Compressão lossless de ficheiros de texto

Trabalho Prático 2 de Teoria da Informação 2021/2022

Gonçalo Almeida
nº 2020218868
Departamento de Engenharia Informática
Universidade de Coimbra
Coimbra, Portugal
gfalmeida@student.dei.uc.pt

João Santos
nº 2020218995
Departamento de Engenharia Informática
Universidade de Coimbra
Coimbra, Portugal
jbsantos@student.dei.uc.pt

Samuel Machado
nº 2020219391
Departamento de Engenharia Informática
Universidade de Coimbra
Coimbra, Portugal
samuel.machado@student.uc.pt

***Abstract*— Nas últimas décadas, grande parte das pessoas de todo o mundo tem-se tornado dependente das tecnologias em diversas áreas da sua vida. A par desta mudança, tornou-se crucial haver uma circulação rápida de grandes quantidades de informação bem como um armazenamento otimizado. Assim, mais do que conveniente, a compressão de dados tem-se tornado essencial. Neste artigo iremos comparar algoritmos que comprimem ficheiros de texto em relação à sua velocidade e taxa de compressão e descompressão, aplicando-os a diferentes formatos de texto.**

Keywords*— *Compressão, Descompressão, Compressão não destrutiva, Velocidade de compressão, Velocidade de descompressão, Taxa de compressão

I. Introdução

A compressão de dados consiste na redução do tamanho de ficheiros para uma menor ocupação de espaço em memória, possibilitando, por exemplo, o envio mais rápido dos mesmos através de canais com uma largura de banda limitada.

Existem dois tipos de compressão: *lossy* e *lossless*. A compressão destrutiva (*lossy*) comprime um ficheiro à custa de parte do mesmo, isto é, após a compressão não é possível reconstruí-lo na íntegra devido à perda de informação na compressão. *JPEG* e *MP3* são exemplos de algoritmos *lossy* que podem ser usados na compressão de imagem e áudio, respetivamente. A compressão não destrutiva (*Lossless*) comprime e descomprime um ficheiro sem se perder informação do mesmo, ou seja, a compressão é reversível ao ponto do descodificador de informação (*decoder)* reconstruir totalmente a informação codificada pelo codificador (*encoder)* [1]. O facto de nenhuma parte da informação ser perdida faz com que a taxa de compressão de um algoritmo *lossless* seja inferior à de um *lossy*. *LZW* e *Huffman Coding* são exemplos de algoritmos de compressão *lossless* que podem ser usados para a compressão de imagens médicas, executáveis e, como vai ser abordada principalmente neste artigo, compressão de texto. [2]

II. Metodologia

Neste artigo, são utilizados diversos algoritmos de compressão *lossless* para comprimir ficheiros de texto. Apresentam-se, em primeiro lugar, algoritmos que servem de base, juntamente com algumas transformadas, para outros algoritmos de compressão mais complexos, que são depois apresentados. Para recolher informação sobre estes algoritmos, foram utilizados o livro [3] e o artigo [4].

A. Algoritmos base

1) *Huffman Coding* – Algoritmo cujo objetivo é criar códigos para cada símbolo, sendo que os símbolos mais frequentes na fonte terão códigos mais pequenos. Estes códigos são gerados a partir da "árvore de *Huffman*" que garante que os códigos sejam unicamente descodificáveis (não existe ambiguidade ao ler uma sequência de símbolos) e também que o descodificador reconheça o fim de cada símbolo sem necessitar do símbolo seguinte (instantaneidade). Os códigos de Huffman são principalmente divididos em dois tipos: (a) fixo, onde uma tabela de frequências dos símbolos é gerada previamente, sendo criada uma única árvore para compressão e descompressão; (b) dinâmico, onde são criadas duas árvores (uma para compressão e outra para descompressão), e estas vão sendo atualizadas ao receber cada símbolo.

2) *Run Length Encoding (RLE)* – Algoritmo que substitui sequências de símbolos iguais por um carater especial, seguido do símbolo e do número de vezes seguidas que aparece.

3) *Lempel-Ziv 77 (LZ77)* - Algoritmo que tira proveito de padrões frequentes numa fonte. Consiste em utilizar dois *buffers*, uma *search window* e uma *look-ahead window*, em que é procurado no primeiro o maior padrão que ocorre no segundo.

4) *Lempel-Ziv 78 (LZ78)* – Algoritmo que utiliza um dicionário construído de forma adaptativa tanto no codificador como no descodificador, tirando proveito da repetição de padrões numa fonte. Os símbolos são codificados com o seu índice no dicionário e o código do carater seguinte.

B. Transformadas

1) Burrows-Wheeler Transform (BWT) – Algoritmo que transforma a sequência original de símbolos numa outra onde os símbolos iguais tendem a ficar juntos. Tendo uma sequência de n símbolos, são geradas n-1 sequências, através dum *shift* cíclico, também de n símbolos; de seguida, ordena-se a primeira coluna por ordem lexicográfica, e a última coluna é o resultado da transformada, sendo que se guarda a linha da sequência original para a descodificação.

