CODECS LOSSLESS DE TEXTO

Feito por:

- Pedro Martins (2019216826)
- João Silva (2019216753)

O que são codecs lossless de texto?

Codecs lossless de texto consistem em algoritmos responsáveis por comprimir ficheiros de texto, diminuindo assim o seu tamanho e, consequentemente, o espaço ocupado pelos mesmos. Estes algoritmos podem separar-se em duas categorias, “**lossless**” (sem perda de dados) e “**lossy**” (com perda de dados); como foco do nosso trabalho apenas nos iremos focar no primeiro.

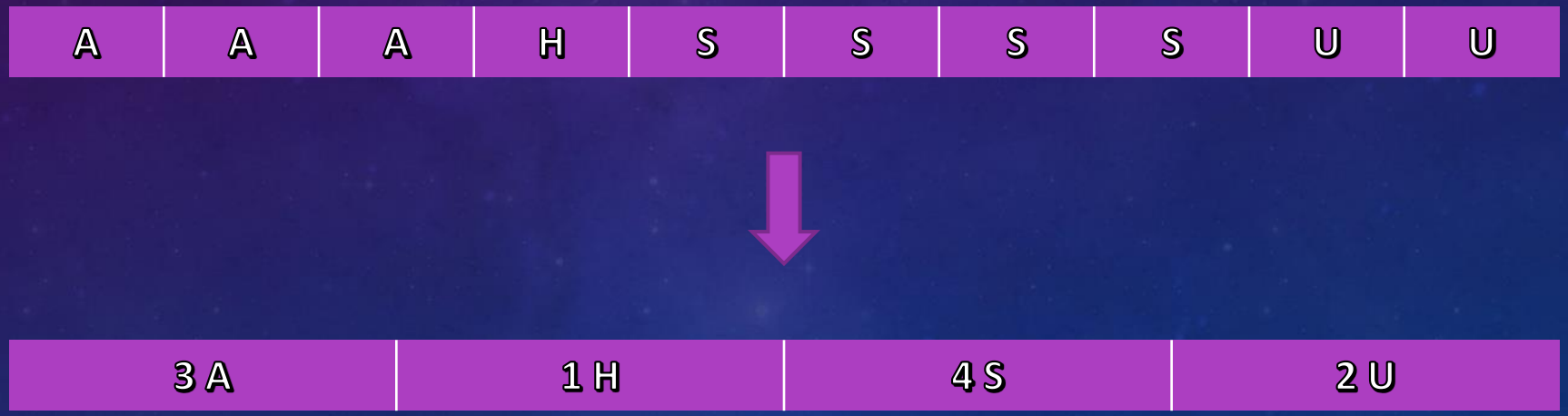
Exemplos destes algoritmos

Em concordância com a categoria de codecs previamente mencionado temos o RLE (Run Lenght Encoding), LZ77, LZW, Huffman Encoding, com alguns exemplos destes algoritmos, sendo estes 4 o principal alvo de discussão mais à frente

RLE (Run Lenght Encoding)

Este algoritmo assenta na procura de repetições (sequência) no texto a analisar.

Dada a string: “AAAHSSSSUU”:



RLE (Run Lenght Encoding)

Quando este algoritmo é utilizado em ficheiros cujas sequências de texto apresentem bastantes repetições, o ficheiro final apresenta um bom rácio de compressão, encurtando de forma considerável a quantidade de dados escritos.

Contudo, caso o ficheiro inicial apresente poucas repetições consecutivas, o ficheiro final não deverá apresentar um nível elevado de compressão, podendo o mesmo ser superior em tamanho face ao inicial.

LZ77

Algoritmo de compressão que recorre ao uso de um dicionário para armazenar as diversas sequências de texto encontradas num ficheiro.

Este método de compressão é recorrentemente usado na compressão de texto de forma lossless. Na sua testagem, usámos um tamanho de janela de 256 e tamanho de buffer de 50.

LZ77

Sequência: “A_ASA_DA_CASA”:

A	_	A	S	A	_	D	A	_	C	A	S	A

(0,0,A)	(0,0,_)	(1,1,S)	(3,2,D)	(2,2,C)	(7,3,EOF)

LZW

Algoritmo de funcionamento semelhante ao LZ77. As novas sequências de texto vão sendo progressivamente armazenadas no dicionário. Este método de compressão tem uma grande eficácia aquando da compressão de ficheiros de texto, sobretudo em comparação com imagens.

