Jean Perrin
Faculté des Sciences
Université d'Artois

C avancé
TD n° 6

UNIVERSITÉ D'ARTOIS
Licence 3 – semestre 1
2022 – 2023

Structures

Le but de cet exercice est d'implanter un algorithme de compression/décompression de fichier basé sur le codage de Huffman. Le codage de Huffman est un algorithme de compression de données sans perte. Il utilise un code à longueur variable pour représenter un symbole de la source. Le code est déterminé à partir d'une estimation des probabilités d'apparition des symboles de source, un code court étant associé aux symboles de source les plus fréquents. Pour cela est il est nécessaire dans un premier temps de construire une table de fréquence. Cette table consiste en un comptage empirique des fragments au sein des données à compresser. Prenons l'exemple d'un texte : 'Le codage de Huffman est un super codage'. On obtient la table suivante :

Lettres	Occurences	Fréquence
L	1	2.5%
H	1	2.5%
e	6	15%
c	2	5%
o	2	5%
d	3	7.5%
a	3	7.5%
g	2	5%
u	3	7.5%
f	2	5%
m	1	2.5%
n	2	5%
s	2	5%
t	1	2.5%
p	1	2.5%
r	1	2.5%
[Espace]	7	17.5%
Total	40	100%

Nous avons supposé que le fichier ne comporte que des caractères contenus dans la table ASCII, cependant en pratique nous pouvons considérer n'importe quel entier de type caractère puisqu'ils sont vu comme des suites octets.

Une fois la table de fréquence calculée, la seconde étape consiste à construire une structure d'arbre dont le chemin de la racine à une feuille permet de donner le code de la valeur considérée. L'arbre est créé de la manière suivante, on associe chaque fois les deux nœuds de plus faibles poids, pour donner un nouveau nœud dont le poids équivaut à la somme des poids de ses fils. On réitère ce processus jusqu'à n'en avoir plus qu'un seul : la racine. On associe ensuite par exemple le code 1 à chaque embranchement partant vers la gauche et le code 0 vers la droite.

Pour l'ensemble du code que vous aurez à écrire il est important que :

- Vous n'oubliez pas de vérifier les retours des fonctions que vous utilisez ;
- Vous testiez vos fonctions sur des petits exemples ;
- Vous utilisiez des assertions (`assert.h`) afin de vérifier le comportement de votre programme ;
- Vous utilisiez `valgrind` afin de vous assurez que vous n'avez pas de fuites mémoires ;
- Vous utilisiez des petits fichiers textes afin de valider votre algorithme de compression.

Travail à réaliser :

1. Dans un premier temps vous allez coder des fonctions permettant de lire un fichier texte caractère par caractère de telle sorte que vous puissiez aussi lire bit par bit. Le code suivant donne les en-têtes des fonctions que j'ai utilisé (vous n'êtes pas limité à cela) :

```c
#include <stdbool.h>

typedef struct _dataBufferReader {
  /*
        A définir
  */
} dataBufferReader;

// initialise une structure de donnée de type dataBufferReader et la retourne.
dataBufferReader *createBufferReader(char *fileName);

// ferme le fichier et libére la mémoire alloué.
void destroyBufferReader(dataBufferReader *data);

// récupére le prochain caractére.
int getCurrentChar(dataBufferReader *data);

// consomme le prochain caractére.
void consumeChar(dataBufferReader *data);

// vérifie si nous n'avons pas atteint la fin du fichier.
bool eof(dataBufferReader *data);

// retourne au début du fichier.
bool moveBeginning(dataBufferReader *data);

// récupére le prochain bit (on suppose une lecture bit par bit).
bool getCurrentBit(dataBufferReader *data);

// consomme le prochaine bit (on suppose une lecture bit par bit).
bool consumeBit(dataBufferReader *data);
```

2. Écrire un module permettant d'écrire un fichier texte bit par bit. Idem, ici une partie de mon code au sujet de cette partie :

```c
#include <stdbool.h>
#include <sys/types.h>

#define BUFFER_CAPACITY (8 * 1024)

typedef struct _dataBufferWriter {
/*
  A définir
*/
} dataBufferWriter;

// crée une structure de type dataBufferWriter et l'initialise.
dataBufferWriter *createBufferWriter(char *fileName);

// ferme le fichier et libére la mémoire allouée.
void destroyBufferWriter(dataBufferWriter *data);

// vide le buffer pour l'écrire dans le fichier.
void flush(dataBufferWriter *data);
```

```
// ajoute un bit dans le buffer (s'il est plein il faut le vider!).
void putBit(dataBufferWriter *data, bool val);

// ajoute un unsigned char dans le buffer (s'il est plein il faut le vider!).
void putUnsignedChar(dataBufferWriter *data, u_int8_t val);

// ajoute un unsigned int dans le buffer (s'il est plein il faut le vider!).
void putUnsignedInt(dataBufferWriter *data, unsigned val);
```

3. Maintenant nous allons nous intéresser à la représentation et manipulation de l'arbre de Huffman. Le code suivant donne les fonctionnalités que l'on souhaite avoir :

```
#include <sys/types.h>

typedef struct _node {
/*
  A définir
*/
} node;

// afficher l'arbre sur la sortie standard.
void display(node *n);

// récupére pour chaque caractére son code.
void codeConstruction(node *tree, long unsigned *correspondenceByte,
                      unsigned *correspondenceSize,
                      unsigned sizeCorrespondence, unsigned long word,
                      unsigned depth);

// libére la mémoire utilisée par notre arbre.
void destroyTree(node *tree);

// récupére les feuilles de l'arbre dans un certain ordre (à vous de choisir).
int getLeaveInOrder(node *tree, u_int8_t *orderedLeaf);

// afficher la structure qui sera dans le fichier compressé.
void displayStructure(node *tree, dataBufferWriter *bufferWriter);
```

4. Dans votre `main` vous aurez uniquement le code suivant :

```
#include <assert.h>
#include <stdio.h>
#include <string.h>

#include "huffman.h"

int main(int argc, char **argv) {
  if (argc != 4 || (strcmp(argv[1], "-c") && strcmp(argv[1], "-u"))) {
    printf("Pour␣compresser␣:␣\n");
    printf("[USAGE]␣%s␣-c␣INPUT_FILE␣OUTPUT_FILE\n", *argv);
    printf("\nPour␣décompresser␣:␣\n");
    printf("[USAGE]␣%s␣-u␣INPUT_FILE␣OUTPUT_FILE\n", *argv);
    exit(EXIT_FAILURE);
  }

  if (!strcmp(argv[1], "-c")) {
    printf("Compression␣de␣%s␣vers␣%s\n", argv[2], argv[3]);
    compress(argv[2], argv[3]);
  } else if (!strcmp(argv[1], "-u")) {
```

```
    printf("Décompression de %s vers %s\n", argv[2], argv[3]);
    uncompress(argv[2], argv[3]);
  } else {
    fprintf(stderr, "Option inconnue\n");
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```