CAPEL Louna

DE SAINT GERMAIN Arnaud

FROMENT Félix

IBSAINE Brahim

MIFTAH Mohamed Achraf

A l'attention

de Romain DUJOL

Projet Scala

Grande Ecole de sciences,
d'ingénierie, d'économie et de
gestion de **CY Cergy Paris Université**

PAU

2021/2022

Sommaire :

I - Introduction **3**

II - Conception **3**

III - Méthode RLE **4**

Description 4

Implémentation 4

IV - Méthode statistiques **5**

Méthode Huffman 5

Description 5

Implémentation 5

Méthode Shannon-Fano 6

Description 6

Implémentation 6

Comparaison 6

V - Méthode à dictionnaire **7**

Méthode LZ78 7

Description 7

Implémentation 7

Méthode LZW 8

Description 8

Implémentation 9

Comparaison 9

VI - Conclusion **10**

I - Introduction

En cette deuxième année d'école d'ingénieur, nous avons un module appelé Programmation Fonctionnelle. Au cours de ce module, nous avions à réaliser un projet en Scala.

Ce projet avait pour but de nous familiariser avec ce langage de programmation en codant différentes méthodes de compression sans perte.
Les méthodes de compression concernées sont RLE, Huffman, Shannon-Fano, LZ78, LZW. Elles s'appuient sur différents principes connus de compression basés sur les arbres binaires avec des statistiques et les dictionnaires. Les principes et les méthodes ont été divisés, et utilisent des grandes classes communes telles que EncodingTree et StatisticCompressor.

Les premières parties de ce rapport seront consacrées à la mise en place de la conception puis de l'implémentation.
La deuxième partie sera consacrée à l'étude des méthodes de compression.

Les membres de cette équipe sont CAPEL Louna, DE SAINT GERMAIN Arnaud, FROMENT Félix, IBSAINE Brahim et MIFTAH Mohamed Achraf.

II - Conception

Afin de réaliser au mieux ce projet nous avons décidé de multiplier les méthodes. Nous nous sommes répartis les grandes méthodes et chacun d'entre nous a essayé de couper sa partie en différentes fonctions.
Ce découpage avait pour but de faciliter au maximum la conception et également de pouvoir tester chaque conversion, calcul et itération. La compression et la décompression n'étant pas des choses faciles à coder, les petites fonctions nous ont permis d'appliquer la récursivité pas à pas.

De plus, chacun ayant sa vision des choses, nous nous sommes concertés afin d'échanger les points de vue. Ainsi, nous avons essayé de voir quelles étaient les meilleures méthodes trouvées selon deux critères : la complexité temporelle et la simplicité de compréhension.

La complexité d'un tel projet est l'organisation et la conception. Il faut bien se répartir le travail afin de ne pas se gêner et d'avancer au mieux.

III - Méthode RLE

1. Description

La méthode de compression RLE (*Run Length Encoding*) est utilisée par plusieurs formats d'images comme BMP ou TIFF. Elle est basée sur la redondance d'apparition d'éléments consécutifs.

La compression RLE ne fonctionne que pour les données possédant de nombreux éléments consécutifs avec une fréquence d'apparition élevée, notamment les images possédant de larges parties uniformes.

Cette méthode a toutefois l'avantage d'être peu difficile à mettre en œuvre, et applicable sur de nombreux domaines.

Exemple :

- "REELLEMENT" -> (1, R), (2, E), (2, L), (1, E), (1, M), (1, E), (1, N), (1, T)
- "AAAAAHHHHHHHHHHHHHH" -> (5, A), (14, H)

2. Implémentation

Pour cette méthode, nous avons utilisé cinq fonctions. Deux d'entre elles servent à la compression et 3 pour la décompression, la difficulté de cette deuxième a été de retourner une option avec None si le résultat était impossible.

Dans ces fonctions, nous avons utilisé .tail qui renvoie "la queue" la même séquence mise en paramètre sans le premier caractère.

