Projet de Programmation C (IUP 2000/2001) *Revision* : 1.2

Roberto Di Cosmo

Date : 2000/12/0315 : 30 : 32

Le codage (compression) de Huffman.

Le codage de Huffman peut être utilisé pour compresser une suite d'éléments (par exemple un texte, qui est une suite de caractères) en remplaçant chaque élément par son code, qui est une séquence de bits de longueur variable. On s'intéressera ici à coder des suites de `char` lues à partir d'un fichier en clair et à decoder des suites de bits lues à partir d'un fichier compressé.

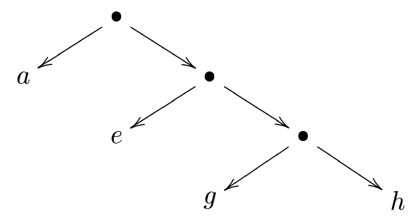
Préliminaires.

Voyons d'abord le décodage, qui est le plus simple à réaliser. Un "codage de Huffman" est constitué d'un ensemble de séquences binaires dont aucune n'est préfixe d'un autre. Cela corresponde très exactement aux branches d'un arbre binaire, si l'on décide de coder "gauche" avec 0 et "droite" avec 1 par exemple, et il est bien raisonnable que l'on garde sur chaque feuille les éléments dont le code correspond à la branche qui mène à cette feuille.

Plus exactement, étant donné un `chararbre` dont toutes les feuilles sont différentes, on peut coder les éléments qu'il contient de la manière suivante :

le chemin qui va de la racine de l'arbre à une feuille peut être représenté par la liste des choix 0 (*sous-arbre de gauche*) ou 1 (*sous-arbre de droite*) effectués à chaque nœud interne. Ceci définit une injection de l'ensemble des feuilles de l'arbre dans l'ensemble des listes de bits. Appelons *ensemble des codes valides* l'image de cette injection.

Par exemple, le dessin suivant représente un arbre avec des `char` sur les feuilles :



Pour cet arbre, la liste `[1; 1; 0]` est un code valide qui représente la feuille g. Par contre, les listes `[1; 1]` et `[1; 0; 0]` ne sont pas des codes valides.

Remarque importante

On ecrira nos fonctions de codage et decodage de façon à pouvoir les tester sur des listes, tout en pouvant les reutiliser plus tard sur des fichiers.
Pour cela, on vous demande d'ecrire votre code en utilisant, pour lire ou écrire un bit, l'interface suivante, que vous implementerez pour travailler sur des listes ou des fichiers selon le cas:

```
int getbit(int *b, entree e);
       /* cette fonction retourne un code d'erreur (par exemple -1) */
       /* s'il n'y a plus de bits en entree, et 0 sinon             */
       /* la valeur du prochain bit est placee dans b               */

void putbit(int b, sortie s);
       /* ajoute un bit au fond de la sortie, liste ou fichier      */
```

Le décodage.

- Écrivez une fonction `decode` qui, étant donnés un `chararbre` et une `intliste` L qui commence par un code valide, retourne l'élément codé et le reste de la liste (le suffixe de L qui n'a pas été utilisé pour trouver l'élément.) Par exemple, `decode` appliquée à l'arbre ci-dessus et la liste `[1; 1; 0; 0]` retourne g et `[0]`.
- Écrivez une fonction `decode_liste` qui, étant donnés un `chararbre` et une `intliste` obtenue par concatenation de codes corrects, retourne la `charliste` des éléments qu'elle code. Par exemple, `decode_liste` appliquée à l'arbre ci-dessus et la liste `[1; 1; 0; 0]` retourne la liste `[g; a]`.

L'arbre de Codage.

