**Compression et décompression à l’aide du codage de Huffman**

Nicolas CATONI

Alice DEVILDER

Groupe MN03

- Gdoc raffinage à mettre dans livrable sous forme pdf

- Pour le 15 janv dans livrable :

• Raffinage (gdoc)

• Source (codes)

• Manuel d’utilisation (manuel.pdf) : décrit comment utiliser les programmes développés. Il est illustré avec des copies d’écran.

• Rapport (rapport.pdf)

• Démo (demo.pdf) : 4 minutes décrit le déroulé de la démonstration et explique ce que vous allez montrer.

• Presentation orale (presentation.pdf) : 4 minutes nous expliquer les principaux choix fait dans votre projet : architecture en modules, structures de données, principaux algorithmes, avancement du projet...

*Le rapport doit au moins contenir les informations suivantes :*

*— un résumé qui décrit l’objectif et le contenu du rapport (10 lignes maxi),*

*— une introduction qui présente le problème traité et le plan du document,*

*— l’architecture de l’application en modules,*

*— la présentation des principaux choix réalisés,*

*— la présentation des principaux algorithmes et types de données,*

*— la démarche adoptée pour tester le programme,*

*— les difficultés rencontrées et les solutions adoptées en justifiant vos choix (en particulier*

*quand vous avez envisagé plusieurs solutions),*

*— l’organisation de l’équipe (qui a fait quoi, etc.),*

*— un bilan technique donnant un état d’avancement du projet et les perspectives d’amélioration/ évolution,*

*— une bilan personnel et individuel : intérêt, temps passé, temps passé à la conception,*

*temps passé à l’implantation, temps passé à la mise au point, temps passé sur le rapport, enseignements tirés de ce projet, etc.*

**Objectif**

Le codage de Huffman (1952) repose sur le codage à taille variable des caractères dans un texte en tenant compte de leur fréquence d’apparition. Il permet ainsi de définir un nouveau codage optimale dont les caractères de fréquences élevées sont codés sur moins de bits que ceux de fréquences faibles. C’est pourquoi ce codage statistique est utilisé pour la compression de fichier sans perte de données.

L’objectif de ce projet est d’écrire deux programmes, le premier qui compresse des fichiers en utilisant le codage de Huffman et le second qui les décompresse.

Dans ce rapport, nous présentons donc les méthodes de réalisation de ce projet, les décisions prises quant aux divers modules, sous-programmes, types de données utilisés et test réalisés ainsi que les difficultés que l’on a pu rencontrer et ce dont on en a appris.

**Cahier des charges**

- Obtenir un tableau de fréquences de chaque caractère présent dans le fichier texte à compresser.

- Construire un arbre de Huffman sous forme d’un arbre binaire dont la valeur de la fréquence de la branche droite est plus grande que celle de la branche gauche.

- Obtenir un tableau contenant le code de Huffman pour chaque caractère présent dans le fichier texte à compresser.

- Déterminer quand la décompression est terminée à l’aide d’un caractère de fin de fichier « /$ ».

- Construire une interface utilisateur et afficher les informations liées à la compression/décompression si l’option « -b » ou --bavard » est appelée.

**Introduction**

Le code de Huffman utilise la fréquence d’apparition (ou nombre d’occurence) des caractères dans le fichier donc afin de réaliser l’algorithme de Huffman, il faut tout d’abord construire un tableau contenant les fréquences de chaque caractère.

*Exemple. Prenons un fichier texte qui contient le texte suivant :*

*“Ceci est un exemple du codage de huffman”*

*Dans ce texte, le caractère ‘e’ apparaît 7 fois, c, u, d : 3 ; n, m, a, f : 2 ; i, s, t, x, p, l, g, h : 1,*

Ensuite, l’algorithme de Huffman consiste à créer un arbre binaire parfait ayant pour feuilles les caractères du texte et pour valeur leur fréquence d’apparition. Les noeuds, ayant toujours deux fils, permettent d’organiser les feuilles de manière à ce que la feuille dont le caractère a une fréquence élevée soit proche de la racine. À chaque noeud est associé une valeur qui est la somme de la valeur de son fils gauche et de la valeur de son fils droit. Par ailleurs, le fils gauche est celui dont la fréquence est la plus faible. Ainsi, la valeur de la racine correspond au nombre de caractères du texte. De plus, chaque branche de l’arbre est étiquetée par la valeur 0 pour le sous-arbre gauche et la valeur 1 pour le sous-arbre droit.

Le [code binaire](https://www.dcode.fr/code-binaire) de chaque caractère est alors obtenu en parcourant l'arbre de la racine jusqu'à la feuille (suivant un parcours infixe) et en notant le parcours (0 ou 1) à chaque [noeud](https://www.dcode.fr/notation-noeuds).

*Exemple*. Illustration de l’arbre

Une fois avoir construit l’arbre de Huffman, la décompression résulte à parcourir l’arbre jusqu’à obtenir un code correspond à un caractère et à répéter cette opération jusque’à la fin du texte encodé.

**Plan**

1. Conception de l’application

1.1 Architecture de l’application en modules

1.2 Principaux choix réalisés pour les algorithmes

1.3 Types de données

2. Réalisation et codage

2.1 Principaux algorithmes

2.2 Tests

2.3 Difficultés et solutions apportées (état d’avancement)

3. Organisation de l’équipe

4. Bilan

4.1 Bilan technique

4.2 Bilan personnel

5. Conclusion

**1. Conception et codage**

**1.1 Architecture de l’application en modules**

Afin de pouvoir utiliser des éléments de programme dans d’autres programmes, nous avons opté pour une architecture de l’application à l’aide de module.

Un module regroupe des abstractions de données et des abstractions de contrôle sous forme d’unité d’encapsulation. Ainsi l’application (programme compresser et decompresser) sera vu comme un ensemble de modules qui utilisent les uns et les autres.

A premier abord, nous avions crée un module *types* contenant les types que nous voulions utiliser dans les différents modules. Cependant ce module s’est avéré non essentiel car nous avons finalement défini les types associés à leur utilisation.

Nous avons donc implanté un module *memorie* dans lequel sont définis les types T\_octet, T\_bit et T\_bits (tableau de taille 8 de T\_bit). Cf 1.3

Ensuite, afin de lire les fichiers mis en argument de compression et réaliser différentes actions sur les fichiers, nous avons réalisé un module *fichier* dans lequel se trouve les sous-programmes Creer, Taille, Lire\_bit,Lire\_byte, Ecrire et Ajouter ainsi que les types *T\_tab\_bit* et *T\_tab\_octet* qui sont un tableau de taille **size\*8** de T\_bit et un tableau de taille **size** de T\_octet.