Nous nous sommes servi de .head, pour renvoier le premier élément de la séquence en paramètre, "la tête".

Ensuite nous avons utilisé .foldLeft qui permet de parcourir tous les éléments de la séquence de gauche à droite et d'y appliquer une fonction sur un accumulateur.

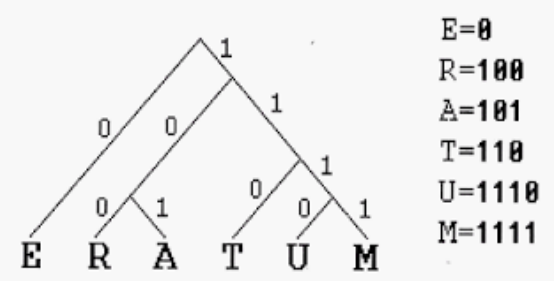
Enfin, ._n qui permet de récupérer le $n^{ième}$ éléments d'un n-uplet.

IV - Méthode statistiques

Le modèle statique lit et compresse en utilisant la fréquence d'apparition de chaque élément. Cette méthode utilise également souvent des tables de probabilités statiques.

1. Méthode Huffman

a) Description



La méthode de compression Huffman se rapproche du morse. Son principe est de coder les éléments les plus fréquents sur peu de place (bits) et ceux plus rares sur des séquences plus longues.

Plus il y a une importante différence entre les fréquences d'apparition des caractères, plus l'algorithme sera efficace. En revanche, le principal inconvénient de cette méthode est qu'elle n'est pas optimale lorsqu'un message contient beaucoup de caractères avec une faible fréquence d'apparition (ou des fréquences similaires).
De plus, afin de pouvoir décompresser un message il faut connaître l'arbre qui a permis de compresser celui-ci. Ce qui nécessite de la place dans la mémoire et un fichier à part pour transmettre l'arbre.

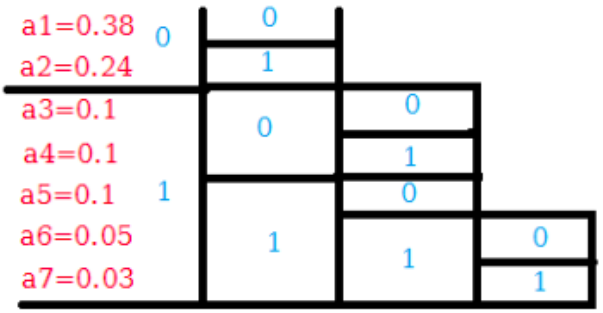
b) Implémentation

Pour cette méthode nous avons utilisé .sortWith(n) qui permet de trier les séquences selon n.

Enfin, nous nous sommes servis de .nonEmpty qui vérifie si la séquence en paramètre est vide ou non et qui renvoie un booléen.

2. Méthode Shannon-Fano

a) Description



Les a_i représente les fréquences d'apparition des éléments

Le codage de Shannon-Fano est un algorithme de compression de données sans perte élaboré par Robert Fano à partir d'une idée de Claude Shannon. Sa construction est très proche de la méthode huffman. Cependant au lieu de créer une branche avec les deux caractères ou sous arbres les moins fréquents en remontant l'arbre comme pour la méthode de Huffman, avec l'algorithme de Shannon-Fano on découpe les occurrences triées en deux parties qui ont une probabilité similaire.

