Projet d'Algorithmique

Transformée de Burrows-wheeler

1 Description de la méthode

La transformée de Burrows-Wheeler, couramment appelée BWT (pour Burrows-Wheeler Transform) est une technique de compression de données. Elle fut inventée par Michael Burrows et David Wheeler. Cette technique fut rendue publique en 1994, suite à de précédents travaux de Wheeler en 1983. Il ne s'agit pas à proprement parler d'un algorithme de compression, car aucune réduction de taille n'est effectuée, mais bien d'une méthode de réorganisation des données : les probabilités pour que des caractères identiques initialement éloignés les uns des autres se retrouvent côte à côte sont alors augmentées. Cette technique n'est pas très utilisée, mais l'on peut cependant remarquer qu'elle est présente dans le format bzip2 qui est actuellement l'un des formats offrant le plus grand quotient de compression.

1.1 La transformation

Tout d'abord, la chaîne de caractères à coder doit être copiée dans une matrice en décalant la chaîne d'un caractère vers la droite à chaque nouvelle ligne. Ces lignes sont ensuite classées par ordre alphabétique. Nous savons que, grâce au décalage, chaque dernière lettre de chaque ligne précède la première lettre de la même ligne, sauf pour la ligne originale dont on notera la position. De plus, comme les lignes sont rangées par ordre alphabétique, on peut retrouver la première colonne du tableau grâce à la dernière colonne.

Prenons un exemple. Supposons que la chaîne à coder soit "TEXTE". On calcule tout d'abord la matrice suivante:

T E X T E
E T E X T
T E T E X
X T E T E
E X T E T

Puis l'on classe ces chaînes par ordre alphabétique :

E T E X T
E X T E T
T E T E X
T E X T E
X T E T E

Ensuite, on garde en mémoire la ligne I de la matrice qui correspond à la chaîne originale, ici 4 ainsi que la dernière colonne L, ici TTXEE. Pour coder la liste L, on utilise en général l'algorithme Move-To-Front (voir Section 1.3).

1.2 La transformation inverse

Lors de la décompression, la chaîne codée est rangée par ordre alphabétique (on reprend l'exemple précédent, cette fois-ci dans le sens de la décompression) :

	1	2	3	4	5
Codé	T	T	X	E	E
Classé	E	E	T	T	X

C'est ici que l'on se sert du chiffre transmis (4). Nous savons que les deux caractères correspondant à cet indice ne se suivent pas et que le caractère de la ligne classée est le premier de la chaîne originale.

On part donc ici du 'T' en position 4. Ce 'T' est le deuxième de la ligne classée. On recherche donc le deuxième 'T' de la ligne codée, ce qui correspond à la position 2. Ce 'T' est donc suivi d'un 'E'. Ce 'E' est le deuxième de la ligne classée. On retourne donc chercher le deuxième 'E' de la ligne codée. On arrive en position 5. Ce 'E' est suivi d'un 'X'.... On continue ainsi jusqu'à tomber sur le 'E' en position 4 de la ligne codée. La décompression est alors terminée. On retrouve bien nos données initiales, à savoir la chaîne "TEXTE".

1.3 L'algorithme Move-To-Front

L'algorithme MTF (Move-To-Front) sert à coder une liste de caractères L par un vecteur d'indices. La technique consiste à remplacer chaque caractère par un indice, donné par un tableau évoluant de manière dynamique.

Le tableau est tout d'abord initialisé en rangeant les caractères utilisés pour le codage comme ceci :

Indice	0	1	2	3	4	5	6	...	25
Caractère	A	B	C	D	E	F	G	...	Z

Lorsqu'un caractère est lu, son indice est émis, puis ce caractère est placé en première position et tous les autres caractères décalés (d'où le nom de Move to Front). Par exemple si le premier caractère à coder est un E, le tableau deviendrait :

Indice	0	1	2	3	4	5	6	...	25
Caractère	E	A	B	C	D	F	G	...	Z

Ainsi, lorsque des caractères semblables se suivent (cas de la transformée de Burrows-Wheeler), le flux émis contiendra beaucoup de 0, ce qui dans une compression statistique (type codage de Huffman décrit dans la section 1.4) augmentera considérablement le gain de compression. On note que dans ce cas, l'émission d'un 0 laisse le tableau identique, et que dans les autre cas, le réarrangement ne concerne que les premiers éléments du tableau.

