Codecs lossless de imagem

Pedro Martins
Universidade de Coimbra
Coimbra, Portugal
Pedro.afonso2001.pm@gmail.com

João Silva
Universidade de Coimbra
Coimbra, Portugal
j0a0carll21@gmail.com

***Abstract*— Documento sobre os vários tipos de codecs lossless para a compressão de uma imagem e respetivas vantagens e desvantagens associadas a cada um dos algoritmos mencionados.**

Keywords*— *lossless, codecs, RLE, Huffman Code, LZ_77, LZ_78, LZW, GOLOMB, CALIC, PNG, Deflate.

I. INTRODUÇÃO

Ao longo deste relatório iremos analisar os diversos métodos de codificação de imagem no formato .bmp (tipo não-comprimido), de forma a reduzir drasticamente o tamanho ocupado pelos mesmos. Estes métodos serão do tipo *lossless* uma vez que não irão alterar de forma significativa a qualidade de imagem. Também iremos proceder à análise de diversos algoritmos concebidos com o intuito de ajudar no processo de compressão e compará-los com o método PNG.

II. CODECS LOSSLESS ANTIGOS

Existem diversos codecs que procedem à codificação de informação sem perdas de qualidade. Nesta lista estão representados os algoritmos mais antigos.

A. Run Lenght Encoding (RLE):

Consiste na compressão de uma imagem de forma sequencial. Procura repetição de símbolos de forma a diminuir a repetição do mesmo sinal e assim levar a um aumento da compressão e respetiva diminuição do tamanho**[1]**. Na Figura 1 está presente um exemplo da aplicação deste algoritmo.

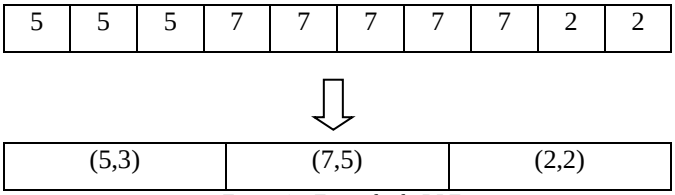

Figura 1 – Exemplo do RLE

Este tipo de compressão não é o mais eficaz contudo, é relativamente simples de executar e leva a uma diminuição do ficheiro final. Os resultados dependem bastante de ficheiro para ficheiro.

B. Huffman Code:

É um método de compressão que contabiliza um número de ocorrências de cada simbolo e cria uma árvore binária a partir desses dados criando códigos de tamanho variável para cada símbolo**[2,3]**.

Primeiramente, os símbolos são ordenados por ordem crescente. De seguida, procede-se à construção de uma árvore binária onde os dois símbolos menos frequentes são combinados num único símbolo (folhas). Por fim, acrescentar o novo símbolo à lista em que a frequência de ocorrência é a soma das frequências individuais. Esta iteração ocorre até que exista um único símbolo. Na Figura 2 está representado este algoritmo.

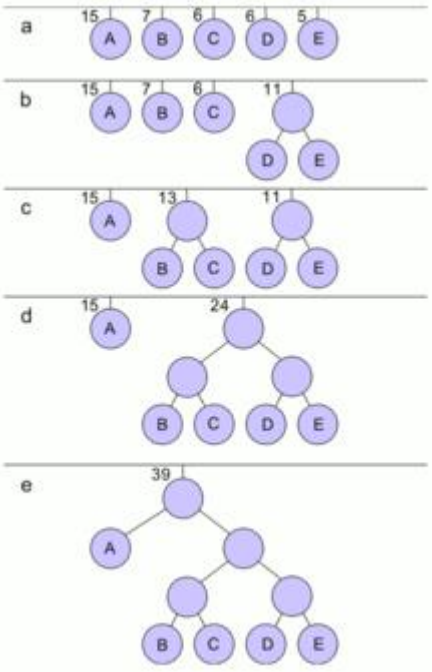

Figura 2 – Representação de uma árvore de Huffman**[2]**

Algumas vantagens a apontar**[15,16]**:

- Não necessita de conhecer a estatística dos símbolos fonte.
- Se a estatística variar, o código adapta-se automaticamente.

