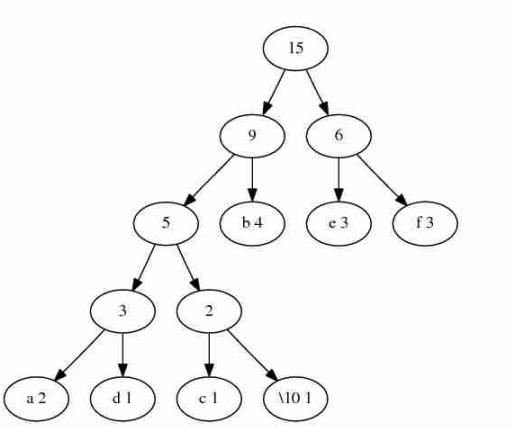
Projet 1 : compression par codage de Huffman

Un codage de Huffman est construit à partir d'un arbre issu de l'analyse des fréquences d'apparition des caractères d'un texte. L'arbre ci-dessous par exemple est construit à partir des fréquences suivantes :
a 2, d 1, c 1, \n 1, b 4, c 3, f3



Pour obtenir le codage d'un caractère *x*, on établit le chemin de la racine à la feuille portée par *x* en étiquetant chaque arête par un 0 s'il s'agit d'un lien vers un fils gauche et par 1 sinon.

1. Dans un fichier `encodage.c`, écrivez une fonction

```
void analyseFichier(FILE *fp, int freq[256]);
```

qui remplit le tableau `freq` par le nombre d'occurrences de chaque caractère. On suppose le fichier associé à `fp` déja ouvert. Ainsi, si dans le fichier pointé par `fp`, le caractère 'a' apparaît 13 fois, on aura `freq['a'] = 13`.

2. Donnez le fichier `arbre.c` correspondant au fichier d'entête `arbre.h` suivant :

```
#ifndef __ARBRE_H_
#define __ARBRE_H_

#include <stdio.h>
#include "bool.h"

// 0 : fils gauche, 1 : fils droit
typedef struct noeud {
      unsigned char c;
      int freq;
      struct noeud *fg, *fd; // fils gauche, fils droit
} *Arbre;

Arbre creerFeuille(unsigned char c, int freq);
Arbre enrac(Arbre a_gauche, Arbre a_droit); // freq = freq de a_gauche + freq
a_droit

// selecteurs et tests élémentaires
Arbre fg(Arbre);
Arbre fd(Arbre);
int frequenceRacine(Arbre);
bool estVide(Arbre);
bool estFeuille(Arbre);
```

```
unsigned char carRacine(Arbre);

// affichage infixe d'un arbre (fg - racine - fd)
void afficheArbre(Arbre);

// construction de l'arbre de huffman à partir d'un tableau de fréquence
Arbre construitArbre(int freq[256]);

#endif
```

3. Ecrire une fonction encode d'encodage des caractères qui à partir d'un arbre de Huffman et d'un caractère produit la séquence de bits du code de Huffman.

Ecrire la fonction decode qui réalise l'opération inverse, c'est-à-dire qui donne le caractère à partir du code de Huffman et de l'arbre.

4. Ecrivez deux programmes principaux de compression et de decompression (fichiers `compression.c` et `decompression.c`). Pour cela, récupérez sur la page du sujet les fonctions

```
unsigned char litNiemeBit(unsigned char *buff, int n);
void ecritNiemeBit(unsigned char *buff, unsigned char bit, int n);
```

qui lisent/écrivent le `n`-ieme bit dans un buffer. Un bit est représenté par un `unsigned char` de valeur 0 ou 1, `n` indique la position du bit dans `buff`, le bit de poids fort dans l'octet `buf[0]` est le bit de position 0. Ainsi n est dans l'intervalle [0 ; (taille de buf en octets*8)-1].

Par ailleurs, le programme de compression affichera le taux de compression (taille fichier comp. / taille fichier initial).