Teoria da Informação
2º Trabalho Prático
CODEC não destrutivo para Texto

Eduardo Nunes, *2020217675* ,André Moreira, *2020239416* e Diogo Tavares, *2020236566*

Abstract—Neste trabalho prático, exploramos conceitos de Teoria de Informação, em particular, relacionados com a Teoria da Compressão. Pretendemos responder, principalmente, à questão: Quais serão os algoritmos mais eficazes de compressão não destrutiva de texto e as suas principais diferenças?
Para tal, calculamos o limite mínimo teórico para o número médio de bits por símbolo, a Entropia, para cada uma das fontes fornecidas (todas baseadas em documentos de texto) e, seguidamente, comprimimos essas fontes com diversos algoritmos de compressão de texto. Por fim, monitorizamos o número médio de bits por símbolo, verificando, assim, quais destes se aproximam mais da Entropia, de acordo com o CODEC utilizado. Os algoritmos ideais são aqueles que se aproximam mais da Entropia consistentemente nas diversas fontes e não abdicam de muito poder e tempo de processamento para a sua codificação.

Index Terms—Teoria da Informação, IEEE, CODECs, LaTeX, algoritmos, compressão de texto, Python.

1 Introdução

1.1 Problema e a sua importância

Este trabalho visa encontrar uma solução para a compressão eficiente não destrutiva de texto. Um dos pilares da vida humana é **transmitir informação o mais rapidamente possível**, primeiro fisicamente, através de cartas, sinais luminosos, sinais sonoros, impulsos elétricos e**, na atualidade, essencialmente através de bits digitais**. A determinada altura **é certo que iremos chegar a um limite físico da rapidez com que enviamos bits**. Se continuarmos a necessitar de enviar grandes quantidades de informação cada vez mais rapidamente, é necessário encontrar uma maneira de representar a mesma informação com menor número de bits – **é crucial comprimir a *data* a enviar.**

1.2 Implementação

É de realçar que **o foco desta pesquisa não reside na implementação propriamente dita, mas sim na sua fundamentação e validação**. Nesse sentido, decidimos demonstrar o código dos diferentes CODECs[1] com base na linguagem *Python*, devido à sua facilidade de leitura e implementação, caso o leitor pretenda testar os algoritmos apresentados, e ao fácil acesso de módulos que podem auxiliar o tratamento de dados das fontes.

1.2.1 Dados por base

Para responder a esta importante questão, usamos 4 fontes de texto distintas:

1) *bible.txt*: Ficheiro de texto composto por uma transcrição da Bíblia. A sua entropia é 4.34275 bits/símbolo.
2) *finance.csv*: Ficheiro de texto composto por data separada por vírgulas. A sua entropia é 5.15995 bits/símbolo.
3) *jquery-3.6.0.js*: Ficheiro de texto composto pelo código fonte da livraria de *JavaScript*:*Jquery*. A sua entropia é 5.06698 bits/símbolo.
4) *random.txt*: Ficheiro de texto composto por um conjunto de caracteres aleatório. A sua entropia é 6 bits/símbolo.

Para codificar estas fontes, é essencial referir que haverá CODECs melhores do que outros, dependendo da fonte selecionada. Portanto, pretende-se encontrar não um CODEC ideal para uma determinada fonte, mas sim **o mais eficaz geralmente**.

1.3 *State of the art*

Como foi referido inicialmente, para o ser humano viver em sociedade, é indispensável transmitir informação o mais rapidamente possível. Por isso, é fundamental haver um constante avanço (nem que seja mínimo) por parte dos estudos da matéria em causa. O uso de um algoritmo ideal de compressão sem perdas de data é, obviamente, muito dependente do tipo de data em causa.
Atualmente, é pertinente ter em conta o tipo de data a codificar na sua codificação, apesar de existirem CODECs não destrutivos para todo o tipo de data.

1.3.1 Codificação ***Delta Encoding***

Um bom exemplo da importância da dependência de data é observado quando se codifica uma imagem. Como se

- *Docente da cadeira: Paulo Fernando Pereira de Carvalho.*
- *Docente da aula: Marco António Machado Simões*
  Grupo 5 da turma PL6.

