Codage de Huffman

Marin Franot & Testu François

Équipe GH10

Sommaire

I Résumé

L'objectif de ce projet est de compresser et décompresser un fichier texte à l'aide de la méthode de Huffman. Cette méthode de compression consiste en l'utilisation de code moins coûteux pour les caractères utilisés le plus fréquemment. Ainsi, il y aura donc moins de bits utilisés pour le codage de tout un fichier texte. La décompression consiste en la récupération du fichier initial. C'est l'opération réciproque.

II Problème

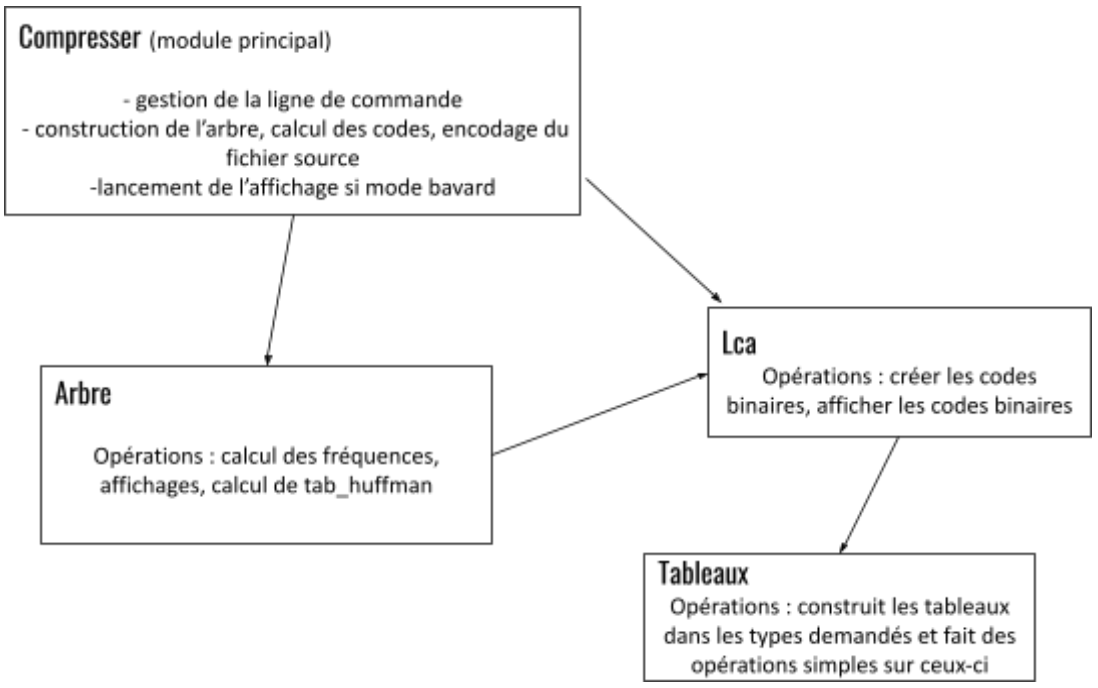
❖ Introduction

Pour mettre en place la compression et la décompression d'un fichier texte, nous avons créé différents modules afin de définir des types et fonctions qui nous seront ensuite utiles dans le programme principal. L'architecture de ces modules sera ainsi présentée.

La création de ces modules et des fonctions à l'intérieur résultent de choix sur la résolution du problème. Ces choix ont permis de réussir à implanter la compression et la décompression d'un fichier. Les principaux choix seront alors explicités.

Enfin, les principaux algorithmes et types de données seront présentés afin de mettre en évidence la démarche suivie.

❖ Architecture de l'application en modules



❖ Présentation des principaux choix réalisés

Les principaux choix que nous avons effectué sont les suivant :

➔ Le choix de la généricité des modules : Nous avons choisi de n'avoir que le module tableau en générique. Ceci nous a permis de simplifier notre création et notre utilisation des modules lca et arbre. Ces modules n'avaient d'ailleurs pas d'intérêt particulier à être en générique pour ce projet précis. Le module tableau, en générique, était utile puisqu'on utilise des tableaux avec différents types de données.

