

Compressão de Imagens para Ensino

Guilherme Hiago Santos




¹Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Caixa Postal 1429 – 90619-900 – Porto Alegre – RS – Brasil

{guilherme.hiago}@edu.pucrs.br

Resumo. *Este artigo tem por finalidade apresentar o desenvolvimento de uma aplicação educativa sobre a compressão de imagens, para isto são apresentadas as técnicas de compressão escolhidas, Run-Length e Color Cell Compression, assim como seus motivos de escolha, seguidas das suas implementações e da interface interativa.*

1. Introdução

Métodos de compressão de arquivos já são utilizados e pesquisados diariamente com o intuito de diminuir a quantidade de dados necessários para apresentação de dados.

A compressão de imagem é um alvo evidente  de estudo, pois a sua utilização tem crescido, tendo em vista que, como dito no artigo [1], todas as imagens obtidas na internet são comprimidas. Tendo em vista as inúmeras vantagens da compressão de imagem, para cada caso específico, como a velocidade de navegação na internet, fica evidente a importância da compressão de imagens e suas diferentes vertentes.

Neste projeto será apresentado as escolhas feitas para os algoritmos que estarão em uma aplicação com viés educativo, para o ensino de técnicas de compressão de imagens.

As próximas seções estarão divididas da seguinte maneira: seção 2, relata o problema abordado, os algoritmos escolhidos e o motivo da escolha. Seção 3, detalhamento das técnicas escolhidas e dos passos de procedimento de cada um. Na seção 4, é abordado como cada algoritmo foi implementado, levantando sua relação com o funcionamento teórico, além da implementação gráfica. Já na seção 5 é feita a conclusão do trabalho, realizando um fechamento das ideias apresentadas.

2. Sobre o Problema Computacional

A necessidade do ensino didático sobre compressão de imagens é cada vez mais evidente. O processo de compressão de arquivos já é usado a muito tempo para a redução de ocupação de memória e redução de redundância, com imagens o caso não é diferente. O aumento da resolução de imagens tiradas por câmeras mais recentes, traz, em conjunto, uma maior ocupação de memória de aparelhos digitais, assim como ocupação de banda de internet, o que acarrete muitas consequências, como o aumento do tempo de processamento de sites e aplicações web. Contudo, como a velocidade de conexão importa, muitas vezes opta-se por comprimir estas imagens, aumentando a demanda por compressões de imagens. Porém ao tentar comprimir seu arquivo o indivíduo depara-se com inúmeras formas de compressão e acaba não sabendo qual a melhor para sua situação.

Como o ensino sobre compressão de imagens é necessário, visa-se encontrar algoritmos diferentes que abrangem situações diferentes de forma evidente, cada um tendo

suas vantagens e desvantagens. Para tal foram escolhidos os algoritmos RLE e Color Cell Compression, pois ambos são de duas categorias diferentes, compressão sem perdas e compressão com perdas, respectivamente, o que significa que com um dos arquivos comprimidos é possível chegar em uma imagem idêntica à original e com a outra apenas em uma aproximação. Sendo assim, ambos algoritmos devem ser implementados em uma interface gráfica que facilite a interação dos usuários com os métodos de compressão.

3. Algoritmos de Compressão Seleccionados

Para um melhor entendimento dos tipos de compressão de imagem, foi considerada uma melhor escolha utilizar um algoritmo de compressão sem perdas e outro algoritmo com perdas. Os algoritmos escolhidos foram o RLE (Run-Length) e o CCC (Color Cell Compression).

3.1. Run-Length

O RLE trata-se de um algoritmo de compressão sem perdas que

1. Determinar uma FLAG que não exista no texto a comprimir.
2. Ler um caractere Ler os próximos caracteres enquanto forem iguais ao primeiro caractere lido.
3. Se o número total de caracteres lidos for igual ou superior a 4, comprimir essa cadeia de caracteres da seguinte forma: FLAG + n° de repetições + caractere.
4. Se o número total de caracteres lidos for inferior a 4, não se efetua compressão, logo essa cadeia de caracteres permanece inalterável.

A seguir consta pseudocódigo do algoritmo Run-Length semelhante ao apresentado no artigo [2].

