Projet - Construction du code de Huffman

L'algorithme de Huffman sert à compresser des textes. Cet algorithme se déroule en quatres phases successives.

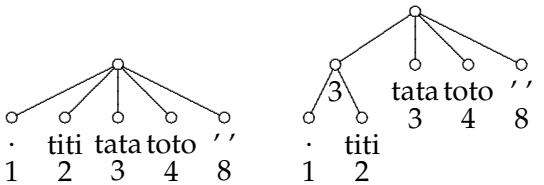
Nous allons utiliser le "texte" suivant pour illustrer les explications :
toto toto tata titi toto tata titi toto tata.

Première phase :
L'algorithme comptabilise le nombre d'occurrences de chaque "mot". Dans notre cas, un mot est :
- soit une succession de lettres ('a' à 'z' et 'A' à 'Z' et lettres accentuées) et de chiffres ('0' à '9'). Ainsi L33tg4M3r sera considéré comme un mot par notre algorithme.
- soit un autre caractère ('.', ' ' (espace), '?', retour chariot, *etc*.). Ainsi, trois espaces successifs comptent pour trois mots différents.

Notre exemple renvoie donc :
"toto" apparaît 4 fois.
"tata" apparaît 3 fois.
"titi" apparaît 2 fois.
" " apparaît 8 fois.
"." apparaît 1 fois.

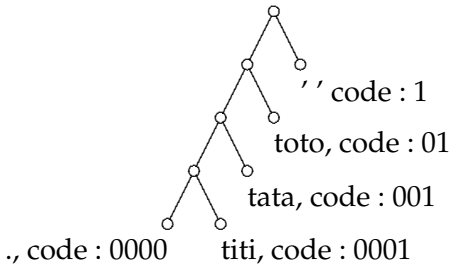
Deuxième phase :
Initialisation : chaque mot est transformé en un arbre composé d'une seule feuille. Cette feuille est labelée par le mot correspondant. Le poids d'un arbre est le nombre d'occurrences du mot contenu dans la feuille.
Tant qu'il existe plusieurs arbres : choisir deux arbres de poids minimaux et les fusionner en rajoutant un sommet de degré 2 dont les deux sous-arbres correspondent aux deux arbres de poids minimaux. Le poids du nouvel arbre est équivalent à la somme des poids des deux sous-arbres.

La première étape de notre exemple génère l'arbre suivant :



Troisième phase :
Les arêtes reliant un fils gauche à son père sont labelées 0, celles reliant un fils droit à son père sont labelées 1. Chaque mot est associé à un code correspondant au mot binaire obtenu en allant de la racine à la feuille contenant le mot.

Notre exemple génère le code suivant :



Quatrième phase :
Chaque mot du texte est remplacé par son code. Le texte est ainsi compressé car les mots les plus fréquents dans le texte sont associés à des codes courts. Les codes longs sont associés à des mots peu usités et donc consomment, au final, peu de place.

Notre exemple, une fois compressé "toto toto tata titi toto tata titi toto tata." devient :
01 1 01 1 00,1 1 0001 1 0,1 1 001 1 00,01 1 01 1 00,1 0000 (5 octets).

Première remarque : un mot est fait de caractères qui sont encodés sur un octet soit 8 bits. Son code est un mot binaire dont un caractère n'occupe qu'un seul bit.

Deuxième remarque : Pour décompresser le fichier, il suffit de lire les bits tout en descendant dans l'arbre. Quand une feuille est atteinte, on écrit le mot correspondant sur la sortie et on remonte à la racine. Il est donc nécessaire d'encoder l'arbre dans le fichier compressé. Mais la taille de l'arbre est généralement faible comparaît au reste du fichier.

Le but de ce projet est de créer un programme qui va lire un fichier texte dont le chemin est donné dans la ligne de commande appelant le programme. Puis, le programme calcule le code de Huffman des mots apparaissant dans le texte contenu par ce fichier.

Exercice 1. *Traitement de l'entrée & Lecture d'un fichier*
Le but de cette partie est d'ouvrir le fichier passé dans la ligne de commande, de lire son contenu et de repérer les différents mots qui le composent.

La fonction **char * getWord(FILE* input, char* buffer)** utilise le buffer donné en argument et renvois un pointeur vers une zone mémoire, alloué par cette fonction et de la taille du mot (+1 - pour le caractère de fin). Lorsque qu'il n'y aura plus de mot à lire la fonction renverra la valeur $NULL$.

