

**RAPPORT PROJET SCALA**

**Méthodes de compression sans pertes**

Jules Hurbin

Emile Geroudet-Dalle

Yasmine Chenouf

Victor Vitcheff

Antoine Le Du

Classe GMI01

Sommaire

1.Introduction

2. Méthode RLE

3. Méthode statistiques (Huffman er Shannon-Fano)

4.Méthodes à dictionnaire (LZ78 et LZW)

**Introduction**

Après la pratique du langage de programmation SCALA durant les différents TD, nous devions réaliser un projet de fin de semestre dont le but était de compresser et décompresser des chaînes de caractères sans pertes. En effet, la compression permet de réduire la taille et le temps de transfert et la décompression réalisait alors le chemin inverse pour obtenir, à partir d’une version compressée, les données telles quelles étaient à leur état initial.

Plusieurs méthodes ont été étudiées dans ce projet, à savoir : la méthode RLE, les méthodes statistiques avec la méthode Huffman et celle de Shannon-Fano ainsi que les méthodes à dictionnaire, méthode LZ78 et LZW

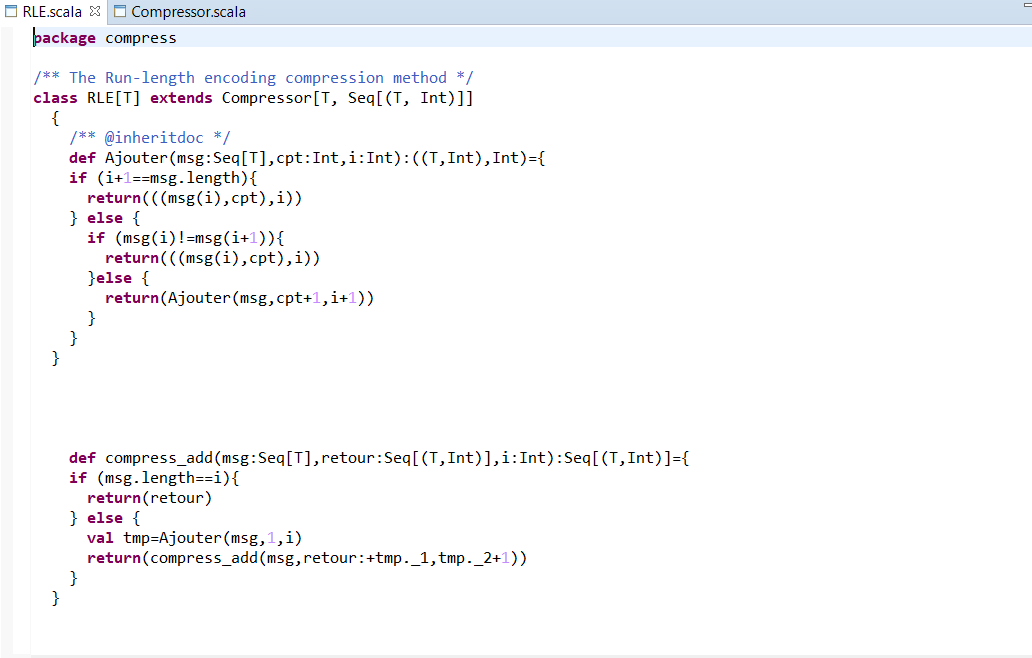
**Méthode RLE**

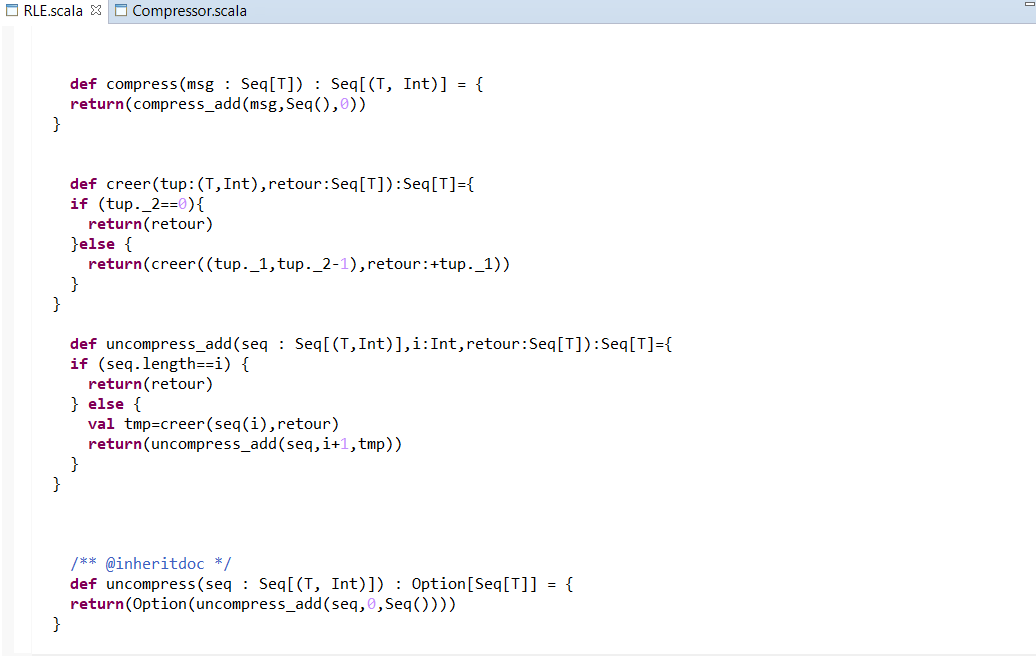
La méthode RLE consiste à compresser les symboles identiques qui se suivent.