Figure 1. Pseudocódigo RLE

```
Início
Sendo A = altura da imagem
Sendo L = largura da imagem
Sendo IMG = uma matriz[A][L], em que cada posição
é um vetor de 3 posições representando as cores do pixel.
Sendo V = [(IMG[L][A], 0)], pois é o primeiro pixel armazenado
Para i de 0 até A-1 faça
    Para j de 0 até L-1 faça
        Se IMG[i][j] == V[Tamanho(V)[0]]
            V[Tamanho(V)][1] += 1
        Se não
            V = V + (IMG[i][j], 1)
Fim
```

3.2. Color Cell Compression

Este é um algoritmo de compressão com perdas, que possui 7 etapas, (na realidade são 8 etapas, mas como a sexta é opcional, não será aplicada). Estas etapas foram extraídas do artigo [3] e do livro [4], essas etapas são:

1. Para cada pixel da imagem calcular o valor de luminescência Y que se dá por Equação 1.

$$Y(i, j) = 0.30 * R(i, j) + 0.59 * G(i, j) + 0.11 * B(i, j). \quad (1)$$

2. A imagem agora está subdividida em blocos de 4 pixels por 4 pixels, representada na Tabela 1, e, a média aritmética da luminância de cada pixel no bloco, representada na Tabela 2, é usado para selecionar um valor de luminância representativo.

Table 1. Celula 4x4 de Cores

(182,121,242)	(142,129,138)	(230,9,162)	(190,17,58)
(22,153,82)	(238,161,234)	(70,41,2)	(30,49,154)
(118,185,178)	(78,193,74)	(166,73,98)	(126,81,250)
(214,217,18)	(174,225,170)	(6,105,194)	(222,113,90)

Fonte: Pins M, 1991.

Table 2. Luminescência dos Valores da Tabela1

152	133	92	73
105	192	45	54
164	145	103	113
194	203	85	143

Fonte: Pins M, 1991.

3. Cada pixel é dividido em dois grupos, os que possuem luminescência maior que a média e os que possuem valor menor ou igual a média. Se um pixel pertence a um grupo é representado por um valor "0" ou "1" em um bitmap, separado, de 16 entradas, representado na Tabela 3.

Table 3. Bitmap Exemplo da Tabela 2

1	1	0	0
0	1	0	0
1	1	0	0
1	1	0	1

Fonte: Pins M, 1991.

4. Duas cores representativas de 24 bits são selecionadas, uma para cada grupo, sendo a primeira cor a média aritmética do grupo "1" do bitmap e a segunda, sendo a média aritmética das cores do grupo "0".

5. Armazena-se o bitmap de luminescência anexado as duas cores representativas. Neste estágio, a imagem foi compactada em um bitmap de 16 entradas com dois valores binários de 24 bits anexados. O tamanho total do bloco comprimido é agora de 16 bits para o bitmap de luminância e duas quantidades binárias de 24 bits para cada cor representativa, resultando em um tamanho total de 64 bits, que, quando dividido por 16 (o número de pixel no bloco), resulta em 4, ou seja, 4 bits por pixel.
6. Cria-se um histograma de todas as cores de 24 bits presentes na imagem e escolhe-se as 256 cores mais frequentes na imagem.
7. Por fim verifica-se quais cores estão mais próximas das duas cores representativas da célula atual e substitui-se essas duas cores representativas pelos dois índices de 8 bits do histograma. Com isso chega-se em um resultado de compressão de $16 + 8 + 8 = 32$ bits, que quando dividido por 16 resulta em 2 bits por pixel.

4. Implementação

A implementação dos algoritmos RLE e CCC serão feitas em Python. Esta implementação utiliza duas bibliotecas adicionais à linguagem, Pillow, para lidar com a leitura e construção de imagens, e Numpy, para lidar com os vetores gerados. Essa escolha de linguagem foi feita pois Python facilita o processo de prototipagem, assim como possui diversos pacotes adicionais, como para lidar com imagens ou outros tipos de dados, nela há também estruturas que facilitam, **como listas e dicionários, que funcionam semelhante a um hashmap**. Nas duas etapas seguintes será comentado sobre a implementação de cada um dos dois algoritmos.