Par exemple, la séquence "EEEEEA" serait transformée en la suite "400001"; le tableau évoluerait comme suit:

Indice	0	1	2	3	4	5	6	...	25
État initial	A	B	C	D	E	F	G	...	Z
Tableau modifié par le premier "E"	E	A	B	C	D	F	G	...	Z
Tableau conservé 4 par les 4 "E" suivants	E	A	B	C	D	F	G	...	Z
Tableau modifié par le "A"	A	E	B	C	D	F	G	...	Z

Le décodage est tout aussi simple: à partir du même tableau initial, il suffit d'émettre le caractère correspondant à l'indice et de ranger le tableau en passant ce caractère en premier. Le tableau évolue exactement comme pendant la phase de codage.

1.4 Encodage de Huffman

Étant donné un texte T construit sur un certain alphabet $\mathscr{A}$ contenant au moins deux lettres, la méthode de compression de Huffman consiste à transformer T en une suite de bits à l'aide d'une fonction d'encodage f vérifiant les propriétés suivantes :

- $f : \mathscr{A} \to \{0,1\}^*$,
- $x \neq y \Rightarrow f(x)$ n'est pas préfixe de $f(y)$,
- si x a strictement plus d'occurrences que y dans T, alors $f(x)$ n'est pas plus long que $f(y)$.

Noter que f est représentable par un arbre binaire complet dont les feuilles sont étiquetées par les éléments de $\mathscr{A}$. L'image de chaque lettre est obtenue à partir du chemin atteignant cette lettre depuis la racine, en considérant qu'une branche gauche code un 0 et une branche droite un 1.

Exemple : Si $\mathscr{A} = \{a,b,c,d\}$ et $f(a) = 1, f(b) = 01, f(c) = 001, f(d) = 000$, la fonction f est représentable par l'arbre de la figure 1.

La fonction f peut évidemment être étendue à l'ensemble des mots sur $\mathscr{A}$ en posant $f(x_1x_2\dots x_n) = f(x_1)f(x_2)\dots f(x_n)$. On a par exemple $f(daccaba) = 000\,1\,001\,001\,1\,01\,1$ avec la définition de f précédente.

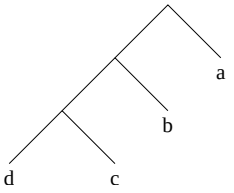

Figure 1: Arbre binaire associé à f

1.4.1 Construction de la fonction d'encodage

1) À partir d'une première lecture de T, on détermine pour chaque lettre x de $\mathscr{A}$ le nombre n_x d'occurrences de x dans T. Par exemple, pour $A = \{a,b,c,d,e\}$ et $T =$ *bcedaeadabae*, on obtient :

$$n_a = 4, n_b = 2, n_c = 1, n_d = 2, n_e = 3$$

2) On construit étape par étape l'arbre associé à une fonction d'encodage appropriée, à partir d'une forêt constituée d'autant d'arbres que d'éléments dans $\mathscr{A}$. Tous ces arbres sont réduits à leur racine et ont des poids égaux aux nombres d'occurrences de chaque lettre. À chaque étape, on supprime un arbre de la forêt en choisissant parmi les arbres restant deux arbres de poids minimaux. Ces deux arbres sont remplacés par un seul arbre ayant un fils gauche et un fils droit égaux aux deux arbres sacrifiés. De plus son poids est égal à la somme des poids de ces deux arbres.

Exemple : avec le nombre d'occurrences de l'exemple ci-dessus, on aurait 5 étapes comme décrit dans la Figure 2. Noter que la liste d'arbres est ici conservée triée par ordre de poids croissant, de manière à pouvoir construire chaque nouvel arbre à partir des deux premiers éléments.

