CODECS NÃO DESTRUTIVOS PARA TEXTO

Estado de Arte

Bruno Sequeira
Departamento de Engeharia Informática
Universidade de Coimbra
Coimbra, Portugal
uc2020235721@student.uc.pt

Rodrigo Figueiredo
Departamento de Engeharia Informática
Universidade de Coimbra
Coimbra, Portugal
uc2020236687@student.uc.pt

Rui Santos
Departamento de Engeharia Informática
Universidade de Coimbra
Coimbra, Portugal
uc2020225542@student.uc.pt

***Abstract*—Com o aumento de informação textual disponível na internet, como em bibliotecas digitais e bancos de dados de documentos, têm-se verificado uma grande necessidade de comprimir esta informação. A compressão de dados é a arte de representar informação de forma compacta. É uma técnica muito útil que ajuda a reduzir o tamanho da informação e a guardá-la em menos bits, reduzindo assim o espaço que esta ocupa, facilitando assim a sua transmissão. Existem diversos métodos que permitem a compressão de dados, que se podem categorizar por Lossless(não destrutiva) e Lossy(destrutiva).**

Keywords—compressão, lossless

I. INTRODUÇÃO

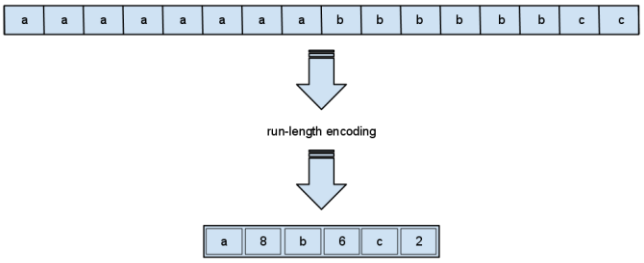
A compressão de dados tem como objetivo reduzir o espaço ocupado por dados num determinado dispositivo. Durante anos têm surgido vários métodos e algoritmos de compressão de modo a preservar esse espaço, mas, não se sabe até que ponto esses métodos são mais eficientes que outros. Quanto maior a eficiência desses métodos e quanto maior o número de casos em que se podem aplicar, melhor s ã o .

II. TÉCNICAS E MÉTODOS DE COMPRESSÃO ATUAIS

Vamos abordar alguns dos algoritmos de compressão mais usados. Estes dividem-se em métodos estatísticos, onde é usada a probabilidade de cada símbolo para obter um código para esse mesmo símbolo, e em métodos de dicionário, onde uma sequência de símbolos é substituída por um ponteiro que aponta para uma ocorrência anterior dessa sequência.

A. RLE(RUN LENGTH ENCODING)

Esta é uma técnica de compressão muito simples principalmente usada em dados com muitas repetições de valores. Este método consiste em agrupar símbolos contíguos iguais num par símbolo-número de ocorrências.



Porém, este tipo de compressão não é muito eficaz num texto escrito naturalmente em língua portuguesa, uma vez que dificilmente encontramos repetições de duas letras consecutivas. No entanto, existem técnicas que podem comprimir mais facilmente textos com o uso de RLE como por exemplo o bzip2.

B. CODIFICAÇÃO DE HUFFMAN

Este é um dos métodos mais bem sucedidos no que toca à compressão de texto. É calculada a frequência de distribuição dos caracteres da fonte de forma a calcular a distribuição de probabilidade. Depois, a partir das probabilidades, agrupam-se os dois símbolos com menor probabilidade e cria-se um novo símbolo cuja probabilidade vai ser a soma da dos dois. Depois continuamos a repetir este processo até a soma das probabilidades ser um.