Algumas desvantagens a apontar:

- O poder obter diversas árvores binárias para os mesmos dados de entrada
- Pouca tolerância a falhas de descodificação
- Pouco eficiente para alfabetos reduzidos

C. LZ_77:

O algoritmo de LZ77**[4]** baseia-se na utilização das partes que já foram lidas de um arquivo como um dicionário, substituindo as próximas ocorrências das mesmas sequências

de símbolos pela posição (absoluta ou relativa) da sua última ocorrência. Sendo ainda mais eficaz para texto do que para imagens

Segue-se o algoritmo de funcionamento para a codificação LZ_77:

NOTAS PARA A LEITURA DO ALGORITMO:

- A janela lê-se da esquerda para a direita mas o indice aumenta da direita para a esquerda.

- O buffer lê-se da esquerda para a direita e o indice aumenta da esquerda para a direita.

- A sequência tem de começar desde o primeiro símbolo do buffer e é sempre a maior existente.

- Indice da sequência é onde a sequência começa pela primeira vez.

Enquanto não chegar ao fim do ficheiro:
dicionário = null
buffer = n primeiros símbolos (tamanho n bytes)
janela = null (tamanho m bytes)

Enquanto existir símbolo para codificar:
Se existir uma sequência presente no buffer que esteja também na janela:

c = próximo símbolo depois da sequência
Enviar código (i(sequência), len(sequencia, c)
janela = janela + sequência + c
atualizar dicionário com sequencia + c

Senão:

c = primeiro símbolo no buffer
Enviar código (0,0 c)
janela = janela + c
atualizar dicionário com c

buffer = n primeiros símbolos depois de c

De seguida está representado um exemplo do método de funcionamento deste algoritmo:

Janela	Buffer	Resto	Tuplo	Detetado	Somado
	A_AS	A_DA_CASA	(0,0,A)	---	A
A	_ASA	_DA_CASA	(0,0,_)	---	_
A_	ASA_	DA_CASA	(1,1,S)	A	AS
A_AS	A_DA	_CASA	(3,2,D)	A_	A_D
A_ASA_D	A_CA	SA	(2,2,C)	A_	A_C
ASA_DA_C	ASA		(7,3,EOF)	ASA	ASA

Algumas desvantagens a apontar: a dimensão do lookahead buffer condiciona a máxima dimensão da sequência a codificar, quanto maior a janela maior a compressão mais tempo de processamento e aumento do número de bits gastos nos componentes posição e tamanho.

D. LZ_78:

O algoritmo de LZ78**[5,6]**, à semelhança do LZ77, usa um dicionário para armazenar as sequências de símbolos encontradas no arquivo, e usa os códigos (posição das sequências no dicionário, ou mesmo um número atribuído sequencialmente à sequências de caracteres encontradas). Sendo ainda mais eficaz para texto do que para imagens.

Segue-se o algoritmo de funcionamento para a codificação LZ_78:

Enquanto não chegar ao fim do ficheiro:
dicionário = null
word = null
c = null

Enquanto existir símbolo para codificar:

Se word + c é uma sequência presente no dicionário:
word = word + c
c = próximo símbolo

Senão:

Enviar código (índice(word), c)
Atualizar dicionário com word + c
word = null

E. LZW:

A codificação LZW**[7,8]** baseia-se num dicionário que contém os diversos símbolos a ser codificados. No início, o dicionário encontra-se vazio, começando a aumentar de tamanho à medida que novas combinações de símbolos vão sendo obtidas ao ler a imagem. Quando se encontra uma combinação de símbolos com o mesmo tamanho e a mesma ordem de símbolos, irá-se adicionar 1 nova ocorrência à sua entrada no dicionário. Assim, no final da leitura, o ficheiro final encontra-se mais comprimido e ocupa menos espaço.

Este método de codificação é ideal quando existem diversos padrões de símbolos que se repetem na imagem e que se encontram vastamente distribuídos. Também é uma técnica relativamente fácil de ser aplicada e que apresenta uma rápida taxa de compressão. Contudo é uma das técnicas sem perda de qualidade mais antigas e é propícia a criar entradas de dicionário que nunca são usadas, diminuindo a compressão total da imagem. Esta técnica é mais usada na compressão de texto.

F. GOLOMB:

Este algoritmo**[9]** é um dos algoritmos mais simples existentes. Precisando apenas de um parametro inicial M, escolhido por nós, consoante o tipo de compressão que procuramos, é calculada uma parte q e uma parte r do número inicial as quais no final serão concatenadas num novo código que de certo ocupará menos bits que o original.

