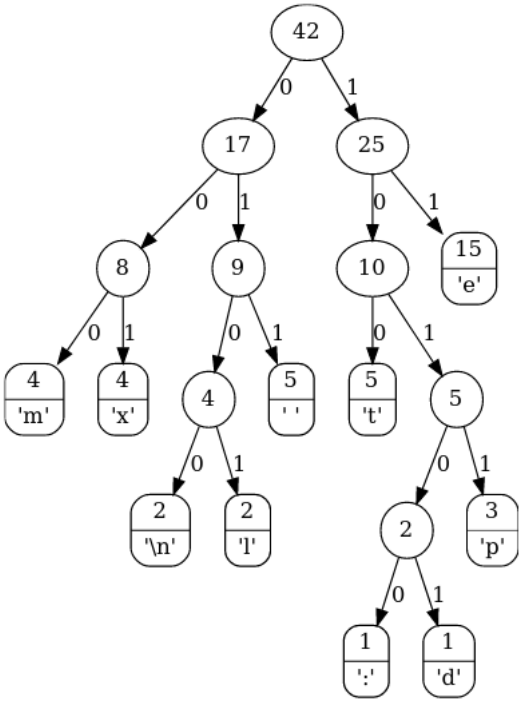
Projet PIM

Codage de Huffman

Réalisation de programmes de compression et de décompression de texte suivant la principe du codage de Huffman

Lisa Weisbecker Rémi Hallopeau



17 Janvier 2022

Table des matières

1 Introduction **3**
1.1 Contenu du rapport 3
1.2 Résumé du sujet 3

2 Choix réalisés **3**
2.1 Architecture des programmes 3
2.2 Type de donnés 4
2.3 Principales fonctions et procédures 5
2.4 Démarche adoptée pour tester le programme 6

3 Difficultés rencontrées et solutions adoptées **6**

4 Organisation au sein de l'équipe **7**

5 Bilan technique **8**

6 Bilans personnels **8**
6.1 Lisa 8
6.2 Rémi 8

1 Introduction

1.1 Contenu du rapport

Ce rapport porte sur un projet de codage de programmes de compression et décompression de texte utilisant le principe du codage Huffman.
Après une introduction,vous trouverez l'architecture de l'application en modules, puis nous vous présenterons nos choix principaux.
Pour continuer, nous avons défini plusieurs types de données et écrit différents algorithmes. Après cela, vous trouverez les démarches adoptées pour tester nos programmes et les difficultés rencontrées. Par la suite, vous aurez un aperçu de l'organisation au sein de notre équipe. Pour conclure ce rapport vous trouverez des bilans technique et personnels du travail réalisé.

1.2 Résumé du sujet

Le projet à réaliser portait sur le principe du codage de Huffman. En utilisant ce principe, nous devions écrire un premier programme permettant de compresser un fichier texte, et un second programme permettant de décompresser le fichier obtenu. La réalisation de ces programmes nécessitait la compréhension du principe du codage de Huffman, ainsi que la réalisation d'un raffinage complet pour ces deux algorithmes.

2 Choix réalisés

2.1 Architecture des programmes

Les deux programmes principaux, compression et décompression utilisent les fonctions du module "Huffman". Ils prennent en argument le nom du fichier que l'utilisateur souhaite compresser ou décompresser et demandent également à l'utilisateur s'il souhaite afficher l'arbre et la table de Huffman ou non.
Nous d'abord, nous avions réalisés séparément les modules gérant les listes et les arbres, avant de finalement mettre l'ensemble de ses fonctions dans le même module pour des raisons de simplicité.
Les deux programmes créent un fichier dans lequel se trouve une chaîne de caractères.

