EISTI. **Année scolaire 2008-2009**

DÉPARTEMENT " INFORMATIQUE "

THÉORIE DE L'INFORMATION

Série d'exercices N^{o}3

PARTIE I. CODAGE DE HUFFMAN

Nous allons considérer une source d'alphabet

$$\Omega = \{a, b, c, d, e, f, g, h, i, j\}$$

et un canal d'alphabet binaire $\{0, 1\}$. Voici les définitions importantes.

Théorème de la borne inférieure de longueur de code

Théorème 0.1. *Soit une source S d'alphabet $\Omega_S = \{s_1, \ldots, s_n\}$ de taille n et de distribution de probabilités $P_S = \{p_1, \ldots, p_n\}$. Soit un canal d'alphabet binaire $\Omega_C = \{0, 1\}$ sans bruit, stationnaire et sans mémoire. Soit un code déchiffrable $\{m_1, \ldots, m_n\}$ de longueurs de mots $\{l_1, \ldots, l_n\}$.*
Alors la longueur moyenne de mots de code vérifie :

$$\bar{L} = \sum_{i=1}^{n} p(s_i) l_i \geq H(S)$$

L'égalité n'est possible que si $\forall i = 1, \ldots, n, \;\; p_i = 2^{-l_i}$**.**

Code absolument optimal C'est un code dont la longueur moyenne de mots est égale à la borne inférieure, $H(S)$.

Code optimal. Un code est dit optimal dans une certaine classe de codes si sa longueur moyenne de mots est minimale dans cette classe. La classe de codes la plus importante est celle de codes sans préfixe.

Exercice 1. Soit une source d'alphabet Ω et de distribution de probabilité suivante

s_i	a	b	c	d	e	f	g	h	i	j
m_i	0.2	0.05	0.1	0.05	0.15	0.05	0.1	0.05	0.15	0.1

1. Quelle est la longueur moyenne minimale pour un code binaire de cette source ?
2. Existe-t-il un code absolument optimal ?
3. Construire un code sans préfixe optimal selon la méthode de Huffman.
4. Représenter ce code sous forme d'arbre.

PARTIE II. PRÉPARATION ALGORITHMIQUE.

Cette partie a pour objectif de préparer la réalisation en scilab de l'algorithme de construction d'un code de Huffman.

Exercice 2. On reprend la même source que dans l'exercice 1. En appliquant la méthode de Huffman dans l'exercice précédant nous avons effectué deux parcours de l'arbre de code : d'abord des feuilles vers la racine et ensuite de la racine vers les feuilles.

Il est possible d'obtenir l'arbre de code directement lors du premier parcours. Pour cela nous allons rappeler la définition réccursive d'un arbre binaire : *un arbre binaire est soit vide, soit un triplet* (R, G, D) *où* R *est la racine et* G *est un arbre binaire, appelé sous-arbre gauche, et* D *est un arbre binaire, appelé sous-arbre droit.*

Nous allons en plus associer à la racine d'un arbre binaire un mot d'alphabet Ω et un poids (une probabilité). Il est alors possible d'effectuer l'algorithme de Huffman de façon équivalente en manipulant un ensemble d'arbres binaires au lieu d'un alphabet.

Pour initialiser l'algorithme, on associe à chaque symbole de l'alphabet Ω un arbre-feuille (composé d'une seule racine) de poids égal à la probabilité du symbole. Ensuite, à chaque "fusion" de symboles on crée un nouvel arbre de poids égal à la somme des poids de ses sous-arbres. La figure ci-dessous illustre ce principe.

Refaire l'algorithme de Huffman avec un ensemble d'arbres pour la source de l'exercice 1.