Segue-se o algoritmo de funcionamento para a codificação GOLOMB:

Enquanto não chegar ao fim do ficheiro:
array_N = null
M = m (escolher um número natural)
b = floor(log2 (M))

Enquanto existir símbolo para codificar:
q = floor(c/M)*1+0
r = n mod M
Se r < 2^b-M:
r usa b bits
Senão:
r = r + 2^(b+1) - M usando b bits (soma)
Atualizar array_N com q + r (concatenação)

De seguida está representado um exemplo da codificação GOLOMB:

Se quisermos codificar por exemplo o número n=16:

q = floor(n/M) = floor(16/10) = 10
r = n mod M = 16 mod 10 = 6 => 6+2^4-10 = 12
valor final = q + r = 10 + 1100 = 101100

III. CODECS LOSSLESS MAIS RECENTES:

São vários as técnicas utilizados na codificação de imagens sem perda de qualidade. De seguida é apresentada uma lista dos algoritmos mais recentes e complexos:

G. CALIC-a:

CALIC-a**[10]** é um método de compressão de imagem sem perda de qualidade que obtém contexto através de uma análise extensiva e de seguida recorre ao codificador aritmético de entropia como principal forma para realizar a compressão.

Numa primeira etapa, irá-se proceder à busca do contexto, usando um método de predição não linear, que irá reduzir a complexidade e o tempo de procura, obtendo uma imagem com perda de qualidade e um resíduo, de forma a poder devolver a qualidade à imagem a ser comprimida

Com o codificador aritmético**[11]** os diversos pixeis são codificados através de um determinado número de bits, sendo que os pixeis que mais vezes se repetem são codificados com um menor número de bits e os que se repetem menos são codificados com um maior número de bits. Este algoritmo, ao contrário da codificação de Huffman, codifica a imagem inteira com 1 número, obtendo uma taxa muito maior de compressão. De forma a alcançar esse objetivo, irá representar a informação entre dois números através de uma probabilidade. Na Figura 3 encontra-se representado um exemplo da aplicação de um codificador aritmético.

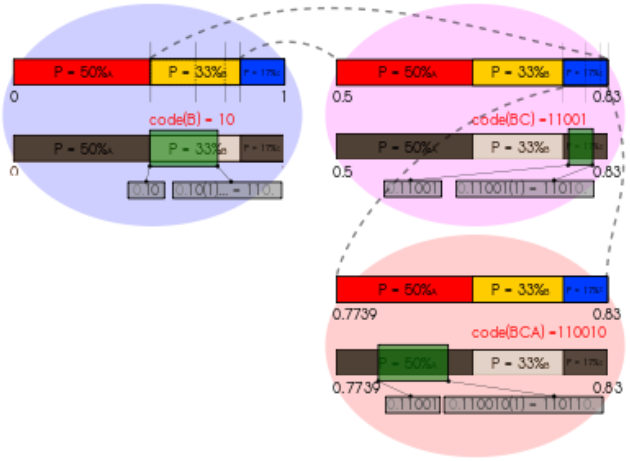

Figura 3 – Esquema do codificador aritmético**[11]**

H. PNG:

PNG**[12,13]** (Portable Network Graphics) é um dos métodos mais conhecidos e usados para compressão de imagens sem qualquer perda de qualidade. Como tal, o processo é dividido em duas etapas:

Filtração: A ideia que está na base da filtração é o conceito de que qualquer pixel numa dada linha ou matriz analisada pode ser rescrito como uma comparação dos pixeis à sua volta. Esta flitração pode ser subdividida em vários submétodos que de acordo com a fonte a ser analisada pode originar resultados distintos, tais como:

- Diferença entre o pixel a substituir e o pixel que o precede na linha.
- Diferença entre o pixel a substituir e o pixel que o precede na coluna.
- Diferença entre o pixel a substituir e a média dos pixeis que o precede na respetiva linha e coluna.

Esta filtração é feita canal a canal e não pixel a pixel, ou seja, primeiro o canal vermelho irá ser filtrada, seguido dos outros.

Codificação: A codificação ocorre logo a seguir à filtração e consiste numa combinação de técnicas de codificação (LZ77 e codificação de Huffman, que se traduz no método Deflate).

