Nancy-Université

ESSTIN – Semestre 2 – 2006-2007 – Informatique – Projet

Consignes

Projet à rendre en binôme pour le 1er juin. Votre projet devra contenir les éléments suivants :

- un fichier java pour chaque classe de votre programme, chaque méthode devra être commentée (entrées / sorties notamment) au moyen de lignes précédées du symbole "`//`",

- un compte rendu détaillé (au format PDF) donnant la structure de votre programme (schéma illustrant les attributs et méthodes de chaque classe) plus une explication du fonctionnement de chacune de vos méthodes. Si vous avez été amené à faire des choix, en particulier à prendre des conventions sur le type d'information manipulée, vous le mentionnerez dans votre compte-rendu.

Ces éléments seront inclus dans un répertoire dont le nom est la concaténation de vos loggins et du mot **"-projet"**, et seront envoyés sur ESCALE au format d'archive zip.

Exercice : Compression de données au moyen de l'algorithme de Huffman

Dans ce projet, vous allez développer un compresseur (OU un décompresseur suivant le sujet qui vous est affecté) de données. L'idée de la compression de données est la suivante :

> Dans un fichier (texte par exemple), certains caractères apparaissent beaucoup plus souvent que d'autres. Or, la plupart des éditeurs de fichier représentent chaque caractère par un code de taille fixe, en général de 2 octets. L'idée du procédé de compression de données que nous allons utiliser consiste à représenter chaque caractère par un code de taille variable dépendant de la fréquence du caractère. Ainsi, un caractère apparaissant de nombreuses fois dans le fichier sera représenté par un petit nombre de bits. Cet procédé a été proposée par David Huffman en 1952.

1. Compression de données

Les grandes étapes de l'algorithme de compression sont :

1. Lecture du fichier à compresser, caractère par caractère (*cf* TD5), en comptant la fréquence de chaque caractère → production d'une liste *caractère-fréquence* (chaque élément de la liste est une cellule contenant un attribut `car` et un attribut `freq`).
   Exemple : si le texte est « nancy », nous avons l'information suivante en entrée :

00000000 01101110	00000000 01100001	00000000 01101110
n (code ascii :110)	a (code ascii :97)	n (code ascii :110)

00000000 01100011	00000000 01111001
c (code ascii :99)	y (code ascii :121)

soit un texte de 10 octets (80 bits). Les fréquences des caractères sont les suivantes :

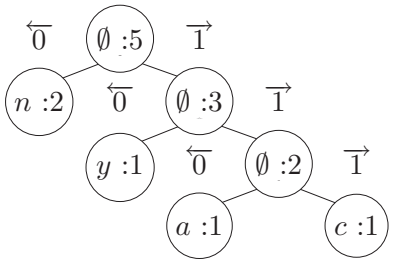
n	2
a	1
c	1
y	1

2. Cette liste de cellules est alors triée par ordre croissant de fréquence (*cf* TD7) :

a	1
c	1
y	1
n	2

3. Les cellules de la liste triée sont alors utilisées pour construire un arbre binaire comme vu au TD 6.

   Ce qui donne pour notre exemple :



4. L'arbre binaire est alors parcouru (*cf* TD6) en nommant le chemin menant à une feuille au moyen d'une suite de "0" ou de "1". Lors du parcours, si on descend à gauche, on colle "0" au nom du chemin, si on descend à droite, on lui colle "1" → création d'un dictionnaire de conversion associant à chaque caractère son nouveau code.
   Dans notre exemple, les caractères se voient attribués les codes suivants :

n	"0"
y	"10"
a	"110"
c	"111"

   On remarque au passage que le code associé à un caractère n'est jamais le préfixe du code associé à un autre caractère.

5. A partir de ce dictionnaire, nous pouvons convertir le texte contenu dans le fichier original.

   NB : attention, le nouveau code est dépendant du fichier compressé. Pour pouvoir décompresser le fichier résultant de la compression, il faut soit (a) disposer des informations de départ, en l'occurence la fréquence des caractères (afin de recalculer les codes des caractères), soit (b) copier le dictionnaire dans le fichier compressé (par exemple en début de fichier).

Nous choisirons l'option (a) expliquée en détails dans la section décompression ci-dessous.

Ainsi, le compresseur va écrire en début de fichier un entête contenant les fréquences de chaque caractère, puis remplira le nouveau fichier à partir du texte original, en écrivant bit à bit (*cf* TD5) le nouveau code de chaque caractère du texte de départ (certains de ces nouveaux codes nécessitent bien moins que 2 octets, d'où la compression).

Dans notre exemple :

(entête)	0	110	0	111	10
	n	a	n	c	y

soit un contenu de 10 bits (sans compter l'entête).

2. Décompression

Le problème principal de la décompression est de pouvoir récupérer (ou recalculer) le dictionnaire, afin d'être capable de retrouver pour chaque suite de bits, quel caractère lui correspond.

Nous vous proposons d'utiliser la technique notée (a) ci-dessus, et d'inclure dans le fichier compressé les fréquences des caractères de la façon suivante :

2 octets	2 octets	2 octets	...	n octets
(A)	(B)	(C)		(D)

où :

A est un entier contenant le nombre d'entrées dans l'entête (nombre de couples *caractère-fréquence*),

B est un caractère,

C est un entier (la fréquence du caractère),

D est le contenu du fichier compressé (point de départ pour la lecture bit à bit).

A, B et C appartiennent à l'*entête* du fichier, et D appelé le *corps* du fichier.

Ainsi, pour décompresser le fichier, vous procéderez comme suit :

1. Lecture des 2 premiers octets → récupération du nombre de couples *caractère-fréquence*.
2. Récupération, au moyen d'une boucle, de la table *fréquence-caractère*.
3. Calcul de l'arbre binaire de Huffman.
4. Parcours de l'arbre pour construire le dictionnaire.
5. Lecture bit à bit (*cf* TD5) du corps du fichier compressé, et création du fichier décompressé.

Bonus Quel(s) autre(s) moyen(s) de stocker l'information nécessaire à la décompression voyez vous ? En d'autres termes, quel(s) autre(s) format(s) d'entête proposez vous ?