Trabalho realizado com base nos conhecimentos obtidos na cadeira da Teoria da Informação.

[1]CODEC significa "Coder-Decoder", algoritmo de compressão de data, de modo a transmiti-la mais rapidamente.

trata de uma imagem, o método *PNG*[2] usa uma versão de *Delta Encoding* ao fazer *Filtering*, onde em vez de se representar todos os valores de uma dada fonte, representa-se a diferença entre eles, o que é muito útil em imagens visto que os pixeis vizinhos estão usualmente correlacionados.

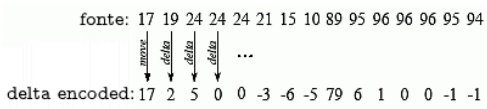

Fig. 1. Exemplo de uma codificação atual para imagens: *Delta Encoding*, a data codificada é obtida pela a diferença do valor anterior. Em imagens (no formato *PNG*), em vez de números à direita, tem-se pixeis com diferentes valores de cores/posições.

1.3.2 Codificação de **Huffman**

O Código de Huffman é **ótimo** devido a nenhum código ser prefixo[3] um do outro. Este é, sem dúvida, um dos métodos de compressão mais conhecidos devido à sua simplicidade, fácil implementação e eficácia. Outra vantagem é que este código não tem patente, daí esta codificação ser **amplamente usada em aplicações de compressão** que vão desde *GZIP*, *PKZIP*, *BZIP2* a formatos de imagem como *PNG* e *JPEG*.

Esta codificação consiste em:

1) Ordenar os símbolos por ordem crescente de ocorrências
2) Construir uma árvore binária até não haver mais símbolos:
    a) Combinar os dois símbolos menos frequentes num único símbolo (cada folha é um símbolo sendo, portanto, um bit com um valor de 1 ou 0)
    b) Acrescentar o novo símbolo à lista em que a frequência é a soma das frequências individuais.

A eficácia desta codificação reside no facto de os símbolos que ocorrem mais vezes serem codificados com menos bits, estando os símbolos com mais ocorrências mais próximos da raiz da árvore binária.

A eficiência do código é dada por:

$$H(S) \leq \bar{l} < H(S) + 1$$

Sendo $H(S)$ a entropia e $\bar{l}$ o número médio de bits.

Apesar da sua grande eficiência, é de extrema importância referir que este código não é perfeito. Além de termos que codificar a árvore binária gerada depois de codificar a fonte, o código de Huffman assume que:

1) A data usada é independente, ou seja, que os valores a comprimir são independentes, o que geralmente não são[4].
2) Tem de existir o modelo da distribuição estatística. Uma resposta para este problema foi o *Modelo Adaptativo de Huffman*.

Para terminar, uma tentativa de otimização da **codificação de Huffman** foi tentar agrupar vários símbolos de modo a aumentar a dependência interna entre eles. No entanto, não é costume fazer isto, porque se um alfabeto tiver m símbolos, um agrupamento n implica um alfabeto com m^n símbolos (o número de símbolos cresce exponencialmente o que em termos de memória não é ideal). Uma solução usada foram os *Códigos Aritméticos*.

1.3.3 Codificação **LZW**

A codificação LZW[5], uma variante do LZ78[6], evita o envio duplo do "codificador" visto que, com este método, o codificador é construido progressivamente ao contrário da variante *LZ78*.

O método consiste em:

1) Criar um dicionário com todos os símbolos do alfabeto.
2) Encontrar a maior sequência de símbolos ante existentes no dicionário, quando surge um novo padrão constituído $a|b$ sendo que a já se encontra no dicionário e ab não.
3) Transmitir o índice do dicionário e criar uma entrada nova ab no dicionário, ficando assim disponível a sequência ab para a sua próxima ocorrência.
4) Continuar a codificação no símbolo b até acabar a cadeia a codificar.

É proveitoso referir que este CODEC foi **um dos primeiros métodos de codificação de texto não destrutivos universalmente usado por computadores**. Um documento de texto em inglês de grande dimensão pode reduzir o seu tamanho original para metade, ao aplicarmos esta codificação! Ainda assim, a implementação pode ser um pouco confusa devido à gestão do dicionário, em termos de eficiência, ao ser lido um novo caracter. Este tem que percorrer todo o dicionário à procura das formações com a cadeia, o que pode ser desgastante para o CPU em causa...