La chaîne de caractères « aabbccc » donnera alors Seq ( (a,2),(b,2),(c,3) ).

Pour la décompression il faudra donc partir de Seq ( (a,2),(b,2),(c,3) ) pour obtenir « aabbccc » .

Voici l’implémentation de cette méthode





Cette classe fut l’une des plus faciles à implémenter. Elle n’a pas demandé beaucoup de réfléxion car le concept fut plutôt facile à comprendre. Les tests que nous avons utilisé sur cette fonction n’ont présenté aucun problème. Nous avons réalisé cette partie de manière efficace et rapide.

**Méthode Statistiques**

Les méthodes de Huffman et Shannon-Fano sont assez similaires puisque la compression avec celles-ci nous donne comme résultat une séquence de bits. Il y a donc un codage binaire de chaque symbole.

« a » va donner 00

« abba» va donner 00010100.

La compression s’est tout de même ici avéré plus difficile. En effet, les classes étaient imbriquées les unes dans les autres et comprendre le code ainsi que les arbres a été plus difficile. Nous avons essayé de voir tous les cas possibles pour les méthodes statistiques mais malheureuseument, nous avons fait face à des problèmes de compilation. Nous n’avons malheureusement pas eu le temps de coder la création des arbres ainsi que les méthodes à dictionnaire. Nous saurons cependant vous les expliquer.

**Méthode Huffman**

Voici en quoi consiste plus précisément la méthode Huffman en quelques étapes avec un exemple et la chaîne de caractère « abbca »    :

|  |  |  |  |
| --- | --- | --- | --- |
| *Symbole* | a | b | c |
| *Effectif* | 2 | 2 | 1 |

-Trier la distribution des symboles par ordre décroissant du  
nombre d'apparitions

- Fusionner les deux dernières colonnes et les réinsérer en  
conservant l'ordre décroissant

|  |  |  |
| --- | --- | --- |
| *Symbole* | bc | a |
| *Effectif* | 3 | 2 |

- Recommencer l'étape 2 jusqu'à obtenir une seule colonne

-Construire l'arbre en inversant les fusions successives

|  |  |
| --- | --- |
| *Symbole* | abc |
| *Effectif* | 5 |

**Méthode Shannon-Fano**

Voici en quoi consiste plus précisément la méthode Shannon-Fano en quelques étapes avec un exemple et la chaînes de caractère « abbca » :

|  |  |  |
| --- | --- | --- |
| *a* | b | c |
| 2 | 2 | 2 |

- Trier la distribution des symboles rangée par ordre décroissant du  
nombre d'apparitions

- Couper en deux sous-distributions *les plus équilibrées possibles*

|  |  |  |  |
| --- | --- | --- | --- |
| a |  | b | c |
| 2 |  | 2 | 1 |

- Recommencer l'étape 2 sur chaque sous-distribution jusqu'à obtenir  
des distributions à un symbole

- Construire l'arbre à partir des coupes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a |  | b |  | c |
| 2 |  | 2 |  | 1 |

**Méthodes à dictionnaire**

Voici en quoi consiste la compression selon les méthodes à dictionnaire :

-Encodage des séquences de caractères par des références à  
des emplacements dans un dictionnaire  
-Dictionnaire construit à partir du texte lui-même contenant  
toutes les séquences déjà rencontrées  
-Très utilisé dans les logiciels de compression/décompression

Cette méthode permet la construction du dictionnaire au fur et à mesure ce qui la distingue de la méthode statistique qui nécessite l’arbre d’encodage.