2) Move-to-Front (MTF) – Algoritmo cujo objetivo é transformar sequências longas no mesmo símbolo, o que aumenta a probabilidade deste símbolo, e, consequentemente, diminui a entropia. Normalmente, é aplicado um *BWT* antes deste algoritmo, e um *RLE* ou um codificador entrópico depois. Consiste em, tendo uma lista com cada símbolo por ordem lexicográfica, codificar cada um com o seu índice na lista, e depois mover para o início desta.

3) Delta encoding – Algoritmo que transforma sequências de símbolos em diferenças de símbolos adjacentes, isto é, cada símbolo é o resultado da diferença entre o símbolo anterior e ele próprio.

4) Range encoding – Algoritmo semelhante aos códigos aritméticos. Começa por dividir um intervalo proporcionalmente à distribuição de probabilidade de cada símbolo, sendo cada um codificado pelo seu subintervalo; na iteração seguinte, este subintervalo é novamente dividido proporcionalmente à probabilidade de cada símbolo, e sucessivamente.

C. *Algoritmos mais complexos*

1) Lempel-Ziv-Welch (LZW) – Variante do *LZ78* em que é apenas enviado o índice do símbolo no dicionário. Em cada iteração, ao ler-se um símbolo, procura-se a maior sequência desse símbolo na fonte que ainda não exista no dicionário e cria-se uma nova entrada neste.

2) BZip2 – Algoritmo com diversas camadas de codificação: começa por aplicar os algoritmos *RLE*, *BWT* e *MTF*, por esta ordem, seguido de *RLE* novamente; de seguida, *Huffman coding* é aplicado no resultado destes algoritmos, gerando múltiplas tabelas de *Huffman* que são novamente codificadas, através de uma codificação unária de base 1, sendo depois aplicado *Delta encoding*; finalmente, uma tabela de bits é usada para registar os símbolos utilizados.

3) Lempel-Ziv Markov Chain Algorithm (LZMA) - primeiro, aplica *Delta encoding* à fonte; de seguida, utiliza *LZ77* para encontrar padrões repetidos; finalmente, usa *range encoding* no resultado do passo anterior.

4) Deflate - Algoritmo que divide a fonte em blocos, aplicando a estratégia mais adequada a cada um, sendo que estas incluem: não codificar; utilizar árvores de *Huffman* fixas; utilizar árvores de *Huffman* dinâmicas; ou pode ainda ser um bloco reservado. [5]

5) Prediction by Partial Matching (PPM) – Algoritmo que utiliza cadeias de Markov de ordem n, em que, ao percorrer a fonte, vai atualizando as probabilidades dos símbolos baseando-se nos n últimos símbolos.

III. INFORMAÇÕES DO DATASET

Para a realização deste artigo vão ser utilizados os seguintes ficheiros:

TABELA I. LISTA DE FICHEIROS A UTILIZAR

Nome	**Informações acerca dos ficheiros**			
	Descrição	***Entropia (bits/símbolo)***	***Tipo***	***Tamanho (bytes)***
bible	Texto integral da Bíblia (versão King James)	4.343	Texto em inglês	4 047 392
finance	Informação do "annual enterprise survey" da Nova Zelândia (ano 2020)	5.16	Ficheiro CSV	5 881 081
jquery-3.6.0	Biblioteca jQuery (versão de desenvolvimento 3.6.0)	5.067	JavaScript	288 580
random	Coleção aleatória de caracteres	5.999	Texto em inglês	100 000

Analisando os primeiros 3 ficheiros ("bible.txt", "finance.csv" e "jquery-3.6.0.js"), deteta-se uma clara prevalência de uns símbolos sobre outros (Fig. 1-3.). Além disso, há padrões que se repetem frequentemente: em "bible.txt", muitas frases começam da mesma forma; em "finance.csv", as linhas têm o mesmo formato; e, em "jquery-3.6.0.js", há *keywords* que são muitas vezes repetidas. Assim, os algoritmos de compressão apresentados anteriormente tiram grande proveito destas tendências. No entanto, o último ficheiro, "random.txt" (Fig. 4.), é composto por símbolos completamente aleatórios, pelo que os algoritmos de compressão não vão conseguir grandes taxas de compressão, podendo até aumentar o tamanho dos ficheiros, visto que, de um ponto de vista estatístico, não há padrões bem definidos que se repetem (pois tudo é aleatório).