b) Implémentation

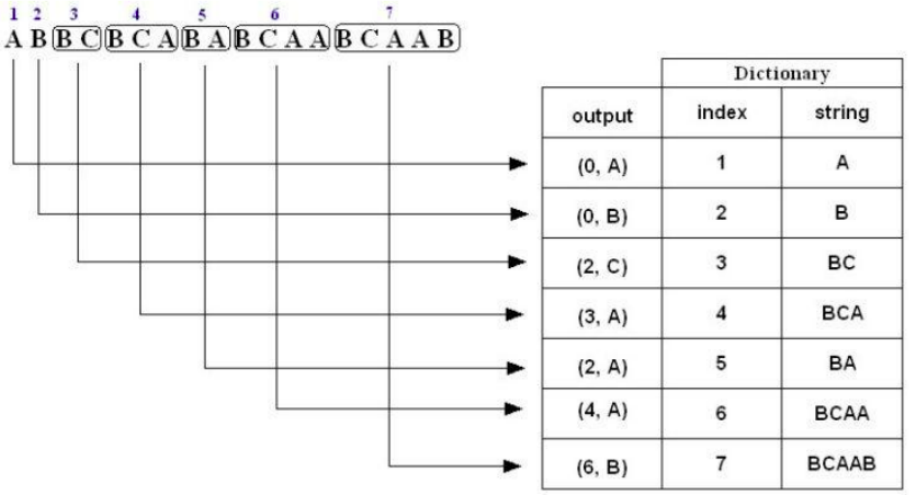
Pour cette méthode, nous avons utilisé .drop(n) qui renvoie la séquence en paramètre, moins les n premiers éléments.

V - Méthode à dictionnaire

Les algorithmes de compression par dictionnaire fonctionnent par la recherche de ressemblance entre le message à compresser et les éléments déjà présents dans leurs dictionnaires. Lorsqu'une séquence, non présente dans le dictionnaire, est trouvée, elle est ajoutée à la suite des autres éléments répertoriés.

1. Méthode LZ78

a) Description



	Dictionary	
output	index	string
(0, A)	1	A
(0, B)	2	B
(2, C)	3	BC
(3, A)	4	BCA
(2, A)	5	BA
(4, A)	6	BCAA
(6, B)	7	BCAAB

La méthode LZ78 part d'un dictionnaire qui est initialement vide et qui se remplit au fur et à mesure de la lecture du message.

Ainsi, la répétition de séquence est préférable pour l'utilisation de cette méthode, chaque nouvel élément prenant une place dans le dictionnaire.

Cette méthode est plus lourde que les autres méthodes, ainsi cette méthode peut demander un temps de calcul plus important.

b) Implémentation

Pour cette méthode, on a utilisé .indexOf(n) qui permet de récupérer l'indice de l'élément n dans la séquence.

Ensuite, nous avons employé .take(n) qui permet de garder les n premiers éléments de la séquence.

Enfin nous avons utilisé .contains(n) qui renvoient un booléen si n est contenu dans la séquence.

2. Méthode LZW

a) Description

Cet algorithme de compression fonctionne de la même manière que LZ78, il cherche la plus longue séquence qui n'est pas dans son dictionnaire pour l'ajouter dans ce dernier.

A la différence de la méthode LZ78, la méthode LZW utilise la table ASCII, qui est déjà présente dans son dictionnaire. Elle demande donc plus de temps de calcul car elle est plus lourde.

De plus, cette méthode renvoie une séquence d'entier, elle ne renvoie aucun char.

Mot lu	Code écrit	Mot ajouté au dictionnaire (+ emplacement)
b		
ba	98	ba, 257
a		
ar	97	ar, 258
r		
rb	114	rb, 259
b		
ba		
bap	257	bap, 260
p		
pa	112	pa, 261
a		
ap	97	ap, 262
p		
pa	261	

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

b) Implémentation

Pour cette méthode nous avons utilisé .last qui permet de renvoyer le dernier éléments de la séquence.

Enfin nous sommes servis de .toString qui permet de convertir la séquence en paramètre en String.

VI - Conclusion

Grâce à ce projet, nous avons pu découvrir des méthodes de compression très différentes, tout en augmentant nos compétences en scala.

Chaque méthode ayant sa particularité, il fallait s'adapter et comprendre le fonctionnement de chaque méthode.

Chacune possède son type de message où elle sera le plus efficace, il est donc important de comprendre la forme du message qu'on cherche à compresser afin d'utiliser la méthode de compression la plus efficace et la plus adaptée.

Nous remercions, Romain Dujol, pour avoir partagé ses connaissances et pour son aide tout au long de ce projet.