1.3.4 Codificação **Run-Lenght-Encoding**

Este método consiste em representar sequências de valores iguais seguidos de forma eficiente. O algoritmo, aplicável se o comprimento da sequência for maior que 3 para evitar aumentar a data inicial, consiste em substituir conjuntos de letras seguidas pelo seu número de ocorrências em vez de elas todas, como por exemplo:

Data	aaabbbaaaaabbc
Comprimida	3a3b5a2b1c

1.3.5 Codificação **Burrows-Wheeler**

A codificação **Burrows-Wheeler** consiste em, dada uma cadeia, adicionarmos um caracter de controlo no final, criarmos todas as rotações (sendo uma rotação a troca do ultimo caracter em primeiro lugar) da data, organizarmos todas

[2]significa *"Portable Network Graphics"*

[3]Um código diz-se "de prefixo" quando nenhuma palavra é prefixo da outra, ou seja, um código de prefixa é um código unicamente descodificável.

[4]Como foi referido no inicio deste Capítulo, a dependência da data é fundamental quando queremos reduzir a entropia, **é necessário saber o tipo de data que queremos comprimir** porque sabemos que dependências tomar partido de para obter uma codificação ideal

[5]Por extenso, *Lempel-Ziv-Welch*.

[6]Outro método de compressão não destrutivo, feito em 1978 por Lempel-Ziv

as colunas de rotações alfabeticamente e retirarmos a nossa cadeia codificada, sendo esta a ultima coluna das rotações ordenadas alfabeticamente. Apesar de ser preciso guardar um caracter a mais (o caracter de controlo adicionado no final da cadeia em primeiro lugar) para ser feita a descompressão, os caracteres repetidos ficam muito próximos uns dos outros, **criando um ambiente para o uso de outro método por cima deste, tal como o RLE**[7].

Demonstremos a codificação da palavra *PANAMABANANA$*, sendo *&* o caracter mais pequeno:

Todas as Rotações	Rotações Organizadas
PANAMABANANA$	$PANAMABANAN**A**
ANAMABANANA$P	A$PANAMABANA**N**
NAMABANANA$PA	ABANANA$PANA**M**
AMABANANA$PAN	AMABANANA$PA**N**
MABANANA$PANA	ANA$PANAMAB**A**N
ABANANA$PANAM	ANANA$PANAMA**B**
BANANA$PANAMA	ANAMABANANA$**P**
ANANA$PANAMAB	BANANA$PANAM**A**
NANA$PANAMABA	MABANANA$PA**N**A
ANA$PANAMABAN	NA$PANAMABA**N**A
NA$PANAMABANA	NAMABANANA$P**A**
A$PANAMABANAN	NANA$PANAMAB**A**
$PANAMABANANA	PANAMABANANA**$**

Ao retirarmos o ultimo caracter da tabela *Rotações Organizadas*, temos a cadeia codificada:

$$BW(PANAMABANANA\$) = ANMNNBPAAAAA\$$$

1.3.6 Codificação ***Move-to-Front***

Este método de compressão faz com que os caracteres repetidos mais vezes na cadeia fiquem no principio do alfabeto da cadeia codificada.

Conseguimos ver isto em prática com o exemplo da codificação da palavra *"panama"*:

Adicionado	Sequência	Lista do Alfabeto
3	p	amnp
1	pa	pamn
3	pan	apmn
1	pana	napm
3	panam	anpm
1	panama	manp

A lista codificada final será: $[3, 1, 3, 1, 3, 1]$

1.4 Codificações a usar

Para o nosso trabalho, usaremos as seguintes combinações:

1) **Codificação de Huffman**
2) **Codificação de LZW**
3) **Codificação LZW + Huffman** ou **Deflate**, uma junção que normalmente resulta em compressões muito boas, visto que depois de codificar em LZW, um conjunto de bits dado da codificação LZW, como por exemplo, 000, pode ser representado por um só bit 0 se aplicarmos Huffman a seguir
4) **Codificação Move-to-Front + Delta Encoding**, uma combinação também muito eficaz porque ao aplicarmos o Move-to-Front, reduzimos o texto a uma lista em que os números dos índices se repetem mais vezes, o que é propicio para aplicarmos um Delta Encoding. É importante referir que estes dois métodos de codificação não comprimem em si o ficheiro significativamente mas sim "escrevem-no" de outra forma de modo a outros métodos de compressão serem mais eficientes.
5) **Codificação Move-to-Front + Huffman**, outra combinação excelente porque a codificação **Move-to-Front** faz com que os caracteres que estão a ser comprimidos se transformem numa sequência de números de "movidas", sendo uma "movida" os passos que um caracter faz da sua posição atual no alfabeto até ao seu principio. Assim, a lista gerada é, normalmente, uma lista com elementos repetidos diversas vezes, o que torna ideal a utilização da codificação de Huffman que codifica essas repetições para estarem com o menor número de bits possíveis.
6) **Codificação BZIP2**, uma das codificações mais recentes, *open-source* usado extensivamente pela *WEB*. Esta codificação usa diversos níveis de codificação:
   a) *Run-length encoding* (RLE)
   b) *Burrows–Wheeler transform* (BWT)
   c) *Move-to-front* (MTF).
   d) *Run-length encoding* (RLE) do resultado MTF
   e) *Huffman*.
   f) Seleção entre diversas tabelas de Huffman.
   g) Codificação unária (base-1) das tabelas de Huffman
   h) *Delta Encoding* do tamanho de bits da tabela de Huffman.

2 Métodos

Começámos então por explorar o Dataset a comprimir, sendo composto por 4 fontes de texto distintas. Em seguida, calculámos a Entropia[8] de cada fonte e a sua distribuição estatística.

Depois, comprimimos as diferentes fontes usando 6 algoritmos de compressão diferentes. É importante referir, mais um vez, que não procuramos o *CODEC* ideal, mas sim compreender as consequências que este provoca na nossa fonte.

1) Codificação de Huffman
2) Codificação de LZW
3) Codificação LZW + Huffman, também conhecida por Deflate
4) Codificação Move-to-Front + Delta Encoding
5) Codificação Move-to-Front + Huffman
6) Codificação BZIP2

Finalmente, depois dos ficheiros comprimidos, calculámos a taxa de compressão dada pela fórmula:

$$Taxa_{Compressão}(\%) = \frac{Dim_{Original} - Dim_{Comprimida}}{Dim_{Original}}$$

Comparámos também a entropia inicial com a entropia comprimida e verificámos a eficácia da compressão de data

[7] Run-Lenght-Encoding.

[8] limite mínimo teórico para o número médio de bits por símbolo

em causa. Além disso, também tivemos em conta o tempo de compressão e descompressão.

3 RESULTADOS

3.1 Previsões CODECS

No geral, é de esperar que os *CODECS* em que se aplica um ou mais métodos de codificação irão comprimir mais o ficheiro e os métodos de codificação/descodificação que evitem **ordenar** o ficheiro ou **percorre-lo** diversas vezes irão demorar menos tempo.

3.2 Data

Após anotar os dados de compressão e os tempos de *encode* e *decode* com 25 amostras, verificou-se os seguintes resultados:

1) **Ficheiro: bible.txt**
Tamanho data: 4047392 bytes
Tamanho alfabeto: 63
Entropia no pior caso: 5.97728 bits/símbolo
Entropia do ficheiro: 4.34275 bits/símbolo

Este ficheiro é composto por uma transcrição da Bíblia **em inglês**. Se, por um lado, este é um dos maiores ficheiros de texto, visto que é a transcrição total da Bíblia, por outro lado, alguns conjuntos de palavras repetir-se-ão regularmente, uma vez que o texto é escrito em inglês.
Alguns exemplos disso são *and*, *Amen* e *the* que, para compressões como *LZW*, são cruciais para haver uma elevada taxa de compressão.

