Travaux Pratiques de Structures de Données v2 n°5

Cours d'Informatique de Seconde Année

— Licence 2.2 —

Code de Huffman

L'objectif de ce TP est d'implémenter deux fonctions :
`int compression(char* fichier_a_compresser,char* fichier_compresse)`,
`int decompression(char* fichier_a_decompresser,char* fichier_decompresse)` qui compressent et décompressent des fichiers selon l'algorithme de Huffman.

▸ **Exercice 1.** *Implémenter une fonction* `int compter(char* nom,int T[256])` *qui initialise le tableau de la manière suivante : pour chaque lettre* `a` *du fichier, la case* `T[a]` *doit contenir le nombre d'apparitions de* `a` *dans le fichier* `nom`*. La fonction doit renvoyer le nombre de case non nulle de* `T` *ou -1 en cas d'erreur. On suppose que le tableau T est déjà alloué.*

```
typedef struct {
char lettre;
int occurence;
int gauche,droit;
}NoeudHuffman;
```

▸ **Exercice 2.** *Implémenter une fonction* `int fabriquer_arbre(NoeudHuffman* arbre,int T[256],int nb)` *qui fabrique l'arbre de Huffman à partir du poids des lettres contenus dans le tableau* `T` *et renvoie le nombre de noeud de l'arbre.* `nb` *indique le nombre de cases non nulles de* `T`*. La fonction doit renvoyer -1 en cas d'erreur. Pour représenter les lettres on utilisera le type* `unsigned char`*. Le tableau arbre est supposé déjà avoir été alloué, il doit contenir* `2*nb-1` *cases. Le tableau T est sensé avoir été initialisé avec* `compter`*.*

```
typedef struct {
char *code;
int nombreBit;
}codage;
```

▸ **Exercice 3.** *Implémenter une fonction*
`void fabriquer_code(codage C[256],NoeudHuffman* Arbre,int nb_noeud)` *qui, pour chaque lettre* `a` *présente dans l'arbre, initialise la case* `C[a]` *avec le code de Huffman de la lettre* `a`*. Le code est représenté par un tableau de* `char` *ou chaque case non nulle représente le bit* `1` *et chaque case nulle le bit* `0`*.*

▸ **Exercice 4.** *Implémenter une fonction* `int coder_arbre(NoeudHuffman* arbre,FILE_b* f` *qui code le parcours préfixe de l'arbre de Huffman avec des bits et des codes ASCII et qui l'écrit dans le fichier* `f`*. On utilisera les fonctions implémentées dans le fichier* `manip_bit.c` *dont une description est donnée à la fin du sujet.*

▸ **Exercice 5.** *Implémenter la fonction* `int compression(char* a_compresser,char* compresse)`.

▸ **Exercice 6.** *Implémenter une fonction* `int fabriquer_cle(FILE_b* f,Arbre* a)` *qui fabrique l'arbre d'Huffman à partir de la suite de bit qui le code. La fonction renverra la hauteur maximal de l'arbre (qui est la taille maximum d'un code) ou -1 en cas d'erreur. Pour la structure d'arbre binaire on utilisera la structure que l'on veut (*`Noeud*`*, tableau, ...).*

▸ **Exercice 7.** *Implémenter une fonction* `int decoder_bit(unsigned char* c,Arbre a,FILE_b* f)` *qui lit une suite de bit dans* `f` *et s'arrête lorsque la suite code un caractère selon l'arbre d'Huffman* `a`*. La fonction initialise* `*c` *par la valeur du caractère décodé. La fonction renvoie le nombre de bit lues (dans le cas ou la fin du fichier est atteinte elle doit donc renvoyer 0).*

▸ **Exercice 8.** *Implémenter la fonction* `int decompression(char* a_decompresser,char* decompresse)`.

A Annexe

Voici les entêtes des fonctions d'écriture et lecture de fichier bit par bit.

- `FILE_b` est un type qui permet de manipuler les fichier bit à bit. Il fait le lien entre le programme et le fichier sur lequel on opère. On ne manipulera que des pointeurs sur `FILE_b` et jamais directement des valeurs pointées. On récupère une variable de type `FILE_b*` grâce à `b_open`.
- `FILE_b* b_open(char* nom,int lecture_ecriture);`
  est une fonction qui ouvre un fichier dont le nom est donné par la chaine `nom`.
  Si `lecture_ecriture` vaut 1 le fichier est ouvert en écriture (penser que 1 ressemble à la première lettre de in). Le fichier est effacé s'il existe. S'il n'existe pas il est créé vide.
  Si `lecture_ecriture` vaut 0 alors le fichier est ouvert en lecture seul et le curseur se place au début du fichier (penser que 0 ressemble à la première lettre de out).
- `void b_close(FILE_b *f);`
  Ferme le fichier ouvert avec `b_open`.
- `int lire(FILE_b* f);`
  Lit le prochain bit du fichier et le retourne. Si la fin du fichier est atteinte retourne -1.
- `int ecrire_b(FILE_b * f,int bit);`
  Écrit un bit dans le fichier. Si `bit` est nul la fonction écrit 0 sinon la fonction écrit 1. Si le fichier dans lequel on tente d'écrire est en lecture seul la fonction renvoie -1 sinon elle renvoie 1.