2 Travail à réaliser

On vous demande d'écrire en pseudo-langage, puis programmer en C les opérations suivantes :

1. Une fonction qui prend en entrée une chaîne de caractères et lui fait subir la transformée de Burrows-wheeler. Cette fonction doit avoir des options qui permettent de visualiser les matrices obtenues lors des étapes intermédiaires.

2. Une fonction qui réalise la transformée inverse de Burrows-wheeler décrite dans la Section 1.2.

3. Une fonction qui prend en entrée une chaîne de caractères et lui fait subir la transformée Move-To-Front.

4. Une fonction qui réalise la transformée inverse de l'algorithme Move-To-Front.

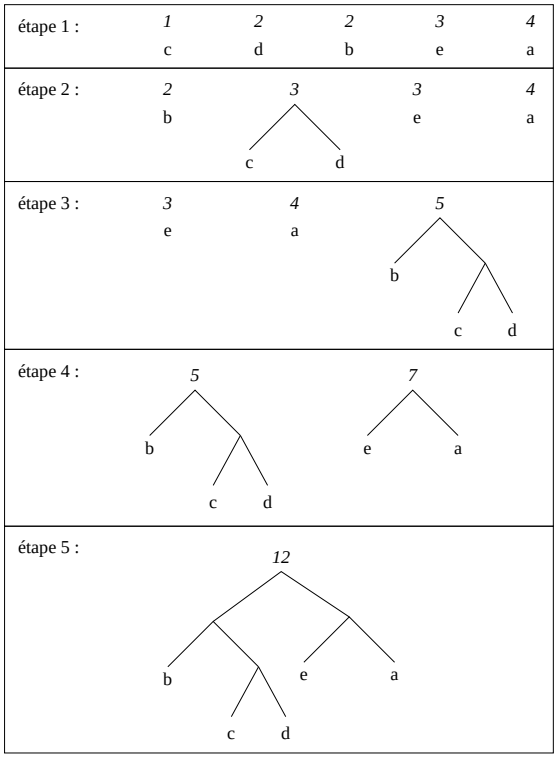

Figure 2: Les étapes de l'algorithme de Huffman

5. Une fonction qui prend en paramètre une chaîne de caractères et renvoie un arbre représentant la fonction de codage de Huffman pour cette chaîne.
6. Une fonction qui, étant donnés un caractère et une fonction d'encodage de Huffman appropriée, encode ce caractère.
7. Une fonction qui, étant donnés l'encodage d'un caractère et une fonction d'encodage de Huffman appropriée, décode ce caractère.
8. Une fonction qui décode une chaîne à partir de son code de Huffman.
9. Une fonction d'encodage qui calcule le code de Huffman d'une chaîne.
10. Une fonction qui reçoit une chaîne de caractères et produit en sortie un fichier qui contient différentes lignes correspondant aux items suivants:
    - La chaîne originale à compresser;
    - La chaîne L résultant de la transformée de Burrows-wheeler;
    - La position de la chaîne initiale dans la matrice triée;

- Le résultat R de la transformée MTF sur la chaîne L;
- La liste des encodages en bits de R par l'encodage de Huffman.

Pour chacun des algorithmes ci-dessus vous devez fournir :

- un fichier `doc.txt` qui spécifie l'algorithme en pseudo-langage (avec son auteur), et sa complexité,
- les sources C bien commentées du programme (avec son auteur), et
- un répertoire `tst` qui contient au minimum trois tests *intéressants* fait pour votre algorithme ainsi que les résultats obtenus. Vous devez bien sûr vérifier que le décodage de l'encodage d'une chaîne vous donne la chaîne elle même. De même, l'encodage du décodage d'un code vous redonne le code.

Un programme qui marche mais sans documentation, ni fichiers de test sera noté à 1/3 de la note finale.

3 Remise du projet

Le projet sera envoyé à l'adresse touili@liafa.jussieu.fr au plus tard le 4 février 2007 à minuit. Un fichier d'explication `remise_projet.txt` doit être inclus.

Ce qui doit être envoyé est un fichier d'archive Unix `VotreNom.tar` contenant les répertoires correspondant aux algorithmes ci-dessus.