- **Huffman:**

Tempo de encode	~ 1.49971 segundos
Tempo de decode	~ 7.72618 segundos
Tamanho comprimido	2219301 bytes
Taxa de compressão	45.58%

- **LZW:**

Tempo de encode	~ 1.13927 segundos
Tempo de decode	~ 2.81163 segundos
Tamanho comprimido	1623544 bytes
Taxa de compressão	60.19%

- **Move-to-Front + Delta Encoding:**

Tempo de encode	~ 3.68428 segundos
Tempo de decode	~ 6.72503 segundos
Tamanho comprimido	4047392 bytes
Taxa de compressão	0.75%

- **LZW + Huffman:**

Tempo de encode	~ 1.76540 segundos
Tempo de decode	~ 7.96280 segundos
Tamanho comprimido	1680850 bytes
Taxa de compressão	58.47%

- **Move-to-Front + Huffman:**

Tempo de encode	~ 3.74796 segundos
Tempo de decode	~ 12.03883 segundos
Tamanho comprimido	2424791 bytes
Taxa de compressão	40.54%

- **BZIP2:**

Tempo de encode	~ 0.33608 segundos
Tempo de decode	~ 0.51012 segundos
Tamanho comprimido	846238 bytes
Taxa de compressão	79.25%

2) **Ficheiro: finance.csv**
Tamanho data: 4047392 bytes
Tamanho alfabeto: 67
Entropia no pior caso: 6.06609 bits/símbolo
Entropia do ficheiro: 5.15995 bits/símbolo

Este ficheiro é composto por um texto que contém data que pode ser lida em formato Excel e que está separada por linhas. Como este ficheiro contem muitos números, talvez a compressão BWT+RLE não será muito efeciente, visto que os números não podem ser codificados em RLE devido a ser perdida data no processo.

- **Huffman:**

Tempo de encode	~ 2.37696 segundos
Tempo de decode	~ 14.00243 segundos
Tamanho comprimido	3792043 bytes
Taxa de compressão	35.52%

- **LZW:**

Tempo de encode	~ 1.51367 segundos
Tempo de decode	~ 3.90788 segundos
Tamanho comprimido	1546802 bytes
Taxa de compressão	73.70 %

- **Move-to-Front + Delta Encoding:**

Tempo de encode	~ 6.54668 segundos
Tempo de decode	~ 14.41666 segundos
Tamanho comprimido	5844000 bytes
Taxa de compressão	0.63%

- **LZW + Huffman:**

Tempo de encode	~ 2.27451 segundos
Tempo de decode	~ 10.04949 segundos
Tamanho comprimido	1680850 bytes
Taxa de compressão	68.96%

- **Move-to-Front + Huffman:**

Tempo de encode	~ 6.54847 segundos
Tempo de decode	~ 19.86012 segundos
Tamanho original	4011148 bytes
Taxa de compressão	31.80%

- **BZIP2**

Tempo de encode	~ 0.60914 segundos
Tempo de decode	~ 0.754173 segundos
Tamanho comprimido	189436 bytes
Taxa de compressão	96.78%

3) **Ficheiro: jquery-3.6.0.js**
Tamanho data: 4047392 bytes
Tamanho alfabeto: 97
Entropia no pior caso: 6.59991 bits/simbolo
Entropia do ficheiro: 5.06698 bits/simbolo

Este ficheiro é o código fonte da biblioteca jquery, muito utilizada na WEB. Devido à natureza de um alfabeto de linguagem de programação irá haver palavras que se vão repetir muitas vezes, como a palavra *if* para o programador fazer comparações, *var* ou *const* para o programador definir constantes, variáveis e funções. Além disso, no geral, devido ao seu tamanho reduzido comparado com a *bible.txt* e *finance.csv*, não é possível grandes taxas de compressão visto que é mais difícil retirar o máximo partido do seu contexto efetivamente.

