Projet Huffman - 2018

Canta Thomas[1] / Desgenetez Charles / Reiter Maxime

December 2018

[1]Listing fait avec Overleaf

Abstract

Voici le listing des fonctions du projet Huffman sous format pdf, Doxygen nous a donné du fil a retordre, on a donc décidé de le faire en LaTeX pour rendre **au plus propre et au plus esthétique** celui-ci.

Il sera **catégorisé** par chaque fichier .h pour vous permettre de vous retrouver **facilement** malgré le nombres différents de fichiers.

Bonne lecture !

Contents

Chapter 1

distri.c/.h

```
/*
* NMAX : Nombre max pour l'affichage de la table ascii;
* NMAX : 127 = Affichage basique de la table ascii;
* NMAX : 255 = Affichage de la table basique et étendue;
*/
#define NMAX 255

* Structure : tab_dist
*
* Attribut :
*    @attribut caractere : un caractère;
*    @attribut nboccurence : entier, nombre d'occurence d'une lettre;
*    @attribut probabilite : flottant, taux d'apparition d'une lettre;
*
* Description : Lit un fichier donné dans la sortie standard;
*/
typedef struct tab_dist tab_dist;
struct tab_dist {
    int caractere;
    int nboccurence;
    double probabilite;
};
```

```
/*
* Fonction : readFile
*
* Paramètre :
*    @param argv[ ] : tableau de chaine de caractères
*
* Description : Lit le premier nom donné dans la sortie standard
* (./huf [ici] [...]) et renvoi sa taille;
*/
size_t readFile (char* argv[ ]);

/*
* Fonction : initTab
*
* Paramètre :
*    @param file : le fichier source;
*    @param tailleFichier : la taille du fichier;
*
* Description : Initialise un tableau de probabilité;
*/
tab_dist* initTab ( FILE* file, const int tailleFichier);

/*
* Fonction : getSizeTab
*
* Paramètre :
*    @param tab : tableau de probabilités;
*
* Description : Renvoi la taille du tableau;
*/
int getSizeTab (const tab_dist* tab);


* Fonction : getMinProb
*
* Paramètre :
*    @param tab : tableau de probabilités;
*
* Description : Renvoi la probabilitée minimum du tableau;
*/
float getMinProb (tab_dist* tab);
```

```
/*
* Fonction : triTab
*
* Paramètre :
*    @param tab : tableau de probabilités;
*
* Description : Renvoi le tableau donné en paramètre, trié dans l'ordre
* croissant par rapport à la probabilitée;
*/
tab_dist* triTab (tab_dist* tab);


/*
* Fonction : printTab
*
* Paramètre :
*    @param tab : tableau de probabilités;
*
* Description : Affiche sur la sortie standard le contenu du tableau;
*/
void printTab (const tab_dist* tab);
```