I. Deflate:

Método de compressão**[14]** que utiliza LZ77 e Huffman Code. A partir deste algoritmo conseguimos bons racios de compressão. A maior desvantagem é que se tivermos um ficheiro com poucas repetições este algoritmos será pouco eficaz, pois irá se gastar quase tantos bits como o original.

Sendo que este método é normalmente mais apropriado para texto sendo que em imagem também funciona bem. É um algoritmo rápido em termos de compressão e decompressão mas traz um baixo rácio de compressão.

EXPLORAÇÃO DE MÉTODOS

Vamos analisar vários métodos baseados em algoritmos já referidos previamente. Estes métodos foram concebidos na linguagem de programação Python e vão ser testados num dataset de 4 imagens de formato .bmp com o objetivo de descobrir qual a taxa de compressão e a rapidez de compressão associados aos respetivos métodos.

Como tal a Figura 4 representa a comparação dos 4 ficheiros originais antes e após lhes ser aplicados os diversos métodos de compressão lossless:

	Original	RLE	Huffman Encoding	Deflate
Egg.bmp	17 329 KB	25 622 KB	12 437 KB	6 455 KB
Landscape.bmp	10 749 KB	24 058 KB	10 011 KB	4 341 KB
Zebra.bmp	16 346 KB	27 735 KB	11 982 KB	7 435 KB
Pattern.bmp	46 881 KB	7 911 KB	11 294 KB	2 464 KB

Figura 4 – Tamanho final do ficheiro comprimido usando vários métodos

TAXA DE COMPRESSÃO

A taxa de compressão consiste numa valor percentual que representa o quão comprimido um ficheiro ficou em comparação com o seu tamanho inicial. Uma das grandes utilidades desta taxa aplicada a este trabalho é a possibilidade de ajudar a aferir o quão eficaz um método é a comprimir um ficheiro .bmp e a partir desse valor estabelecer um ponto de comparação entre os vários métodos a ser analisados.

Para calcular esta taxa iremos recorrer à seguinte fórmula:

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

Na Figura 5 estão presentes os diversos valores para a taxa de compressão usando os valores dos tamanhos finais de cada ficheiro após a sua compressão que se encontram presentes na Figura 4 para proceder a esse cálculo.

	RLE	Huffman Encoding	Deflate
Egg.bmp	-47,9%	28,2%	62,8%
Landscape.bmp	-123,8%	6,9%	59,6%
Zebra.bmp	-69,7%	26,7%	54,5%
Pattern.bmp	83,1%	75,9%	94,7%

Figura 5 – Comparação das taxas de compressão

Ao analisar os resultados da Figura 5 concluimos que o método Deflate é o que apresenta uma melhor taxa de compressão, com um valor médio (obtido através da média da taxa de compressão das 4 imagens presentes no dataset original) de 67,9%, superando os 34,4% do Huffman Encoding e os -39,6% do RLE (Run Length Encoding).

Um dos principais motivos por detrás do destaque do Deflate é o facto de este método de compressão recorrer a uma combinação de dois métodos, o LZ_77 e o Huffman Encoding, o que resulta num aumento da taxa de compressão.

Também é possível que o RLE na maioria das imagens levou a um aumento do tamanho do ficheiro final, o que se pode justificar através do facto de que este método é muito mais bem sucedido na compressão de texto face à compressão de imagens e de que nas 3 imagens onde se verificou esse aumento é notória uma variação considerável da variação da intensidade da cor. Visto que este algoritmo se baseia em repetição de intensidades de sinais, o ficheiro final denota um tamanho superior.

Contudo na imagem pattern.bmp, verificou-se uma compressão considerável, com uma taxa de compressão de 83,1%, uma vez que esta imagem é a preto e branco (binária), apenas apresentando duas intensidades de sinal. Assim concluímos que este método é apenas recomendado na compressão de imagens binárias ou de texto.

RAPIDEZ DE COMPRESSÃO

A rapidez de compressão consiste no tempo que o ficheiro demora a ser comprimido. Este valor varia de método para método pelo que, quanto menor for a rapidez de compressão, mais eficiente é a sua aplicação na compressão de imagens sem perda de qualidade.

A Figura 6 estabelece uma comparação entre a rapidez de compressão para os métodos a ser estudados.

