CODAGE DE HUFFMAN ET COMPRESSION

Si on se limite à des caractères simples (code ASCII étendu latin-1, par exemple), les caractères d'un texte sont codés sur 8 bits. Donc un texte de n caractères formera une suite de $8n$ bits.

Le codage de Huffman est une façon de compresser le fichier qui contient le texte : plus un caractère apparaît souvent dans le texte, plus on le représente par une suite courte de bits. Ainsi on peut espérer obtenir la même information mais stockée sur un nombre de bits plus petits.

1 Codage de Huffman

On considère un texte contenant des caractères codés sur p bits : pour nous, $p = 5$ pour qu'avec $2^5 = 32$, on puisse coder un texte non accentué en minuscule avec des espaces, des virgules, des points, des apostrophes, des tirets, des points-virgules.

Dans ce TP, les arbres binaires seront hétérogènes, autrement dit on distingue les feuilles et les nœuds (pas d'arbre vide dans cette description)

```
type arbre = F of int * char | N of int * arbre * arbre;;
```

Les feuilles contiendront deux informations : un caractère et un entier (qu'on appelle poids). Le caractère sera l'un des 32 caractères disponibles et le poids sera son nombre d'occurrences dans le texte.

Les nœuds contiendront le poids de l'arbre dont ils sont la racine (autrement dit, la somme des poids de leurs feuilles).

Pour fusionner deux arbres a_0, a_1 de poids n_0, n_1, on crée simplement un arbre dont le nœud racine contiendra la somme des poids $n_0 + n_1$ et dont les fils gauche et droit sont a_0 et a_1.

On construit l'arbre de Huffman de la façon suivante :

- on calcule les occurrences de tous les caractères possibles dans le texte et on construit la liste des feuilles ;
- tant qu'il reste au moins deux arbres dans la liste, on fusionne deux arbres qui ont un poids minimum ;
- l'arbre de Huffman est le dernier arbre restant dans la liste.

Une fois l'arbre construit, on peut alors coder chaque caractère du texte. Son code de Huffman se calcule de la façon suivante : dans l'arbre de Huffman ainsi construit, on suit le chemin de la racine jusqu'à la feuille contenant le caractère : chaque fois qu'on part à gauche, on note "0" et on note "1" si on part à droite. Le code de Huffman d'un caractère est donc une suite de bits.

Vous trouverez dans le répertoire Documents un fichier `huffman-source.ml` qui contient deux fonctions, l'une qui transforme un caractère parmi les 32 disponibles en son numéro, entier compris entre 0 et 31, et l'autre qui est sa réciproque, ainsi que le type arbre et une fonction utilitaire de calcul de poids.

```
let car2int c =
 match c with
 | ' ' -> 26
 | ',' -> 27
 | ';' -> 28
 | '.' -> 29
 | ''' -> 30
 | '-' -> 31
 | _ -> int_of_char c - 97;;

let int2car x =
 match x with
 | 26 -> ' '
 | 27 -> ','
 | 28 -> ';'
 | 29 -> '.'
 | 30 -> '''
 | 31 -> '-'
 | _ -> char_of_int (x +97);;
```

Écrivez les fonctions suivantes (entre autres) :

1) occurrences s de type string -> int vect qui calcule le tableau de longueur 32 qui contient dans chaque case i le nombre d'apparitions du caractère numéro i dans la chaîne s.

2) init s de type string -> arbre list, qui calcule la liste des feuilles initiales

3) mini l de type arbre list -> arbre, qui trouve un arbre de poids minimal dans une liste d'arbres.

4) fusionner a b de type arbre -> arbre -> arbre, qui calcule l'arbre obtenu par fusion de deux arbres.

5) huffman s de type string -> arbre, qui calcule un arbre de Huffman associé à la chaîne s.

6) codes s de type string -> int list vect, qui prend en paramètre la chaîne s, calcule en interne un arbre de Huffman associé et calcule finalement le tableau contenant dans chaque case i le codage du caractère numéro i (sous forme d'une liste de 0 ou de 1).

Remarque : on aurait même pu être plus efficace : si le caractère numéro i n'apparaît même pas dans la chaîne, alors on laisse la case i du tableau vide, puisque ce code ne servira jamais dans la compression du texte.

2 Compression

Maintenant qu'on sait coder les caractères de la chaîne s, écrivez la fonction de compression qui transforme la chaîne s en **une liste de** 0 **et de** 1 : vous vérifierez que la liste obtenue est bien de longueur inférieure à $5n$ (n étant la longueur de la chaîne). Attention : on ne veut pas obtenir une liste de listes de 0 et de 1, tout doit être aplati en une liste !

Un peu plus technique : écrivez la fonction de décompression qui permet de retrouver le texte initial à partir de la liste des 0 et 1 codant le texte compressé et l'arbre de Huffman qui a servi à la compression.

En pratique, le texte initial est dans un fichier, le fichier produit par l'algorithme contient deux choses : le texte compressé et l'arbre qui a servi à la compression, de sorte que la fonction de décompression peut facilement retrouver le texte initial.

De plus, l'arbre est construit de manière différente pour obtenir de meilleurs résultats de compression et pour minimiser le temps de calcul des codes de chaque lettre.