LZW

Sequência: “DECODED” (Usando dicionário de tamanho 128)

1. **D – 68** (Armazena DE no dicionário, guarda D no output)
2. **E – 69** (Armazena EC no dicionário, guarda E no output)
3. **C – 67** (Armazena CO no dicionário, guarda C no output)
4. **O – 79** (Armazena OD no dicionário, guarda O no output)
5. **DE – 128** (DE já armazenado no dicionário (posição 128), guarda DE no output)
6. **D – 68** (Armazena DED no dicionário, guarda D)

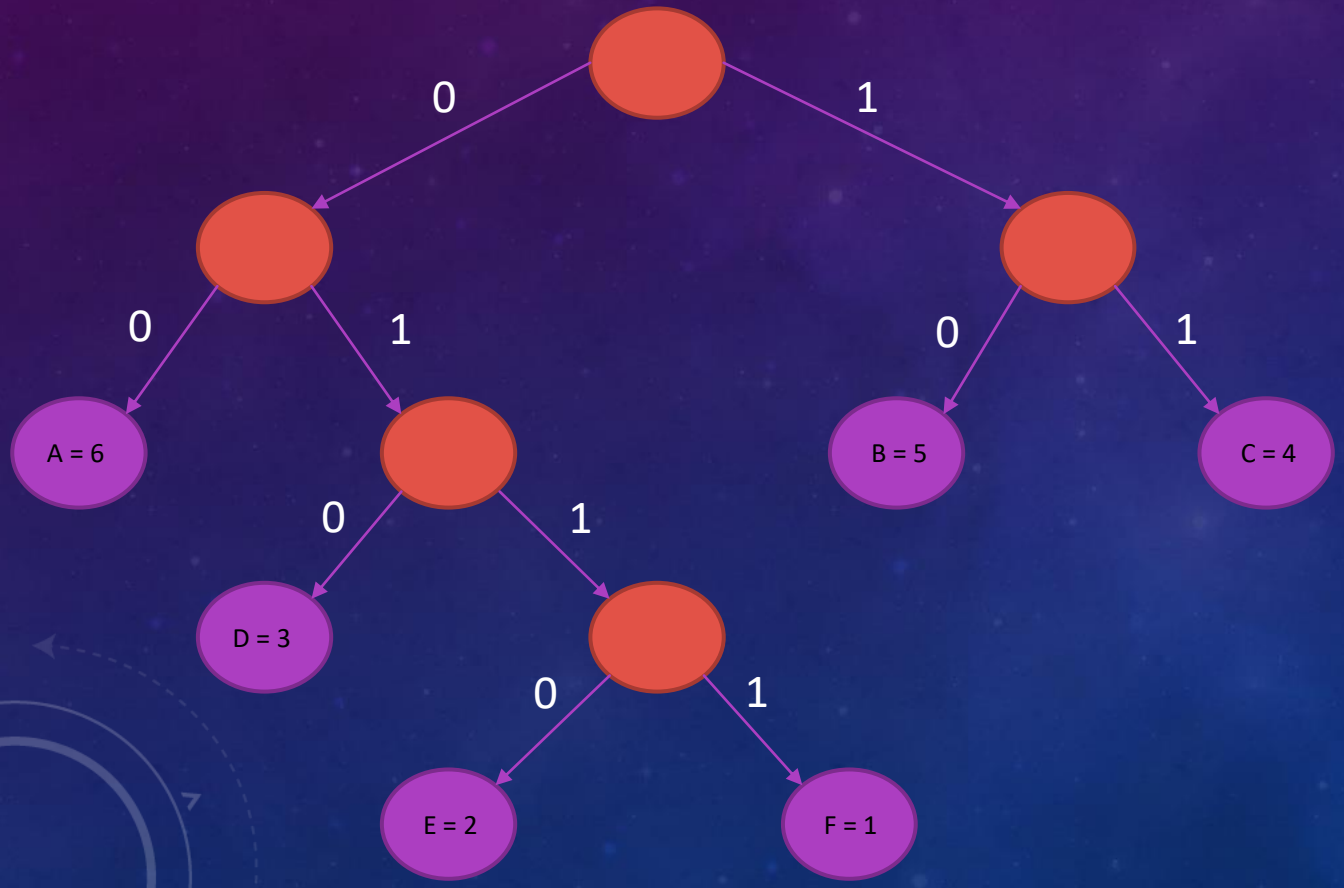
Huffman Encoding

Algoritmo de compressão que utiliza as probabilidades de ocorrência dos símbolos presentes na ‘string’ (neste caso texto) de modo a gerar códigos de tamanho variável para cada caracter que serão armazenados numa árvore binária.

Esta árvore é construída recursivamente a partir da junção dos dois símbolos menos frequentes que são combinados num único símbolo (folhas).

Huffman Encoding

Sequência: “AAAAAABBBBBCCCCDDDEEF”:



Símbolo	Código
A	000
B	001
C	010
D	011
E	100
F	101

Exploração de métodos

De forma a poder chegar a mais conclusões, decidimos aplicar diversos testes aos métodos de compressão por nós aplicados na linguagem Python (4 previamente mencionados, mais Deflate e combinação de RLE e Huffman) com o intuito de descobrir qual o melhor algoritmo a usar aquando da compressão de texto.