	RLE	Huffman Encoding	Deflate
Egg.bmp	27,34s	9,34s	1,10s
Landscape.bmp	23,61s	7,09s	0,89s
Zebra.bmp	32,60s	7,55s	0,68s
Pattern.bmp	7,30s	15,62s	0,55s

Figura 6 – Comparação da rapidez de compressão

Pela Figura 6 podemos constatar que o método do Deflate é o algoritmo mais rápido na compressão de imagens.

Logo, tanto em termos de taxa de compressão, como em rapidez de compressão, o Deflate provou ser superior aos outros métodos. Este algoritmo assenta na transformação da informação presente numa imagem não comprimida e aplica-lhe o método LZ_77 que devolve informação comprimida, sendo-lhe aplicada de seguida a codificação de Huffman, que gera duas árvores binárias. Em suma, a combinação destes dois algoritmos justifica os bons resultados aquando da compressão das imagens usadas como teste.

ENTROPIA NA COMPRESSÃO

Como teste, procedemos ao cálculo da entropia para cada uma das quatro imagens de forma a estabelecer uma correlação entre ganhos de compressão e entropia associada.

Egg.bmp	Landscape.bmp	Zebra.bmp	Pattern.bmp
5,72423	7,4205	5,8312	1,8292

Figura 7 – Valores de entropia para cada imagem

Na figura 7 temos os valores de entropia para cada imagem. Estes valores apresentam uma correlação direta bastante elevada com os algoritmos RLE e Huffman Encoding. Quanto maior é o valor da entropia (disperção estatística dos símbolos na fonte da imagem), menor vai ser a taxa de compressão visto que estes algoritmos dependem muito da repetição de poucos símbolos distintos.

O Deflate, apesar de também ser afetado por estes valores de entropia, está sujeito a menos flutuações, visto que resulta da combinação de dois algoritmos, resultando numa menor flutuação da taxa de compressão.

MÉTODO DE COMPRESSÃO PNG

Tal como já foi referido anteriormente, o PNG consiste de
Um método de compressão lossless altamente eficaz e mundialmente reconhecido. Este método funciona através de uma filtragem inicial dos pixeis de uma imagem, que consiste em substituir os pixeis por comparações com os seus vizinhos.

De seguida aplica o método Deflate à imagem já filtrada para finalizar a compressão.

Como tal a Figura 8 mostra a comparação dos métodos utilizados com os valores obtidos usando o PNG.

	Original	RLE	Huffman Encoding	Deflate	PNG
Egg.bmp	17 329 KB	25 622 KB	12 437 KB	6 455 KB	4 516 KB
Landscape.bmp	10 749 KB	24 058 KB	10 011 KB	4 341 KB	3 246 KB
Zebra.bmp	16 346 KB	27 735 KB	11 982 KB	7 435 KB	5 335 KB
Pattern.bmp	46 881 KB	7 911 KB	11 294 KB	2 464 KB	2 222 KB

Figura 8 – Comparação do tamanho final do ficheiro comprimido com o PNG

Agora que a Figura 8 já incorpora os valores dos 4 ficheiros originais comprimidos, podemos criar a tabela para comparar a taxa de compressão dos diferentes métodos face ao PNG.

	RLE	Huffman Encoding	Deflate	PNG
Egg.bmp	-47,9%	28,2%	62,8%	73,9%
Landscape.bmp	-123,8%	6,9%	59,6%	69,8%
Zebra.bmp	-69,7%	26,7%	54,5%	67,4%
Pattern.bmp	83,1%	75,9%	94,7%	95,3%

Figura 9 – Comparação das taxas de compressão com o PNG

Como é notório através da Figura 9, o PNG é o método de compressão que atinge melhores resultados. Com uma média de taxa de compressão (no dataset original de 4 imagens) de 76,5% supera qualquer outro método analisado, provando assim a sua eficácia. A comparação da média da taxa de comparação pode ser analisada através do gráfico da Figura 10.

Em comparação direta com o Deflate, o PNG recorre a um processo de filtração que reduz a variação da intensidade dos pixeis, o que leva a um aumento da redundância estatística.

A segunda etapa do PNG é a aplicação do Deflate (LZ_77 e Huffman Encoding) sob o ficheiro já filtrado. Ora o Deflate é uma parte do PNG, como tal não consegue igualar ou superar o seu sucesso em termos de taxa de compressão.