2.2 Type de donnés

Nous avons défini de nombreux types de données pour répondre aux besoins du sujet.
Premièrement, nous avons évidemment défini un type d'arbre (T_Arbre) pour réaliser l'arbre de Huffman, ainsi que les fonctions et procédures qui permettent de manipuler les données d'un arbre. Il s'agit d'un type d'abre un peu différent de celui décrit en TD car nous n'avons pas utilisé de clés.

```
Type T_Cellule;

Type T_Arbre is access T_Cellule;

Type T_Cellule is
    record
        Occurence: Integer;
        Caractere: T_Octet;
        Sag: T_Arbre;
        Sad: T_Arbre;
    end record;
```

FIGURE 1 – Description du type T_Arbre

Il a fallu ensuite trouver une méthode pour stocker des arbres. Après avoir considérer utiliser des listes d'arbres de type LCA, nous avons décidé de réaliser des tableaux d'arbres (type T_Liste), qu'on utilisera comme des listes.

```
type T_Liste is array (1..257) of T_Arbre;
```

FIGURE 2 – Description du type T_Liste

Enfin, pour la table de Huffman, nous avons défini un autre type de tableau (T_Table) mais dont les cases ne sont cette fois pas des arbres mais des cellules contenant un caractère et son code de Huffman.

```
Type T_Cellule_case;

Type T_Case is access T_Cellule_case;

Type T_Cellule_Case is record
    Caractere: T_Octet;
    Code: Unbounded_String;
end record;

Type T_Table is array (1..257) of T_Case;
```

FIGURE 3 – Description du type T_Table

Les caractères manipulés étant des Octet, nous avons également défini le type T_Octet.

2.3 Principales fonctions et procédures

Pour compresser un fichier, il faut commencer par construire l'arbre de Huffman.
La procédure Get_Frequence permet de récupérer les occurrences des caractères dans un fichier donné et de les stocker dans une liste d'arbres qu'on initialise. Chaque arbre de cette liste contient l'occurrence d'un caractère, le caractère (en Octet), ainsi qu'un sous arbre gauche et un sous arbre droit null.
Cette liste est de taille 257 afin de pouvoir éventuellement stocker tout les caractères possibles ainsi que le caractère de fin de liste \$ (de valeur set à -1 arbitrairement).
L'opération construire_ arbre fusionne les arbres de la liste selon les étapes décrites dans le sujet jusqu'à ce qu'il n'en reste qu'un, il s'agit alors de l'arbre de Huffman.
Ensuite, les procédures construire_Table et rearranger_Table vont permettre, à partir de cet arbre, d'enregistrer le codage de Huffman correspondant à chaque caractère. Pour trouver ce code, comme expliqué dans le sujet,, on descend le long de l'arbre de Huffman jusqu'à arriver à une feuille. Le code du caractère qui se trouve sur cette feuille correspond au chemin réalisé : quand on tourne à droite, on rajoute un 1 au code, et quand on tourne à gauche, on rajoute un 0.
Rearranger_Table permet de placer le caractère de fin de liste (qui n'a pas

de code selon ASCII) en début de liste, et à lui attribuer sa position dans l'arbre en tant que code.
On peut désormais réaliser la "signature" du texte compressé qui permettra, lors de la décompression, de retrouver l'arbre de Huffman. Coder_Arbre permet de récupérer le codage de l'arbre selon son parcours infixe et code_caractères donne les caractères et l'ordre dans lequel ils sont placés dans l'arbre. Le dernier caractère est doublé, ce qui permet de repérer la fin de la liste des caractères.
Enfin compresser_texte va coder chaque caractère du texte donné par l'utilisateur selon le codage de Huffman, grâce à la table.
Les procédures Dessiner_Arbre et Dessiner_Table affichent l'arbre et la table de Huffman si l'utilisateur le souhaite.
Pour décompresser, recreer_arbre permet de reconstruire l'arbre de Huffman à partir de la signature du texte.
Decoder_texte va, à partir de la suite de 0 et de 1 du texte compressé, descendre le long de l'arbre en tournant à droite lorsqu'on rencontre un 1, et à gauche lorsqu'on rencontre un 0. Quand on tombe sur une feuille, on a retrouvé un caractère original du texte, il est alors écrit dans le texte décompressé.
Nous avons également réaliser plusieurs sous programmes de conversion permettant de manipuler les données utilisées. Ils nous ont permit de passer d'octet à leur représentation sur 8 bits et inversement.