C. BTW ou Borrows wheeler

Este é um método de processamento de um bloco de dados que aumenta a redundância espacial, facilitando assim a aplicação de outros métodos de compressão. Para aplicar este método constrói-se em primeiro lugar uma matriz n x n onde n é o tamanho do bloco a ser comprimido. A primeira linha desta matriz será o bloco original e nas restantes esse mesmo bloco mas rotacionado para a esquerda de um em um à medida que avançamos pelas linhas da matriz. O segundo passo será ordenar as linhas em ordem lexicográfica . No fim tiramos a letra que está na última coluna de cada linha da matriz e essa será a nossa cadeia final.

```
a_asa_da_casa       0: _asa_da_casaa
_asa_da_casaa       1: _casaa_asa_da
asa_da_casaa_       2: _da_casaa_asa
sa_da_casaa_a       3: a_asa_da_casa <- linha original
a_da_casaa_as       4: a_casaa_asa_d
_da_casaa_asa       5: a_da_casaa_as
da_casaa_asa_       6: aa_asa_da_cas
a_casaa_asa_d       7: asa_da_casaa_
_casaa_asa_da       8: asaa_asa_da_c
casaa_asa_da_       9: casaa_asa_da_
asaa_asa_da_c       0: da_casaa_asa_
saa_asa_da_ca       1: sa_da_casaa_a
aa_asa_da_cas       2: saa_asa_da_ca
```

Cadeia final: "aaaadss_c__aa"

D. *MOVE TO FRONT*

Este método de compressão é usado para melhorar a performance da entropia de compressão. Quando é eficientemente implementado, é tão rápido que é justificável aplica-lo como um passo extra na compressão de dados.
Consiste em codificar uma string substituindo cada símbolo com a sua respetiva posição no array. Em cada substituição será realizada a posição desse símbolo no array, sendo este movido para o inicio do mesmo. Assim, no final, os símbolos que foram mais recentemente usados estarão no inicio do array, aumentando assim a probabilidade de cada símbolo diminuindo assim a entropia.

E. *LZW*

Este é um algoritmo de compressão de dados baseado no registo e localização de padrões de uma estrutura. O algoritmo LZW visa eliminar a necessidade de se emitir um caractere junto do endereço de dicionário.
Este algoritmo consiste em comparar simbolo a simbolo uma mensagem, verificando se esse simbolo já existe no dicionário. A cada combinação nova encontrada, é criada uma entrada no dicionário para ser utilizada em futuras comparações.

F. *LZ77*

Define-se em duas estruturas que serão usadas, janela de procura e buffer de look-ahead). A janela representa as partes do arquivo que já foram lidos, enquanto o look-ahead representa o que ainda será lido e processado pelo algoritmo. A janela tem tamanho definido, permitindo que os dados sejam colocados dentro dela, eliminando os bytes mais antigos quando o seu limite de tamanho é atingido. O buffer também tem tamanho definido mas em geral é dezenas de vezes menor que a janela.

III. Combinações de algoritmos

Podemos também juntar vários algoritmos numa certa ordem para obter um valor ainda melhor de compressão, em comparação com o uso de algoritmos individuais.

A. *BWT + RLE*

Pelo facto de que BWT é um método que facilita a aplicação de outros métodos de compressão, juntando assim todos os caracteres por ordem lexicográfica, poderemos assim combinar com o RLE, sendo este uma técnica de compressão muito simples, pois consiste em agrupar símbolos contíguos iguais num par símbolo - numero de ocorrências, por exemplo “aaassbbb” -> “a3s2b3”

B. *Bzip2*

É uma combinação de RLE, BWT, MTF e Huffman. Começamos por usar um RLE nos dados iniciais, seguidamente aplicamos o BWT de forma a juntar os caracteres com contextos semelhantes. De seguida usamos o MTF, voltamos a comprimir os dados com o RLE e finalmente aplicamos a codificação de Huffman.

C. *Gzip*

É o método de compressão mais popular usado nos servidores da WEB sendo baseado no algoritmo deflate que por sua vez é uma combinação de LZ77 e da codificação de Huffman, sendo assim uma técnica de compressão muito veloz. Usado principalmente em arquivos de código, o gzip pode reduzir o tamanho de arquivo de JavaScript, CSS e HTML até 90%.

IV. Ficheiros a testar

A. *Random.txt:*

Este ficheiro é gerado com letras do alfabeto escritas aleatoriamente, sendo assim sabemos que cada letra vai ter a mesma probabilidade que as outras de aparecer no momento em que o ficheiro é criado. Podemos então prever que a discrepância das ocorrências entre as letras será bastante baixa e podemos observar isso a partir da sua variância. Como a probabilidade de acontecimento das letras é bastante aproximada então a sua variância será aproximada a um número natural também.