➔ L'utilisation de tableaux avec un rangement astucieux : Nous avons décidé d'organiser le rangement des tableaux de manière que l'entier correspondant au code ASCII du caractère soit l'indice des tableaux où ses informations sont stockées. Ceci permet de manipuler facilement les informations du tableau et de ne pas avoir à stocker le caractère ou son code ASCII. On note que le nombre de caractères différents est connu et assez faible ce qui justifie l'utilisation de cette méthode.

➔ Utiliser des octets dans les lca : Dans la démarche de résolution, l'utilisation de binaires dans les lca permet la manipulation plutôt complexe des binaires de manière assez simple et intuitive.

➔ Modélisation des caractères à partir des entiers : Ce choix est apparu lors du codage. Il s'est avéré plus facile de modéliser les caractères par les entiers associés dans leur code ASCII pour gérer la présence du symbole "/$".

➔ Supprimer le paramètre clé dans les lca, car il n'est pas utile dans le projet, les lca sont donc des simples listes.

❖ Présentation des principaux algorithmes et types de données

Les algorithmes principaux sont les suivants :

➔ Pour le module arbre

- creer permet de créer une nouvelle feuille avec la clé cle et la donnée donnee dans arbre.

```
30      procedure creer (arbre : in out T_Arbre; cle : in Integer; donnee : in Integer) is
31      begin
32          if est_vide(arbre) then
33              arbre := new T_Noeud;
34          end if;
35          arbre.all.cle := cle;
36          arbre.all.donnee := donnee;
37          arbre.all.fils_g := null;
38          arbre.all.fils_d := null;
39      end creer;
```

- La procédure Fusionner fusionne les arguments arbre_g et arbre_d en créant arbre qui a pour fils gauche arbre_g et pour fils droit arbre_d. La clé de cet arbre est la somme des clés des arbres fusionnés.

```
67      procedure Fusionner (arbre : out T_Arbre; arbre_g : in T_arbre; arbre_d : in T_arbre) is
68      begin
69          arbre := new T_Noeud;
70          arbre.all.cle := arbre_d.all.cle + arbre_g.all.cle;
71          arbre.all.fils_g := arbre_g;
72          arbre.all.fils_d := arbre_d;
73      end Fusionner;
```

- creer_tab_huffman permet de créer un tableau de lca. Chaque lca correspond au chemin associé au caractère dans l'arbre. La position des lca dans l'arbre correspond aux codes ASCII des caractères.

```
146     procedure creer_tab_huffman(arbre : in T_arbre; tab_huffman : in out tab_lca.T_Tableau; code : in out T_LCA) is
147         indice : integer;
148         code0 : T_LCA;
149         code1 : T_LCA;
150     begin
151         if est_feuille(arbre) then
152             indice := la_donnee(arbre);
153             if indice = -1 then
154                 indice := 257;
155             end if;
156             Enregistrer(tab_huffman, indice, code);
157         else
158             copier(code, code0);
159             ajouter(code0, 0);
160             creer_tab_huffman(fils_gauche(arbre), tab_huffman, code0);
161             copier(code, code1);
162             ajouter(code1, 1);
163             creer_tab_huffman(fils_droit(arbre), tab_huffman, code1);
164             vider(code);
165         end if;
166     end creer_tab_huffman;
```

- reconstituer_arbre permet de reconstituer l'arbre grâce à sa structure en lca. C'est la fonction réciproque de creer_tab_huffman qui permettait de créer la lca grâce au tableau. Ici, on crée le tableau grâce à un parcours infixe de la lca structure_arbre.

```
168     procedure reconstituer_arbre(arbre : in out T_Arbre; structure_arbre : in out T_LCA; tab_infixe
169                                  : in tab_entier.T_Tableau; indice : in out Integer) is
170         donnee : Integer;
171     begin
172         if Est_Vide(structure_arbre) then
173             null;
174         elsif la_donnee(structure_arbre) = 1 then
175             indice := indice + 1;
176             donnee := tab_entier.la_donnee(tab_infixe, indice);
177             creer(arbre, 0, donnee);
178             supprimer_1er(structure_arbre);
179         else
180             creer(arbre,0,0);
181             supprimer_1er(structure_arbre);
182             reconstituer_arbre(arbre.all.fils_g, structure_arbre, tab_infixe, indice);
183             reconstituer_arbre(arbre.all.fils_d, structure_arbre, tab_infixe, indice);
184         end if;
185     end reconstituer_arbre;
186
187
188 end arbre;
```

