L'objectif est de faire un programme permettant de coder ou décoder un texte à l'aide de l'algorithme d'Huffman. Le problème va être décomposer en sous-problèmes conduisant à la création de différentes fonctions. Les messages à coder seront donnés sous forme de chaîne de caractères.

1 Arbre d'Huffman

1. Commencer par créer la fonction **tableFrequences** qui étant donné un texte au format chaîne de caractère retourne un dictionnaire ayant pour clés les caractères rencontrés et pour valeurs chacune de leurs occurrences. Le prototype de la fonction sera **def tableFrequences**(texte): et l'appel de **tableFrequences**('abracadabra') renvoie
   {'c': 1, 'a': 5, 'd': 1, 'r': 2, 'b': 2}
2. Créer la fonction **arbreHuffman** qui étant donné un dictionnaire construit par la fonction précédente retourne une représentation de l'arbre de Huffman correspondant (faire attention du cas d'une chaîne n'ayant qu'un seul caractère). Le prototype de la fonction sera le suivant **arbreHuffman** (occurrences): et l'appel de
   arbreHuffman({'c': 1, 'a': 5, 'd': 1, 'r': 2, 'b': 2}) devra retourner un dictionnaire de la forme : {0: 'a', 1: {0: 'r', 1: {0: {0: 'c', 1: 'd'}, 1: 'b'}}} dans lequel chacun des dictionnaires a deux clés, 0 ou 1 prenant pour valeur soit un caractère, soit un dictionnaire correspondant aux sous-arbres de l'arbre binaire. (aux permutations près des lettres de même poids (d et c; b et r))

2 Codage des caractères

Créer la fonction **codeHuffman** qui étant donné dictionnaire correspondant à un arbre de Huffman construit par la fonction précédente retourne un dictionnaire où les clés sont les chaînes binaires et les valeurs les caractères correspondants. Le prototype de la fonction sera le suivant **def codeHuffman**(arbre):. L'appel de **codeHuffman**(arbre) avec arbre= {0: 'a', 1: {0: 'b', 1: {0: {0: 'c', 1: 'd'}, 1: 'r'}}} devra retourner un dictionnaire de la forme suivante {'111': 'r', '1100': 'c', '10': 'b', '1101': 'd', '0': 'a'}.
La fonction appelera la fonction récursive **codeHuffmanParcours** dont le prototype sera le suivant: **def codeHuffmanParcours**(arbre,prefixe,code):, avec prefixe qui sera successivement égal à '', '1' ...

3 Codage et décodage d'un message

On peut désormais créer une fonction **encodage** qui étant donné un code de Huffman construit par la fonction précédente et le texte initial retourne la chaine de bits produite par le codage de Huffman. Le prototype de la fonction sera le suivant **def encodage**(code,texte): , et l'appel de **encodage**(C,'abracadabra') avec C le code précédent avec
arbre={0: 'a', 1: {0: 'b', 1: {0: {0: 'c', 1: 'd'}, 1: 'r'}}}, devra retourner la chaîne binaire suivante '01011101100011010101110'.
Et la fonction **decodage** qui étant donnés un code de Huffman et un texte binaire compressé retourne le texte initial. Le prototype de la fonction sera le suivant **def decodage**(code,texteBinaire): et l'appel de **decodage**(C,'01011101100011010101110')avec
avec C le code précédent obtenu avec arbre={0: 'a', 1: {0: 'b', 1: {0: {0: 'c', 1: 'd'}, 1: 'r'}}}, devra retourner la chaîne binaire suivante 'abracadabra'