Remarque : Le buffer sera alloué qu'une seule fois.
Le programme va ouvrir le fichier et selon le paramètre (cf. exemple ci-dessous) va appeler une fonction (wordReader pour l'exercice 1). Et appellera jusqu'à la valeur $NULL$ la fonction **getWord()** et placera les mots dans une liste chaînée via la fonction **addList**.

Rappel : pour ouvrir un fichier, il faut utiliser la commande **FILE* fopen(char* chemin,char * option)** qui ouvre le fichier indiqué par la chaîne de caractère *chemin* en lecture avec l'option "r" et écriture avec l'option "w". Ensuite, il faut utiliser la commande char **fgetc**(FILE* fp) pour lire le prochain caractère contenu dans le flux fp. Le caractère EOF est renvoyé quand la lecture du fichier est terminée.

Dans un deuxième temps le programme affichera tout les mots de la liste.
L'utilisateur exécutera le programme *huffman* grâce à une ligne de commande du type :

./huffman -w fichier_entrée

-w : (word) indique qu'il faut afficher les mots lu dans le fichier (tout le texte).
fichier_entrée : lien vers le fichier à lire.

Exercice 2. *Comptage des occurrences des mots*

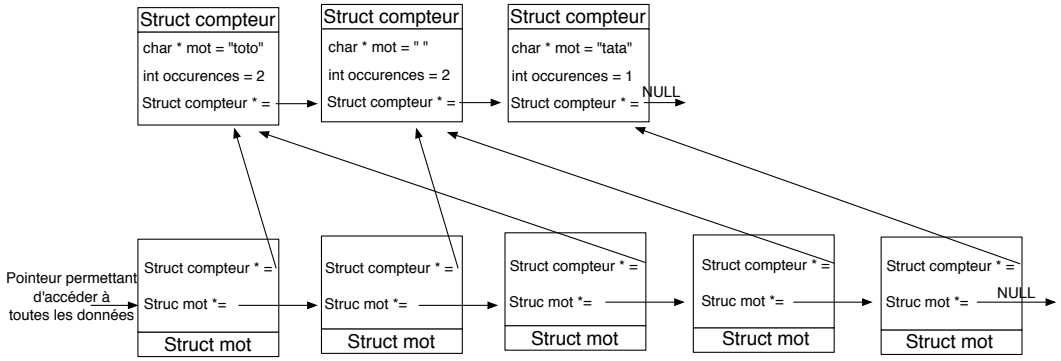
Le programme devra afficher chaque mot une fois avec le nombre d'occurrence.
Le paramètre du programme sera -wc pour word count et le nom de la fonction appelée sera **struct mot* wordCount(FILE * input)** fonction qui renverra le pointeur de début de votre structure.
Définir une structure **compteur_mot** qui comptabilise le nombre d'occurrences d'un mot. Les champs sont :

- char* **mot** : mot correspondant au compteur,
- int **occurrences** : nombres d'occurrences du mot,
- STRUCT compteur_mot* **suivant** : pointeur vers le compteur suivant. Vous utiliserez une structure de liste chaînée pour représenter les compteurs.
- pour la partie suivante, vous êtes ensuite libres de rajouter d'autres champs si nécessaire.

Le texte lu en entrée sera mémorisé sous la forme d'une liste chaînée. Pour cela, définissez une structure **mot** dont les champs sont :

- STRUCT compteur_mot* **mot** : pointeur vers le compteur correspondant au mot lu,
- STRUCT mot* **suivant** : pointeur vers le mot suivant.

Après la lecture successive des mots "toto", " ", "toto", " " et "tata", l'espace mémoire ressemble à ceci :



Remarque : pour accéder à toutes ces variables, il suffit de mémoriser un pointeur vers le premier mot lu.

Exercice 3. *Création de l'arbre et du code*

Maintenant que vous disposez de la liste des compteurs, créez l'arbre de Huffman lui correspondant et générez les codes des différents mots. La structure à employer ainsi que la méthode pour trouver les deux arbres de poids minimaux sont laissées à votre choix.

Pour cela creez la fonction **tree * huffman(struct mot)** renvoyant l'arbre de huffman avec le mot et son code dans les noeud. Utilisez le paramètre "-t" (tree) pour afficher le code de chaque mots.

Exercice 4. *Questions facultatives*

Terminez le programme :

- compressez le texte à partir du code obtenu,

– sauvegardez le texte compressé dans un fichier,
– encodez également l'arbre en plus du texte compressé dans le fichier,
– gérez la décompression d'un fichier encodé grâce à votre programme.

N'oubliez pas de faire un fichier README et Makefile avant de rendre votre projet.
Bonne chance.