➔ Pour le module lca

```
116     function int2byte(val : in integer) return T_octet is
117         octet : T_octet;
118         bit : integer;
119         courant : integer;
120     begin
121         octet := 0;
122         courant := val;
123         if val = -1 then
124             return -1;
125         end if;
126         for i in 1..8 loop
127             bit := courant/128;
128             if bit = 1 then
129                 octet := (octet * 2) or 1;
130             else
131                 octet := (octet * 2) or 0;
132             end if;
133             courant := courant * 2 mod 2**8;
134         end loop;
135         return octet;
136     end int2byte;
```

- int2byte permet de transformer un entier en octet. Pour ce faire, on utilise les opérations définies dans les ressources pour récupérer un à un les bits.

- encoder_sda permet d'écrire dans le fichier file_hff le nouveau code.

```
138  procedure encoder_sda(sda : in T_LCA; compteur : in out Integer; octet : in out T_octet; file_hff
139                        : in out Byte_file.file_type) is
140      bit : T_octet;
141  begin
142      if compteur = 8 then--encoder l'octet quand il est complet
143          compteur := 0;
144          Write(file_hff, octet);
145          octet := 0;
146      end if;
147      if sda /= null then--ajouter la valeur de la lca dans l'octet
148          bit := sda.all.donnee;
149          octet := (octet * 2) or bit;
150          compteur := compteur + 1;
151          encoder_sda(sda.all.suivant, compteur, octet, file_hff);
152      end if;
153  end encoder_sda;
```

III Organisation autour du projet

❖ Démarche adoptée pour tester le programme

➔ Pour tester les programmes des modules arbre et lca, on compilait d'abord pour voir s' il n'y avait pas d'erreur de syntaxe à la compilation.

➔ S'il n'y avait pas d'erreurs à la compilation, pour tester les programmes des modules arbre et lca, on a utilisé des fichiers tests qui vérifiaient si les actions des fonctions et procédures étaient correctes grâce à des "pragma assert".

➔ L'affichage était aussi une bonne méthode pour voir si les résultats intermédiaires étaient cohérents et ceux souhaités.

➔ Enfin, pour vérifier les programmes principaux on a utilisé valgrind pour vérifier que l'espace mémoire était bien géré et aussi pour créer un exécutable.

➔ A l'aide de l'exécutable, on a pu essayer de compresser un fichier test.txt en un nouveau fichier test.txt.hff. Puis, on a pu décompresser test.txt.hff en un nouveau fichier test.txt.hff.txt. On a pu vérifier que le contenu de ce dernier était le même que celui initial.

❖ Difficultés rencontrées/solutions adoptées

★ Difficultés rencontrées

➔ L'utilisation de svn à 2 nous a fait perdre du temps et fait faire des efforts inutiles. On a perdu plus d'une fois les avancées d'un des 2 binômes car on travaillait en même temps.

➔ La localisation des erreurs était aussi une grande source de difficultés. Parfois, le code semble correct et cohérent mais on oublie de l'interpréter comme si on était l'ordinateur.

➔ L'affichage était assez complexe et fastidieux. Pour correspondre parfaitement au modèle demandé il a fallu modifier les affichages à de multiples reprises.

➔ La manipulation des octets a aussi représenté une grande difficulté. Il a fallu essayer quelques méthodes avant de pouvoir compiler et réussir à faire ce que l'on voulait.

➔ La compréhension et la communication entre nous pouvaient aussi parfois être une difficulté puisque c'est difficile de comprendre la logique de programmation de quelqu'un d'autre. Et aussi, puisque le problème était assez grand et ouvert, on pouvait parfois avoir des désaccords sur les méthodes à employer voire sur la compréhension des méthodes proposées.

★ Les solutions adoptées

➔ A force de persévérance et de temps passé sur les programmes, les procédures ont peu à peu commencé à fonctionner et on a pu avancer pas à pas sur le projet.

➔ Les échanges avec le professeur encadrant Mr. Hamrouni nous ont parfois permis de nous débloquer ou de réorienter notre raisonnement vers une meilleure solution.

➔ La manipulation des octets a finalement pu être effectuée grâce à l'utilisation des lca.