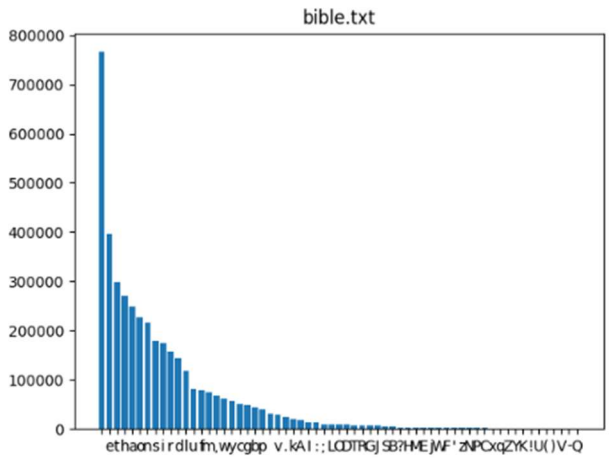

Fig. 1. Histograma de distribuição das ocorrências de cada símbolo de bible.txt

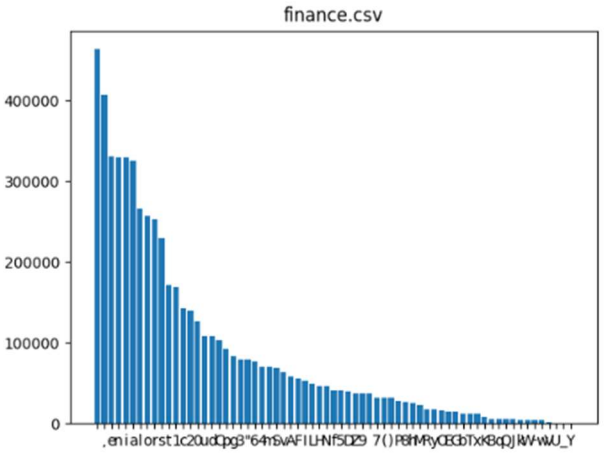

Fig. 2. Histograma de distribuição das ocorrências de cada símbolo de finance.csv

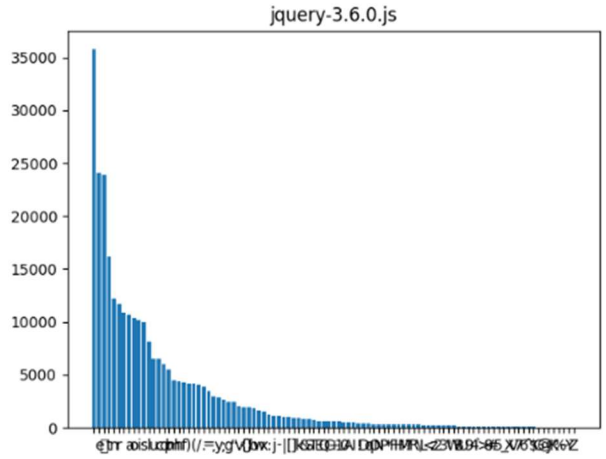

Fig. 3. Histograma de distribuição das ocorrências de cada símbolo de jquery-3.6.0.js

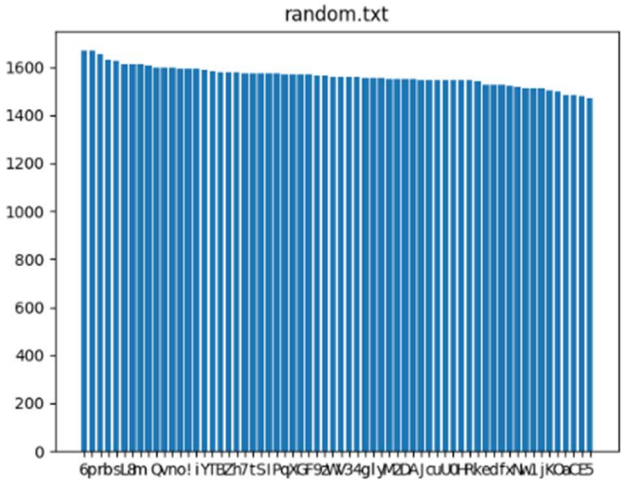

Fig. 4. Histograma de distribuição das ocorrências de cada símbolo de random.txt

REFERÊNCIAS

[1] Bell, T., Witten, I. H., & Cleary, J. G. (1989). Modeling for text compression. ACM Computing Surveys (CSUR), 21(4), 557–591.

[2] Kodituwakku, S. R., & Amarasinghe, U. S. (2010). Comparison of Lossless Data Compression Algorithms. Indian Journal of Computer Science and Engineering, 1(4), 416–425.

[3] K. Sayood. (2000). Introduction to data compression: second edition, Morgan Kaufman

[4] Gupta, A., Bansal, A., & Khanduja, V. (2017). Modern lossless compression techniques: Review, comparison and analysis. Proceedings of the 2017 2nd IEEE International Conference on Electrical, Computer and Communication Technologies, ICECCT 2017, Fevereiro.

[5] A. Feldspar (1997, 23 Agosto). An Explanation of the Deflate Algorithm. Acedido em: Nov. 23, 2021. [Online]. Disponível: https://www.zlib.net/feldspar.html