- **Huffman:**

Tempo de encode	~ 0.12802 segundos
Tempo de decode	~ 0.84603 segundos
Tamanho comprimido	185168 bytes
Taxa de compressão	38.17%

- **LZW:**

Tempo de encode	~ 0.09102 segundos
Tempo de decode	~ 0.17636 segundos
Tamanho comprimido	127652 bytes
Taxa de compressão	57.37%

- **Move-to-Front + Delta Encoding:**

Tempo de encode	~ 0.36485 segundos
Tempo de decode	~ 0.68993 segundos
Tamanho comprimido	288580 bytes
Taxa de compressão	3.63%

- **LZW + Huffman:**

Tempo de encode	~ 0.18103 segundos
Tempo de decode	~ 0.62713 segundos
Tamanho comprimido	164447 bytes
Taxa de compressão	45.09%

- **Move-to-Front + Huffman:**

Tempo de encode	~ 0.38109 segundos
Tempo de decode	~ 1.19827 segundos
Tamanho comprimido	195030 bytes
Taxa de compressão	34.87%

- **BZIP2:**

Tempo de encode	~ 0.02701 segundos
Tempo de decode	~ 0.04001 segundos
Tamanho comprimido	68706 bytes
Taxa de compressão	77.06%

4) **Ficheiro: random.txt**
Tamanho data: 4047392 bytes
Tamanho alfabeto: 64
Entropia no pior caso: 6.00000 bits/símbolo
Entropia do ficheiro: 5.99949 bits/símbolo

Este ficheiro é composto por caracteres do alfabeto latino ao acaso. Assim, as repetições que existem são insignificantes e pouco uniformes, sendo o ficheiro que tem maior entropia e menor taxa de compressão. Como a probabilidade de acontecimento das letras é muito uniforme, a sua variância será próxima de 0.

- **Huffman:**

Tempo de encode	~ 0.04901 segundos
Tempo de decode	~ 0.28807 segundos
Tamanho comprimido	76006 bytes
Taxa de compressão	23.99%

- **LZW:**

Tempo de encode	~ 0.05201 segundos
Tempo de decode	~ 0.11703 segundos
Tamanho comprimido	100758 bytes
Taxa de compressão	-0.76%

- **Move-to-Front + Delta Encoding:**

Tempo de encode	~ 0.26306 segundos
Tempo de decode	~ 0.13604 segundos
Tamanho comprimido	100000 bytes
Taxa de compressão	0.00%

- **LZW + Huffman:**

Tempo de encode	~ 0.10903 segundos
Tempo de decode	~ 0.39477 segundos
Tamanho comprimido	107965 bytes
Taxa de compressão	-7.96%

- **Move-to-Front + Huffman:**

Tempo de encode	~ 0.12103 segundos
Tempo de decode	~ 0.35908 segundos
Tamanho comprimido	75862 bytes
Taxa de compressão	24.14%

- **BZIP2:**

Tempo de encode	~ 0.00900 segundos
Tempo de decode	~ 0.05301 segundos
Tamanho comprimido	75684 bytes
Taxa de compressão	24.32%

4 Conclusões

Dos resultados obtidos, é evidente que a performance dos novos métodos de codificação tende a melhorar drasticamente ao longo do tempo, ao comparar, por exemplo, os dados obtidos da codificação **BZIP2** vs **LZW**. Assim, conseguimos ver que a eficiência do *CODEC BZIP2*, devido ao uso de diversos algoritmos na sua criação, quer por tempo de compressão, quer pela sua taxa de compressão, é muito superior aos outros algoritmos. No entanto, com a inovação de novos métodos de compressão, estes saltos na eficiência da compressão de data não destrutiva começam a ser menores e cada vez mais essenciais, visto que a informação nunca foi transmitida mais rapidamente.
Talvez quando chegarmos a um limite físico de velocidade de transmissão, o único método de transmitir data mais rapidamente será **comprimir não destrutivamente eficazmente**.

5 Trabalho futuro

Este relatório foi falado de diversos *CODECs* de **compressão não destrutiva**. Concluímos que os melhores métodos foram aqueles em que são aplicados mais do que um método de codificação.
Apesar disto, se continuarmos a aplicar métodos, vamos chegar a um limite teórico de data mínima para o ficheiro. Será que é possível calcular esse limite? Como? Que conclusões é que podemos tirar ao calculá-lo?

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.
[2] Gajendra Sharm *Analysis of Huffman Coding and Lempel–Ziv–Welch (LZW) Coding as Data Compression Techniques*
[3] https://bit.ly/3d1F1Zq
[4] https://www.geeksforgeeks.org/deflater-deflate-function-in-java-with-examples/
[5] https://bit.ly/3HZqFao
[6] https://bit.ly/311sAe8