Chapter 2

arbre.c/.h

```
/*
* Structure : noeud
*
* Attribut :
*    @attribut pere : entier, pere du fils gauche et droit;
*    @attribut fils_gauche : entier, fils gauche;
*    @attribut fils_droit : entier, fils droit;
*    @attribut probabilite : flottant, probabilité du noeud;
*
* Description : Structure définissant un noeud reliant un père à ses
* deux fils;
*/
typedef struct noeud noeud;
struct noeud {
    int pere;
    int fils_gauche;
    int fils_droit;
    float probabilite;
};
```

```
/*
* Structure : dictionnaire
*
* Attribut :
*    @attribut caractere : entier, le caractère associé à l'arbre;
*    @attribut taille : entier, taille du code du caractère;
*    @attribut code : chaine de caracteres, code du caractère;
*
* Description : Structure définissant un dictionnaire composé
* d'un caractère et de son code binaire;
*/
typedef struct dictionnaire dictionnaire;
struct dictionnaire {
    int caractere;
    int taille;
    char* code;
};


/*
* Fonction : creeArbre
*
* Paramètre :
*    @param tab : tableau de probabilités;
*
* Description : Crée l'arbre d'Huffman;
*/
noeud* creeArbre (const tab_dist* tab);
```

```
/*
* Fonction : parcoursArbre
*
* Paramètre :
*    @param arbre : structure noeud, arbre;
*    @param dico : structure dictionnaire, contient le caractère et le code
*    de chaque caractères;
*    @param positionDico : entier, position du dictionnaire;
*    @param code : chaine de caracteres, renvoi le code au fur est à mesure
*    des différents noeud;
*    @param positionCode : entier, position du code;
*    @param index : entier, numéro du noeud pointé;
*    @param tailleTab : entier, taille du tableau de probabilité;
*
* Description : Crée le code de chaque caractère de l'arbre;
*/
dictionnaire*  parcoursArbre (noeud* arbre, dictionnaire* dico, int positionDico, char* code, int positionCode, int index, int tailleTab);

/*
* Fonction : getSizeArbre
*
* Paramètre :
*    @param arbre : structure noeud, arbre d'Huffman;
*
* Description : Renvoi la taille de l'arbre;
*/
int getSizeArbre (const noeud* arbre);

/*
* Fonction : getSizeDico
*
* Paramètre :
*    @param dico : structure dictionnaire, dictionnaire de l'arbre;
*
* Description : Renvoi la taille du dictionnaire;
*/
int getSizeDico (const dictionnaire* dico);
```

```
/*
* Fonction : copyPreviousCode
*
* Paramètre :
*    @param dico : structure dictionnaire, dictionnaire de l'arbre;
*    @param positionDico : entier, position du dictionnaire;
*    @param code : chaine de caractères, le code à copier;
*    @param positionCode : entier, position du code;
*
* Description : Copie le code donné en paramètre dans la position du
* dictionnaire donnée;
*/
void copyPreviousCode (dictionnaire* dico, int positionDico, char* code, int
positionCode);

/*
* Fonction : printArbre
*
* Paramètre :
*    @param arbre : struture noeud, arbre d'Huffman;
*
* Description : Affiche sur la sortie standard l'arbre d'Huffman;
*/
void printArbre (const noeud* arbre);

/*
* Fonction : printDico
*
* Paramètre :
*    @param dico : structure dictionnaire, dictionnaire de l'arbre;
*
* Description : Affiche sur la sortie standard le code binaire de chaque
* caractère de l'arbre d'Huffman;
*/
void printDico (const dictionnaire* dico);
```

Chapter 3

compress.c/.h

```
/*
* Fonction : puissance
*
* Paramètre :
*    @param x : entier, le nombre;
*    @param n : entier, la puissance;
*
* @description : Renvoi x à la puissance n;
*/
int puissance (int x, int n);

/*
* Fonction : compress
*
* Paramètre :
*    @param argv[ ] : tableau de chaines de caractères, on récupère le
*    nom du fichier de sortie;
*    @param dico : structure dictionnaire, dictionnaire de l'arbre;
*    @param tailleFichierSource : entier, taille du fichier source;
*
* Description : Compresse le fichier donné en paramètre;
*/
void compress (char* argv[ ], dictionnaire* dico, size_t tailleFichierSource);
```

Chapter 4

decompress.c/.h

```
/*
* Fonction : decompress
*
* Paramètre :
*    @param argv[ ] : tableau de chaines de caracteres, on récupère le
*    nom du fichier de sortie;
*
* Description : Décompresse un fichier compressé;
*/
void decompress (char* argv[ ]);
```

Chapter 5

main.c

```
/*
* Fonction : estDansLesOptions
*
* Paramètre :
*    @param nb_options : entier, nombre d'options qui ont été rentré dans
*    la sortie standard;
*    @param options : chaine de caractères, options dont la présence est
*    à vérifier;
*    @param argv[ ] : tableau de chaines de caracteres;
*
* Description : Permet de vérifié si une option donnée a été donné
* dans la sortie standard;
*/
int estDansLesOptions (int nb_options, char* options, char* argv[ ])
```