❖ Organisation de l'équipe

Avant les vacances, nous avons réfléchi ensemble à notre stratégie, fait les raffinages et passé du temps à définir les bons types à utiliser. Ensuite, nous avons créé les modules secondaires utiles pour faire compresser et décompresser. Nous avons ensuite créé les fichiers tests associés à ces modules.

Pendant les vacances, Marin a beaucoup travaillé sur l'avancée du programme principal compresser (sans que j'en sois au courant). Pour ma part, j'ai voulu profiter de mes vacances pour me reposer et voir ma famille.

Après les vacances, j'ai donc essayé de rattraper le wagon à la rentrée et j'ai aidé Marin à finir le module compresser. Ensuite, Marin avait très vite compris la construction du module décompresser donc nous l'avons fait assez vite. Pour compenser son implication dans l'encodage, je me suis occupé du rapport, de modifier les raffinages, de faire la présentation et des 2 autres pdf à rendre.

IV Bilan

- ❖ Le bilan technique

Nos codes peuvent compresser et décompresser un fichier texte.
Compression du fichier test.txt.

```
ftestu@bulbizarre:~/1A/pim/tp/GH10/Gnat/src$ ./compresser test.txt
00100110110011001001111
' ' --> 101
'!' --> 110101
''' --> 11011
'c' --> 111
'e' --> 0100
'i' --> 0101
'm' --> 0110
'o' --> 00
's' --> 0111
't' --> 1100
'u' --> 100
'\$' --> 110100
(18)
 \--0--(7)
 |     \--0--(3) o
 |     \--1--(4)
 |           \--0--(2)
 |           |     \--0--(1) e
 |           |     \--1--(1) i
 |           \--1--(2)
 |                 \--0--(1) m
 |                 \--1--(1) s
 \--1--(11)
       \--0--(5)
       |     \--0--(2) u
       |     \--1--(3)
       \--1--(6)
             \--0--(3)
             |     \--0--(1) t
             |     \--1--(2)
             |           \--0--(1)
             |           |     \--0--(0) \$
             |           |     \--1--(1) !
             |           \--1--(1) '
             \--1--(3) c
```

test.txt.hff

Ce fichier a été généré.

Décompression du fichier test.txt.hff.

```
ftestu@bulbizarre:~/1A/pim/tp/GH10/Gnat/src$ ./decompresser test.txt.hff
(0)
 \--0--(0)
 |     \--0--(0) o
 |     \--1--(0)
 |           \--0--(0)
 |           |     \--0--(0) e
 |           |     \--1--(0) i
 |           \--1--(0)
 |                 \--0--(0) m
 |                 \--1--(0) s
 \--1--(0)
       \--0--(0)
       |     \--0--(0) u
       |     \--1--(0)
       \--1--(0)
             \--0--(0)
             |     \--0--(0) t
             |     \--1--(0)
             |           \--0--(0)
             |           |     \--0--(0) \$
             |           |     \--1--(0) !
             |           \--1--(0) '
             \--1--(0) c
```

test.txt.hff.txt

Ce fichier a été généré.

Les programmes ne sont pas complètement robustes, ils peuvent être améliorés en traitant la compression d'un fichier vide, ou bien en détectant tous les fichiers compressés qui sont intentionnellement faux et qui ne peuvent pas être décompressés.

❖ Les bilans personnels et individuels

★ Bilan personnel de François

Ce projet m'a fait prendre conscience de l'importance des raffinages et de la réflexion avant de coder. Sur des projets de grandes ampleurs, il est impératif de faire ce travail au préalable pour ne pas se perdre.

Au niveau de l'implantation à 2, la différence de niveau était difficile à gérer puisqu'il fallait réussir à suivre Marin. J'ai aussi pu noter l'importance des commentaires puisqu'il faut que l'autre puisse comprendre le code.

Ce projet a permis de présenter une démarche et une rigueur à suivre pour pouvoir implanter des programmes.

Au niveau du groupe, il faudra que l'on fasse davantage attention à la communication.

★ Bilan personnel de Marin

J'ai pris du plaisir à travailler sur le projet, mais j'ai également beaucoup appris, notamment sur l'importance du bon choix des structures de données, ainsi que sur l'utilisation des données génériques.

J'ai apprécié écrire et faire fonctionner mes fonctions et procédures une par une, tout en voyant la progression de mon travail.

Je regrette de ne pas avoir eu le temps d'améliorer la robustesse du programme.