Comparação entre tamanho

	Original	RLE	LZ77	LZW	Huffman	Deflate	RLE + Huffman
Bible.txt	3953 KB	7768 KB	3150 KB	2282 KB	2167 KB	2458 KB + 3 KB	4316 KB + 1 KB
Finance.csv	5744 KB	11139 KB	1507 KB	1318 KB	3743 KB	1196 KB + 3 KB	7456 KB + 1 KB
Jquery.js	282 KB	513 KB	222 KB	171 KB	180 KB	174 KB + 3 KB	354 KB + 1 KB
Random.txt	98 KB	193 KB	140 KB	142 KB	74 KB	105 KB + 3 KB	147 KB + 1 KB

Taxa de compressão

A taxa de compressão assenta num valor percentual (%) responsável por representar o quão comprimida um ficheiro fica face ao seu tamanho original.

A seguinte fórmula será utilizada para este cálculo:

$$TaxaDeCompressão(\%) = 100 - \frac{TamanhoComprimido}{TamanhoOriginal} \times 100$$

Comparação das taxas de compressão

	RLE	LZ77	LZW	Huffman	Deflate	RLE + Huffman
Bible.txt	-96.96%	20.31%	42.27%	45.22%	37.82%	-9.18%
Finance.csv	-93.92%	73.76%	77.05%	34.84%	79.18%	-29.81%
Jquery.js	-81.91%	21.28%	39.36%	36.17%	38.30%	-25.53%
Random.txt	-96.94%	-42.86%	-44.89%	24.49%	-7.14%	-50.00%
Média	-92.43%	18.12%	28.45%	35.18%	25.29%	-28.63%

RLE e a baixa taxa de compressão

Dos algoritmos testados o RLE (Run Lenght Encoding) é aquele que apresenta a pior taxa de compressão com o ficheiro final a possuir quase o dobro do tamanho do ficheiro inicial.

Isto deve-se ao facto de que os ficheiros de texto possuíam poucos símbolos repetidos de forma consecutiva e um tamanho considerável, o que origina a má performance por parte do RLE.

Rapidez de compressão

A rapidez de compressão é o penúltimo parâmetro que iremos usar para testar os algoritmos previamente definidos.

Consiste no tempo que demora a efetuar a compressão de um ficheiro (neste caso de texto). Quanto menor o tempo for, mais eficiente o algoritmo é a realizar a compressão.

Comparação da rapidez de compressão

	RLE	LZ77	LZW	Huffman	Deflate	RLE + Huffman
Bible.txt	2.31s	22.34s	1.26s	1.03s	23.56s	2.47s
Finance.csv	3,69s	8.26s	1.80s	1.63s	8.97s	4.09s
Jquery.js	0.16s	1.47s	0.08s	0.07s	1.48s	0.18s
Random.txt	0.06s	0.91s	0.03s	0.02s	1.03s	0.08s
Média	1.56s	8.24s	0.79s	0.69s	8.76s	1.71s

Comparação da rapidez de descompressão

A rapidez de descompressão é o último parâmetro que iremos usar para testar os algoritmos previamente definidos.

Consiste no tempo que demora a efetuar a descompressão de um ficheiro (neste caso de texto). Quanto menor o tempo for, mais eficiente o algoritmo é a realizar a respetiva descompressão.

Comparação da rapidez de descompressão

	RLE	LZ77	LZW	Huffman	Deflate	RLE+ Huffman
Bible.txt	1.03s	0.82s	0.39s	5.22s	6.38s	6.56s
Finance.csv	1.39s	0.49s	0.21s	8.73s	3.17s	10.40s
Jquery.js	0.07s	0.06s	0.03s	0.44s	0.43s	0.50s
Random.txt	0.02s	0.03s	0.02s	0.16s	0.25s	0.21s
Média	0.63s	0.35s	0.16s	3.64s	2.56s	4.42s

Possível trabalho futuro

- Os códigos desenvolvidos para os métodos de compressão LZ77, RLE, Deflate usam a função ‘bytes( )’ do Python o que nos limita à não utilização de números superiores que 256. Seria interessante arranjar uma melhor forma de armazenar as codificações feitas pelos nossos métodos de compressão.

- Também se equaciona uma possível alteração à segunda etapa do processo, o Deflate. Será possível alterar o LZ77 por o mais recente LZ78 e se sim, quais seriam os ganhos associados a esta alteração.

Conclusões

Este relatório teve como objetivo o estudo aprofundado de vários codecs na compressão de texto em múltiplos formatos.

Para isso, criámos um estado de arte com os vários métodos existentes e a partir daí estabelecemos comparações entre si de forma a estudar qual o melhor tanto a nível de taxa de compressão como rapidez (compressão e descompressão).