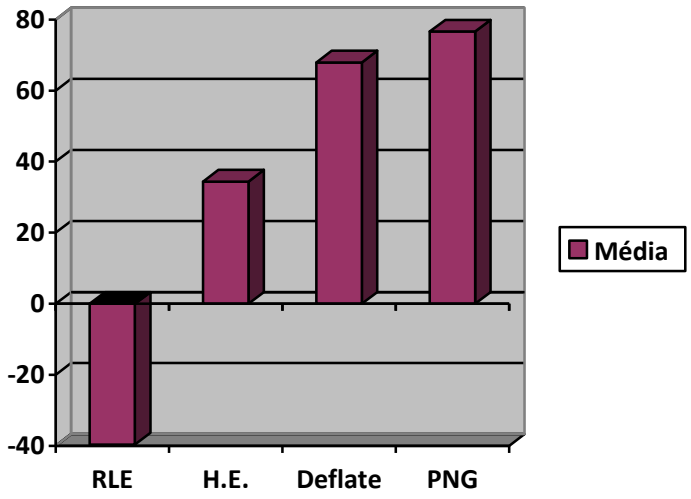

Figura 10 – Comparação da média das taxas de compressão

TRABALHO FUTURO SUGERIDO

Como trabalho futuro, seria interessante averiguar se uma adaptação do método PNG seria mais útil e obteria uma melhor taxa de compressão final. Como é sabido, o PNG recorre à filtragem, pelo que uma otimização deste processo poderia resultar em ganhos na imagem comprimida. Também seria interessante analisar a segunda parte do processo de compressão do PNG, o recurso ao Deflate. Será possível uma alteração deste algoritmo como, por exemplo, trocar o LZ_77 da primeira parte deste método pelo mais recente LZ_78? E se sim, qual seria o ganho associado a esta alteração.

CONCLUSÃO

Este relatório teve como objetivo o estudo e a análise de diversos métodos e algoritmos para a compressão de imagens sem perda de qualidade.

O estado de arte teve como foco explorar diversos tipos de codecs para uma compressão de imagem sem perda de qualidade, explicando os seus algoritmos e módulos e apontando as respetivas vantagens e desvantagens.

De seguida criámos e analisámos diversos algoritmos baseados nos métodos apontados no estado de arte. Foi alvo de estudo a sua capacidade de compressão num universo de 4 imagens e os resultados explicados, comparados e justificados.

Por fim, estabeleceu-se uma comparação estes os mesmos algoritmos e o método PNG, com o intuito de descobrir qual atinge melhor compressão e porquê.

REFERÊNCIAS:

Abaixo seguem-se as referências para a construção do relatório.

[1] https://www.fileformat.info/mirror/egff/ch09_03.htm
[2] https://pt.wikipedia.org/wiki/Codifica%C3%A7%C3%A3o_de_Huffman
[3] Powerpoints das aulas Teoricas, Cap 2 - Teoria da Informação e Codificação (slide 99)
[4] https://pt.wikipedia.org/wiki/LZ77
[5] https://pt.wikipedia.org/wiki/LZ78
[6] http://multimedia.ufp.pt/codecs/compressao-sem-perdas/codificacao-baseada-em-dicionarios/lz78/
[7] https://www.slideshare.net/Renju91/lzw-coding-technique-for-image-compressio
[8] https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch
[9] https://pt.wikipedia.org/wiki/C%C3%B3digos_de_Golomb
[10] https://ieeexplore.ieee.org/abstract/document/7583539
[11] https://en.wikipedia.org/wiki/Arithmetic_coding
[12] https://medium.com/@duhroach/how-png-works-f1174e3cc7b7
[13] https://en.wikipedia.org/wiki/Portable_Network_Graphics
[14] http://multimedia.ufp.pt/codecs/compressao-sem-perdas/codificacao-estatistica/codificador-qm/deflate/
[15] http://multimedia.ufp.pt/codecs/compressao-sem-perdas/codificacao-estatistica/algoritmo-de-huffman-adaptativo/
[16] http://multimedia.ufp.pt/codecs/compressao-sem-perdas/codificacao-estatistica/algoritmo-de-huffman/
[17] https://stackabuse.com/run-length-encoding/
[18] https://github.com/dcwatson/deflate
[19] https://github.com/soxofaan/dahuffman

Material de pesquisa fornecido pelo professor.

[17], [18] e [19] : Adaptações ao código foram feitas para garantir o bom funcionamento aquando da execução da compressão de imagens.