**Méthode LZ78**

Voici en quoi consiste plus précisément la méthode LZ78 en quelques étapes avec un exemple et la chaînes de caractère « belle echelle » :

- Initialement : chaîne vide en position 0  
- Recherche de la chaîne maximale existante  
- Ajout au dictionnaire du couple (entier, caractère) avec  
 .numéro de la chaîne maximale trouvée  
 .dernier caractère suivant cette chaîne

**Compression**

1. "b" en position 1 du dictionnaire, ajout de (0,'b')  
2. "e" en position 2 du dictionnaire, ajout de (0,'e')  
3. "l" en position 3 du dictionnaire, ajout de (0,'l')  
4. "le" en position 4 du dictionnaire, ajout de (3,'e')  
5. "␣" en position 5 du dictionnaire, ajout de (0,'␣')  
6. "ec" en position 6 du dictionnaire, ajout de (2,'c')  
7. "h" en position 7 du dictionnaire, ajout de (0,'h')  
8. "el" en position 8 du dictionnaire, ajout de (2,'l')  
9. "le␣" en position 9 du dictionnaire, ajout de (4,'␣')  
10. "!" en position 10 du dictionnaire, ajout de (0,'!')

→ Seq((0,'b'), (0,'e'), (0,'l'), (3,'e'), (0,'␣'), (2,'c'),(0,'h'), (2,'l'), (4,'␣'), (0,'!'))

**Décompression**

1. (0,'b') = "b" ajouté en position 1 dans le dictionnaire  
2. (0,'e') = "e" ajouté en position 2 dans le dictionnaire  
3. (0,'l') = "l" ajouté en position 3 dans le dictionnaire  
4. (3,'e') = "le" ajouté en position 4 dans le dictionnaire  
5. (0,'␣') = "␣" ajouté en position 5 dans le dictionnaire  
6. (2,'c') = "ec" ajouté en position 6 dans le dictionnaire  
7. (0,'h') = "h" ajouté en position 7 dans le dictionnaire  
8. (2,'l') = "el" ajouté en position 8 dans le dictionnaire  
9. (4,'␣') = "le␣" ajouté en position 9 dans le dictionnaire  
10. (0,'!') = "!" ajouté en position 10 dans le dictionnaire

→ "belle echelle !" (même dictionnaire que pendant la compression)

**Méthode LZW**

Voici en quoi consiste plus précisément la méthode LZW en quelques étapes avec un exemple et la chaînes de caractère « belle echelle » :

-Dictionnaire initial non vide  
 (Par défaut, table ASCII de 0 à 255)

-À l'apparition d'une nouvelle séquence :  
 .on ajoute la chaîne maximale au résultat ;  
 .on reprend la lecture depuis le nouveau caractère.

**Compression**

1. "be" en position 256 du dictionnaire, ajout de 98 ('b')  
2. "el" en position 257 du dictionnaire, ajout de 101 ('e')  
3. "ll" en position 258 du dictionnaire, ajout de 108 ('l')  
4. "le" en position 259 du dictionnaire, ajout de 108 ('l')  
5. "e␣" en position 260 du dictionnaire, ajout de 101 ('e')  
6. "␣e" en position 261 du dictionnaire, ajout de 32 ('␣')  
7. "ec" en position 262 du dictionnaire, ajout de 101 ('e')  
8. "ch" en position 263 du dictionnaire, ajout de 99 ('c')  
9. "he" en position 264 du dictionnaire, ajout de 104 ('h')  
10. "ell" en position 265 du dictionnaire, ajout de 257  
11. "le", ajout de 259

→ Seq(98, 101, 108, 108, 101, 32, 101, 99, 104, 257, 259)

**Décompression**

1. 115 ('b') : ajout de "b"  
2. 101 ('e') : ajout de "e", "be" en position 256 du dictionnaire  
3. 108 ('l') : ajout de "l", "el" en position 257 du dictionnaire  
4. 108 ('l') : ajout de "l", "ll" en position 258 du dictionnaire  
5. 101 ('e') : ajout de "e", "le" en position 259 du dictionnaire  
6. 32 ('␣') : ajout de "␣", "e␣" en position 260 du dictionnaire  
7. 101 ('e') : ajout de "e", "␣e" en position 261 du dictonnaire  
8. 99 ('c') : ajout de "c", "ec" en position 262 du dictionnaire  
9. 104 ('h') : ajout de "h", "ch" en position 263 du dictionnaire  
10. 257 : ajout de "el", "he" en position 264 du dictionnaire  
11. 259 : ajout de "le"

→ "belle echelle" (même dictionnaire que pendant la compression)

Conclusion

Pour conclure,