Variância = 0.0
Entropia: 5.99948840028008

B. *Bible.txt:*

Este ficheiro é obtido a partir da biblia então é de se esperar que em texto corrido haverão caracteres com ocorrências bastante diferentes das demais letras, como é possível observar neste ficheiro e na sua variância. Sendo que há um número elevado de carcateres diferentes este terá também uma entropia sensivelmente grande.

Variância: 2.6963997748077766
Entropia: 4.342751389250247

C. *Finance.csv*

É um ficheiro obtido a partir de uma tabela excel com bastantes dados e diferentes tipos de caracteres. Neste excel verificamos que há vários dados semelhantes onde poucas coisas são alteradas, sendo assim há bastantes ocorrências semelhantes logo terá uma variância menor em relação à

bíblia. A entropia será um pouco menor que a do random pois haverão caracteres com uma probabilidade bastante menor que outros.

Variância: 1.7956654819183129
Entropia: 5.159949825442126

D. jquery-3.6.0.js

Este ficheiro trata-se de um código em javascript. Um código tem sempre bastantes palavras que se repetem como “ var “ ou “ if “ , mas há bastantes palavras que são diferentes como nomes de variáveis. Sendo assim haverão caracteres com uma ocorrência bastante maior que outros o que aumenta a sua variância. A sua variedade de caracteres não será muito diferente em relação a outros ficheiros como o finance, logo a sua entropia será parecida.

Variância: 2.532242671931982
Entropia: 5.066983843372855

V. Previsão

Gzip com “jQuery-3.6.0.js”, pelo facto de que o conteúdo do ficheiro é um código em JavaScript e como o método Gzip é usado principalmente em arquivos de códigos, sendo que pode reduzir o tamanho de arquivo de JavaScript em 90%, é o método mais ideal.

LZW com “Finance.csv”, pelo facto de que o ficheiro contem vários padrões semelhantes onde poucas coisas foram alteradas, tendo assim bastantes ocorrências, e sendo o LZW é um algoritmo de compressão de dados baseado no registo e localização dos padrões é o método mais ideal.

Codificação de Huffman com “biblie.txt”, como o ficheiro tem uma maior discrepância entre valores de ocorrência e a codificação de Huffman ter melhor eficiência nesse aspeto, é o método mais ideal.

Combinação BWT + RLE com “random.txt” - sendo que o ficheiro contém vários símbolos aleatórios, tendo estas probabilidades equivalentes de ocorrer, usamos o BWT para agrupar os símbolos iguais e de seguida o RLE para agrupar os símbolos iguais num par símbolo.

VI. Aplicação dos métodos de compressão

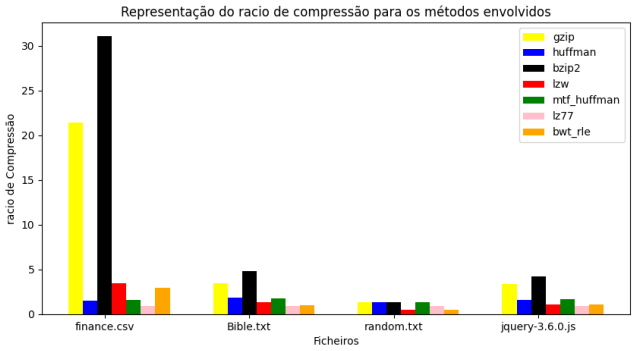
Referindo todos estes métodos, procedemos então à aplicação dos mesmos em cada um dos ficheiros. Obtivemos resultados bastante diferentes para cada um dos métodos e para o que tínhamos previsto no ponto V.
No fim da aplicação de cada método em cada ficheiro obtivemos os seguintes gráficos que mostram o rácio de compressão e o tempo de compressão.