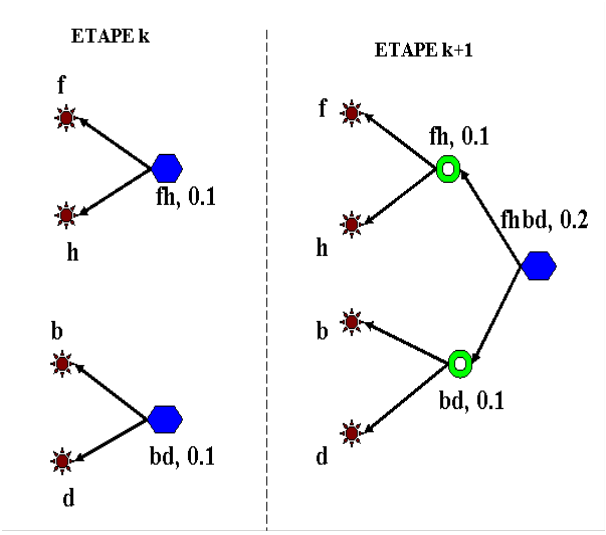

FIGURE 1 – Un exemple de fusion d'arbres

Exercice 3. Ecrire le pseudocode de l'algorithme de Huffman par fusion d'arbres, en supposant que les types abstraits "Arbre Binaire" et "Foret ordonnée" ont été définis. En particulier, on suppose que le type "Arbre binaire" dispose de fonctions

creerArbre(CHAINE chaineRacine, REEL poidsRacine, ARBRE sousArbreGauche, ARBRE sousArbreDroit) constructeur

creerFeuille(CHAINE caract, REEL poids) constructeur pour un arbre qui n'a qu'une racine

getPoidsRacine(A) accesseur, renvoie le poids de la racine

getPoidsRacine(A) accesseur, renvoie la chaîne de la racine

et que le type abstrait "Foret ordonnée" contient des fonctions suivantes

creerForetOrdonneeVide() Constructeur, définit une forêt vide

ajoutArbreForetOrdonnee(FORET F, ARBRE A) ajoute une arbre A dans l'ordre décroissant des poids de racine

supprimeDernier(foret) supprime le dernier arbre de la forêt, renvoie l'arbre supprimé

Exercice 4. ATTENTION ! Les résultats de cet exercice représentent la partie code du livrable 1 du projet de pôle "Informatique théorique". Le travail débuté pendant ce TP doit être terminé et déposé sur AREL avec un rapportAREL assorti avant la date limite fixée sur AREL. Vous trouverez les instructions détaillées sur le contenu du rapport sur la page d'activité du projet.

Un ensemble de fonctions scilab, regroupées dans le fichier ArbresForets.sci, vous est fourni. Ces fonctions permettent de construire et manipuler les arbres binaires et des ensembles d'arbres, appelés forêts. Chaque arbre est défini par quatre éléments :

1. Poids de la racine : nombre réel
2. Chaîne de caractères associée à la racine
3. Sous-arbre gauche
4. sous-arbre droit

Dans le cas particulier d'arbres qui n'ont qu'une racine (des feuilles) les deux derniers paramètres sont des constantes.

Les fonctions fournies utilisent les structures de scilab , tlist. Vous pouvez trouver tous les renseignements nécessaires sur ces structures dans l'aide de scilab. Cependant, aucune connaissance de tlist n'est nécessaire pour utiliser les fonctions fournies. Vous pouvez les manipuler comme une mini librairie qui réalise en scilab les types abstraits "Arbre Binaire" et "Forêt Ordonnée" avec un minimum de fonctions.

1. Ecrire une fonction qui construit le code de Huffman. Elle doit prendre en entrée deux tableaux :
   (a) tableau de n caractères, l'alphabet ;
   (b) tableau de n réels, la distribution de probabilités associée

   La fonction doit renvoyer la table de code, associant à chaque caractère une séquence de bits sous forme de chaîne de caractères. Pour réaliser cette fonction, inspirez vous de l'algorithme de fusion d'arbres vu dans l'exercice précédent
2. Ecrire une fonction qui calcule l'entropie d'une source à partir de sa distribution de probabilités
3. Ecrire une fonction qui calcule la longueur moyenne des mots de code à partir de la table de code
4. Ecrire une fonction qui encode un message composé de caractères de l'alphabet donné en utilisant une table de code.

PARTIE III. POUR RÉVISER OU S'ENTRAÎNER.

Exercice 5. Soit une source d'alphabet Ω et de distribution de probabilité suivante

s_i	a	b	c	d	e	f	g	h	i	j
m_i	0.0625	0.125	0.0625	0.125	0.0625	0.125	0.0625	0.0625	0.25	0.0625

1. Calculer l'entropie de la source.
2. Montrer qu'il existe un code binaire **absolument optimal** pour cette source.
3. Quelles sont les longueurs des mots d'un tel code pour chaque caractère ?
4. Construire un code sans préfixe de longueurs correspondantes en utilisant un arbre binaire.