TP APPLICATION DES AB : LE CODAGE DE HUFFMAN

Le codage de Huffman est une méthode de compression de données inventée par David Albert Huffamn en 1952 et qui permet de réduire la longueur de codage d'un texte. Il repose sur l'utilisation d'un arbre binaire.

On appelle *alphabet* l'ensemble des symboles (ou caractères) composant la donnée à compresser. Nous utiliserons un *alphabet* réduit composé des huit lettres A, B, C, D, E, F, G et H.

Partie A (ANA)

On cherche à coder chacun des caractères précédents par une séquence de chiffres binaires.

1. a. Combien de bits sont nécessaires pour coder ces huit lettres ?

b. Quelle est la longueur en octets d'un message de 1000 caractères construit avec cet alphabet ?

2. Proposer un code de 3 bits pour chacun de ces caractères.

3. On considère maintenant le codage suivant, la longueur du code de chaque caractère étant cette variable :

Lettre	A	B	C	D	E	F	G	H
Code	10	001	000	1100	01	1101	1110	1111

a. En utilisant la table précédente, déterminer le message correspondant au code 001101100111001.

Ce type de code est dit *préfixe* ; aucun code n'est le préfixe d'un autre. Cette propriété permet de séparer les caractères de manière non ambigüe.

b. Donner le code correspondant au mot *CACHE*. Combien de bits utilise-t-il ? Combien en aurait-il fallu avec le codage de la question 1. ?

4. Dans un texte, chacun des caractères a un nombre d'apparitions différent. On a résumé ce nombre d'apparition dans le tableau suivant :

Lettre	A	B	C	D	E	F	G	H
Code	240	140	160	51	280	49	45	35

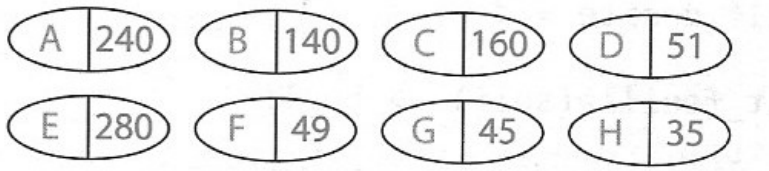
a. En utilisant le code de la question 1. Quelle est la longueur en bits de ce message ?

b. Même question en utilisant le code de la question 2.

<u>Partie B</u> (REA)

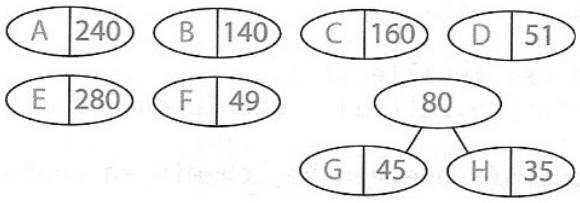
L'objectif du codage de Huffman est de trouver le codage proposé dans la partie A qui minimise la taille en nombre du bits du message codé, en se basant sur la fréquence d'apparition de chaque caractère (on attribue les codes les plus courts aux caractères apparaissant le plus souvent).

Pour déterminer le code optimal, on considère 8 arbres, réduits à une racine, contenant le caractère et son nombre d'apparitions :



On fusionne les deux arbres contenant le plus petit nombre d'apparitions (correspondant ici aux caractères G et H) : on affecte à la valeur de la racine de cet arbre le nombre total d'apparitions, et on choisit de mettre en sous-arbre gauche l'arbre ayant le plus grand nombre d'apparitions, et en sous-arbre droit celui ayant le plus petit nombre d'apparitions (on pourrait choisir de faire l'inverse). En cas d'égalité, le positionnement des sous-arbres n'a pas d'importance.

On obtient alors les sept arbres suivants :



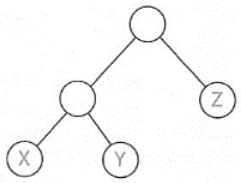
On recommence jusqu'à n'avoir qu'un seul arbre binaire.

1. Combien d'étapes sont ici nécessaires pour obtenir l'arbre final ?

2. Construire cet arbre de Huffman.

3. Le code à attribuer à chaque caractère est déterminé de la manière suivante : on suit le chemin à parcourir dans l'arbre pour arriver au caractère, et on note 0 quand on descend par le fils gauche, 1 quand on descend par le fils droit.

Dans l'exemple ci-contre, X est alors codé 00. Quel sont les codes de Y et Z ?



Donner le code de chacun des huit caractères dont on a construit l'arbre de Huffman.

<u>Partie C</u> (REA)

Le code donné **arbre_huffman.py** permet, à partir d'un fichier nommé *texte.txt*, de construire l'arbre de Huffman puis un dictionnaire qui associe à chaque caractère du fichier d'entrée son code sous forme d'une séquence de bits.

Compléter les parties en ################ de ce code.

Bilan des compétences travaillées

	A	B	C	D
ANA	Décrire et Spécifier les caractéristiques d'un processus, les données d'un problème, ou celles manipulées par un algorithme ou une fonction. (Partie A)			
	Trouver tous les nombre de bits des questions 3. et 4.	*Trouver tous les nombres de bits, avec une aide ponctuelle.*	*Nombre de bits trouver avec de l'aide.*	*Pas les bonnes réponses à 3. et 4.*
REA	Décrire une structure de données permettant de résoudre le problème. (Partie B)			
	Arbre de Huffman construit et application Y et Z.	*Arbre de Huffman construit et application Y et Z avec une aide ponctuelle.*	*Arbre construit avec de l'aide.*	*Pas d'arbre de Huffman.*
REA	Mettre en œuvre une solution, par la traduction d'un algorithme ou d'une structure de données dans un langage de programmation ou un langage de requête. (Partie C)			
	Tout est complété sans aide. Le code fonctionne correctement.	*L'ensemble est complété avec une aide ponctuelle. Le code fonctionne correctement.*	*Incomplet.*	*Rien n'est complété.*