2.4 Démarche adoptée pour tester le programme

Le long de la réalisation des sous programmes, nous avons utilisé un programme de test permettant de créer des arbres ou d'utiliser des textes afin de tester les différentes fonction et procédures du module. En fonctionnant ainsi par étape, cela à pu éviter de propager des erreurs le long du projet.
Une fois les différents sous programmes testés, nous avons réalisé un programme de test "propre" sur un fichier de test contenant des caractères de fréquence d'apparition différentes et qui simule la compression d'un fichier. Le fichier compressé n'est pas créé mais est affiché dans la console.

3 Difficultés rencontrées et solutions adoptées

Au début du projet, nous avons eu beaucoup de mal à comprendre ce que nous avions en entrée de nos deux programmes et ce que nous devions

renvoyer. Nos premiers raffinages et les retours du professeur nous ont permis de cerné le sujet et de se lancer dans la programmation. De plus, nos programmes de test nous ont permis de repérer les erreurs de programmation et de les modifier en conséquence.

Le sous-programme ayant posé le plus de difficultés est Coder_Arbre. En effet après avoir essayé différentes méthodes, sans succès, et l'aide du professeur, nous avons eu l'idée de compter les virages à gauche effectuer dans l'arbre et de fonctionner par récursivité. Le défaut de cette méthode et qu'il manque un 1 à la fin de la signature de l'arbre, qu'il faut donc rajouter hors du programme. Le sous-programme Construire_ able nous a également posé le même genre de problèmes, mais une astuce du professeur nous a permis d'obtenir un algorithme efficace.

Enfin, nous n'avons malheureusement pas eu le temps de finir le projet. Le programme de décompression a été difficile à comprendre pour nous, nous avons donc commencer l'écriture des sous-programmes à partir de notre raffinage, mais sans pouvoir finir. Cela a également été dû à une mauvaise répartition des tâches au sein de l'équipe. A cause d'une mauvaise compréhension des sorties du programme compression, l'avancée du programme de décompression a été difficile.

4 Organisation au sein de l'équipe

Nous avons commencé la réflexion ensembles pour déterminer les types de donné que nous allions créer et construire le raffinage.
Pour le premier rendu, Rémi a réalisé le module de Liste d'arbre et Lisa a réalisé le module Arbre.
Par la suite, Lisa a réalisé les fonctions et procédures servant à la décompression, les programmes de tests ainsi que le programme principal de compresser, le module d'utilisateur et une partie du rapport.
Rémi a réalisé les sous programmes de décompression qui n'ont malheureusement pas pu être finalisés et une partie du rapport.

5 Bilan technique

Comme énoncé dans la partie précédente, nous n'avons pas eu le temps de finir le projet. Mais nous avons tout de même réussi à finir le programme de Compression. En ayant un peu plus travailler au début du projet, nous aurions pu finir le programme de Décompression.
De plus, tous nos programmes ne sont pas optimisés au maximum. Il existe pour certains des programmes des méthodes plus efficaces. Il serait judicieux, avec un peu plus de temps, d'utiliser Valgrind pour repérer les fuites de mémoire et minimiser les pertes de mémoires occasionnées par le programme.

6 Bilans personnels

6.1 Lisa

Je suis relativement satisfaite du travail que j'ai réalisé, bien que je regrette que l'organisation de notre équipe n'ai pas permis de terminer le sujet. La partie compression que j'ai réalisé fonctionne bien, mais j'aurais aimé avoir un peu plus de temps pour rendre certains algorithmes plus efficaces, améliorer leur présentation et spécification, et régler les problèmes de fuite de mémoire.
Le sujet était intéressant mais j'ai trouvé le codage en ADA très fastidieux. Cependant, cela m'a permis d'acquérir un peu plus de rigueur en programmation, particulièrement lorsqu'il s'agit de manipuler différents types de donnée.

6.2 Rémi

Je n'ai pas particulièrement accroché le sujet, même si le principe du codage de Huffman est intéressant. Au total, j'ai dû passer 25 heures sur ce projet, 4h sur la conception et sur les raffinages, 14h sur l'implantation, 1h sur la mise au point et 6h sur le rapport.
Ce projet m'a permis de réaliser l'importance de l'organisation et de la répartition du travail au sein d'un groupe, et de bien préparer l'architecture du programme avant l'implantation. De plus, le bonne définition de chaque type de données en amont de l'écriture des programmes facilite grandement celle-ci.