Le codage de Huffman est efficace si l'arbre de départ est bien choisi : il faut placer les éléments souvent rencontrés le plus proches possible de la racine. Définissez une structure `charintpaire` avec un champ entier et un champs char. Si on a une `charintpaire liste` de la forme `[(`$\texttt{elem}_1$`, `$\texttt{poids}_1$`); ...; (`$\texttt{elem}_n$`, `$\texttt{poids}_n$`)]`, où les $\texttt{elem}_i$ sont les éléments qui doivent apparaître dans l'arbre de Huffman, et les $\texttt{poids}_i$ représentent leur fréquence probables dans les textes à compresser, il est facile de construire un arbre de Huffman optimal contenant les $\texttt{elem}_i$.

- Définissez une structure `arbrepoids` avec un champ `int` et un champs `chararbre`. Écrivez une fonction `coerce` qui prend une `charintpaire liste` en argument et retourne la `arbrepoids liste` où chaque élément $\texttt{elem}_i$ est remplacé par l'arbre qui à un seul nœud contenant $\texttt{elem}_i$ et où les couples sont triées par poids croissants.

L'algorithme suivant permet de construire un arbre optimal :
On choisit dans la liste deux arbres a_1 et a_2 de poids p_1 et p_2 minimaux.
On construit l'arbre A suivant :

$$\begin{tikzcd} & \bullet \arrow[dl] \arrow[dr] & \\ a_1 & & a_2 \\ \vdots & & \vdots \end{tikzcd}$$

On lui donne le poids $p_1 + p_2$, et on considère alors la liste de départ, moins les couples (a_1, p_1) et (a_2, p_2), plus (A, $p_1 + p_2$) insérée comme il faut pour préserver l'ordre des poids. Cette nouvelle liste contient un élément de moins que la précédente. On applique à nouveau la transformation jusqu'à obtenir une liste réduite à un seul couple. Elle contient l'arbre optimal.

Attention reflechissez au cas dégéneré où tous les octets ont la même fréquence, et assurez-vous que votre réalisation de l'algorithme de construction de l'arbre de Huffman vous donne un arbre équilibré.

Attention il serait fort utile de rajouter un code special pour representer la fin de fichier. Pourquoi? Avec quelle fréquence?

Le projet se divise en plusieurs phases, ayant pour but la réalisation d'un veritable programme capable de compresser et decompresser des fichiers sur le disque.

calcul des frequences écrivez une fonction *analyse* qui lit un fichier et calcule, sous la forme d'une `charintpaire liste`, les fréquences rélatives de tous les octets qui y apparaissent

construction de l'arbre de codage programmez une fonction `make_huff` qui prend une `charintpaire liste` en argument, et qui retourne un arbre de Huffman optimal pour cette liste.

table de codage écrivez une fonction qui construit à partir de l'arbre un tableau *code[256]* contenant pour chaque octet possible son codage de Huffman dans l'arbre.

fonction de codage écrivez une fonction de codage qui prend en argument un arbre de Huffman, un pointeur sur un fichier d'entree et un pointeur sur un fichier de sortie, et écrit sur ce dernier le codage (compression) de Huffman du fichier d'entree, precede par une representation de l'arbre de codage.

fonction de decodage écrivez une fonction de codage qui prend en argument un pointeur sur un fichier d'entree et un pointeur sur un fichier de sortie, lit sur le fichier d'entree la representation de l'arbre de codage, puis écrit sur la sortie le decodage du fichier d'entree.

main une fois compile, le programme devra lire et écrire sur l'entrée et la sortie standard si appelé sans options, mais devra aussi reconnaître les options

- **-o file** envoyer la sortie sur le fichier file
- **-i file** lire l'entree sur le fichier file
- **-freq** affiche la table des frequences
- **-arbre** affiche l'arbre de codage
- **-table** affiche la table de codage

Modalités

dates limites le projet devra etre rendu au plus tard le merx credi 17 janvier, sous la forme d'un rapport imprimé contenant le code source et une description des vos choix de conception du programme.

groupes on autorise des groupes de 3 personnes au maximum

programme source on vous demande de rendre un programme C *bien commenté*, avec les instructions detaillées pour la compilation.

soutenance on organisera dans les horaires de TP des soutenances devant machine.