettres plus fréquentes (telles l'espace) sont codées sur moins de bits.

2.5 Génération du texte encodé

On génère le texte encodé de façon très simple, il suffit de remplacer chaque lettre par son code binaire. Chaque succession de 8 bits forme un nouveau caractère.

Notez que le nombre total de bits pour le texte de la chaîne de caractère du départ est 236. Le texte codé en ASCII/ISO Latin 1 sur 8 bits utiliserait 480 bits. La procédure de HUFFMAN réduit la taille du texte d'un facteur supérieur à deux.

```
 [1] 236  31  38  29  92 242 214  59  78 203 211  62 104  21 248 255 121  59  78
[20]  62  14 152  26 253 188  38  91 168 122  22   9
```

2.6 Remarques

Voici quelques remarques sur le codage de Hufman Some remarks about Huffman are in order :

1. Les arbres de Huffman sont optimaux. Il n'existe pas de codage à base d'arbres qui soient plus efficaces que cette méthode.
2. Cependant on doit aussi joindre le dictionnaire à toute transmission de données codées par l'algorithme de Huffman.
3. De ce fait des méthodes de codages légèrement plus compliquées sont parfois préférées pour la compression de données générale, telle que les méthodes LZ, LZW, codage arithmétique, etc. Le principe de ces méthode n'est pas très différent de celui de Huffman. Toutefois, le codage de Huffman est bel et bien utilisé dans la norme JPEG.
4. Les taux de compression obtenus par Huffman (sans le problème du dictionnaire) sont un idéal auquel on compare toutes les autres méthodes. On peut aussi s'en servir comme une indication du niveau de redondance présent dans les données à notre disposition.

3 Contenu du TP

Ce TP est à rendre le vendredi 3 juin 2005, par courrier électronique à l'adresse de l'auteur (`talboth@esiee.fr`). Vous devez rendre un rapport sous forme d'un document au format MS-Word ou PDF, et un programme en Java qui répond au problèmes posés. Le programme sera soit constitué d'un projet complet BlueJ ou d'une collection de fichiers `.java`.

Dans le rapport, explicitez les solutions choisies, en particulier justifiez les structures de données que vous aurez utilisé.

Vous pouvez vous constituer en binôme. Le rapport et le programme devront mentionner dans chaque fichier le nom des auteurs.

La collaboration entre binômes est souhaitable et encouragée, mais chaque binôme devra proposer une solution unique. Le plagiat est interdit.

Exercise 3.1 (Encodage)
En suivant les indications du texte, mettez en oeuvre la méthode de compression de Huffman *en Java. Testez sur le texte* `sample.txt`. *Quel taux de compression est obtenu ?*

Exercise 3.2 (Decodage)
Ecrivez une fonction qui puisse décoder le résultat d'un codage de Huffman*, en supposant le dictionnaire connu. Comment se fait il que le code ne soit pas ambigü ? Testez sur la chaîne de caractère et sur le texte d'exemple.*

Exercise 3.3 (Transport du dictionnaire)
Proposer une façon efficace d'inclure le dictionnaire dans le message compressé. Proposez deux programmes, un de compression, l'autre de décompression, permettant l'échange de données compressée sans connaissance à priori du dictionnaire.

Comparez le taux de compression obtenu avec le programme `zip`.