O rácio de compressão é obtido a partir do seguinte cálculo:

$$\text{Compression Ratio} = \frac{\text{Uncompressed Data Rate}}{\text{Compressed Data Rate}}$$

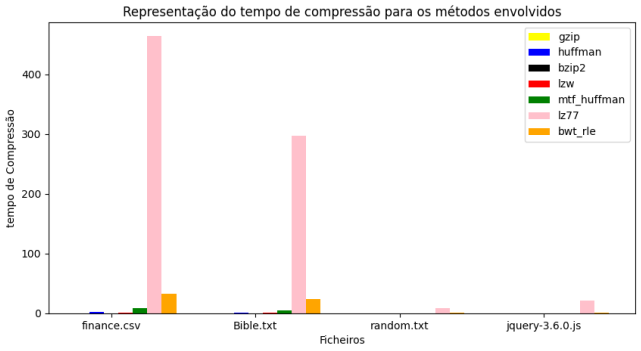
Tendo o tamanho do ficheiro original e do ficheiro comprimido, conseguimos obter o rácio de compressão que nos indica o quão mais pequeno o comprimido ficou em relação ao original.
Tendo isto em conta, apresentamos então os gráficos:

A. Valores dos rácios de compressão:



Ficheiros/ métodos	Huff.	Mtf+huff	Bzip2	gzip	lzw	Lz77	Bwt+rle
Finance.csv	1.55	1.595	31.045	21.393	3.50	0.925	2.997
Bible.txt	1.82	1.789	4.786	3.431	1.36	0.947	1.023
Random.txt	1.33	1.351	1.321	1.319	0.52	0.889	0.524
Jquery-3.6.0.js	1.56	1.653	4.190	3.376	1.11	0.965	1.110

B. Valores dos tempos de compressão:



Ficheiros/ métodos	Huff.	Mtf+huff	Bzip2	gzip	lzw	Lz77	Bwt+rle
Finance.csv	2.26	8.70	5.53	0.09	1.55	463.8	32.63
Bible.txt	1.40	4.82	0.30	0.45	1.05	296.6	24.23
Random.txt	0.04	0.15	0.01	0.005	0.03	8.42	0.63
Jquery-3.6.0.js	0.10	0.36	0.02	0.03	0.06	20.60	1.69

VII. Analíse dos gráficos

A. *Finance.csv*

Neste ficheiro o melhor método de compressão foi o bzip2. Verifica-se que teve o maior rácio de compressão. Apesar do tempo não ser melhor, comparando com os demais métodos, continua a ter uma boa eficiência.

B. *Bible.txt*

Neste ficheiro o melhor método foi também o bzip2 assim com o seu tempo de compressão.

C. *Random.txt*

Pela análise da tabela, podemos observar que 4 dos métodos aplicados obtiveram rácios de compressão bastante próximos. Com isto, passamos a analisar então os tempos. Observando a tabela de tempos, verificamos que o gzip teve um tempo de compressão bastante curto, sendo este então um dos melhores métodos aplicados neste ficheiro.

D. *Jquery-3.6.0.js*

Neste ficheiro o bzip2 teve também o melhor tempo e rácio de compressão

VIII. Conclusão

Concluímos então que o melhor método de compressão, entre todos os referidos neste artigo, é o bzip2, tendo sido o melhor, em todos os casos, em questão de rácio. O que faz sentido, sendo um dos métodos de compressão mais usados para compressão de dados. Sendo utilizado pelo 7zip, WinRar, etc.

IX. Trabalho futuro sugerido

Encontrar um método de compressão ainda mais eficiente, que sirva universalmente para todos os tipos de dados.

X. Referências

[1] https://en.wikipedia.org/wiki/Bzip2
[2] https://pt.wikipedia.org/wiki/Gzip
[3]https://pt.wikipedia.org/wiki/M%C3%A9todo_de_Burrows-Wheeler
[4] https://pt.wikipedia.org/wiki/Codificação_de_Huffman
[4] ficheiros fornecidos pelo professor
[5] slides das aulas teóricas
[6]https://towardsdatascience.com/huffman-encoding-python-implementation-8448c3654328
[7]https://github.com/LLcoolNJ/LZ77/blob/master/src/lz77.py
[8]https://github.com/TiongSun/DataCompression/blob/master/BWT.ipynb
[9] https://docs.python.org/3/library/bz2.html
[10] https://stackabuse.com/run-length-encoding/
[11]https://stackoverflow.com/questions/6834388/basic-lzw-compression-help-in-python
[12] https://en.wikipedia.org/wiki/Move-to-front_transform