4.1. Implementação do Algoritmo RLE

O algoritmo foi implementado em uma classe chamada RLE, que possui 2 métodos auxiliares, "mostraDescomprimido" e tamanhoArquivo", que estão relacionado com a interface gráfica. Mas como o foco será a codificação, a decodificação será omitida do artigo por se tratar apenas da leitura do arquivo compactado.

```
1 def comprime(self):
2     # Converte imagem em matriz
3     data = np.array(self.im) # "data" is a height x width x 4 numpy array
4     red, green, blue = data.T # Temporarily unpack the bands for readability
5
6     # dimensoes da imagem
7     img_width = len(red)
8     img_height = len(red[0])
9
10    size = [str(img_width), str(img_height)]
11
12    # vetor com as cores salvas
13    cores = [[red[0][0], green[0][0], blue[0][0]]]
14    # vetor com as flags
15    flags = [0]
16
17    # loop que percorre as listas
18    for i in range(img_height):
```

```

19         for j in range(img_width):
20             cor_atual = [red[i][j], green[i][j], blue[i][j]]
21
22             # caso a cor seja igual anterior
23             # aumenta o valor da flag
24             if cor_atual == cores[len(cores)-1]:
25                 flags[len(flags)-1] += 1
26             # caso contrario adiciona nova cor
27             else:
28                 cores.append(cor_atual)
29                 flags.append(1)
30
31         saida = open(self.nome_arq + "_rle_compress.txt", "w")
32         saida.write(size[0] + " ")
33         saida.write(size[1] + " ")
34
35         for i in range(len(cores)):
36             saida.write(str(flags[i]) + " " + str(cores[i][0]) + " " + str(
37                 cores[i][1]) + " " + str(cores[i][2]) + " ")
38
39         # fecha arquivo texto
40         saida.close()
41         # print(self.tamanhoArquivo(self.nome_arq + "_rle_compress.txt"), "
42         # kB")
43
44         return "Tamanho do Arquivo Compresso: " + str(self.tamanhoArquivo(
45             self.nome_arq + "_rle_compress.txt")).replace(".", ",") + "kB"

```

Listing 1. Compressão RLE com Python

O método de compressão implementa quase diretamente o algoritmo da Figura 1, sendo as únicas alterações, a divisão das cores e flags em duas listas diferentes para facilitar a manipulação e as interações com arquivos externos como a escrita no arquivo de compressão. Este método retorna, também, uma string com o tamanho do arquivo arquivo gerado e o nome dele.

4.2. Implementação do Algoritmo Color Cell Compression

A implementação foi realizada a partir de uma classe "ColorCellCompression", que possui métodos auxiliares para o cálculo da Equação 1 (o chamado "calcula_luminescencia"), o "cria_tabela_cores" que realiza a etapa 6 dos passos citados em 3.2. Como o foco do projeto é a compressão será dado foco para o método "comprime".

```

1 def comprime(self):
2     cores_uso = self.cria_tabela_cores()
3
4     data = np.array(self.im) # "data" is a height x width x 4 numpy
5     array
6     red, green, blue, alpha = data.T # Temporarily unpack the bands for
7     readability
8
9     img_size = len(red)
10
11     # posicao na largura e altura
12     pos, pos2 = 0, 0
13
14     while pos < img_size and pos2 < img_size:

```

```

13     lumi, bitmap = [], []
14
15     for i in range(self.tam_celula):
16         lumi.append([])
17
18     for i in range(self.tam_celula):
19         bitmap.append([])
20
21     cor1, cor2 = [0, 0, 0], [0, 0, 0]
22
23     # Calcula luminescencia de todos na celula
24     for i in range(pos, pos+self.tam_celula):
25         for j in range(pos2, pos2+self.tam_celula):
26             lumi[i-pos].append(self.calcula_luminescencia(red[i][j]
27 ], green[i][j], blue[i][j]))
28
29     # soma todos elementos de lumi e faz a media
30     lumi_media = sum([sum(x) for x in lumi]) / (len(lumi) * len(
31 lumi[0]))
32
33     # nas posi oes que lumi[i][j] > lumi_media, bitmap[i][j] = 1,
34 se nao, bitmap[i][j] = 0
35     bitmap = [[ 1 if lumi[i][j] > lumi_media else 0 for j in range(
36 len(lumi[0]))] for i in range(len(lumi))]
37
38     # INICIO calculo da media para cor 1 e cor 2
39     soma1 = [0, 0, 0]
40     soma2 = [0, 0, 0]
41
42     n_soma1, n_soma2 = 0, 0
43
44     # Soma cores da celula nos grupos 0 ou 1 do Bitmap
45     for i in range(pos, pos+self.tam_celula):
46         for j in range(pos2, pos2+self.tam_celula):
47             if(bitmap[i-pos][j-pos2] == 1):
48                 soma1[0] += red[i][j]
49                 soma1[1] += green[i][j]
50                 soma1[2] += blue[i][j]
51                 n_soma1 += 1
52             else:
53                 soma2[0] += red[i][j]
54                 soma2[1] += green[i][j]
55                 soma2[2] += blue[i][j]
56                 n_soma2 += 1
57
58     if n_soma1 == 0:
59         cor1 = [red[pos][pos2], green[pos][pos2], blue[pos][pos2]]
60     else:
61         cor1 = [x/n_soma1 for x in soma1]
62
63     if n_soma2 == 0:
64         cor2 = [red[pos][pos2], green[pos][pos2], blue[pos][pos2]]
65     else:
66         cor2 = [x/n_soma2 for x in soma2]
67
68     # FIM calculo da media para cor 1 e cor 2

```

```

65     # troca cor 1 por uma das 256, a mais proxima
66     indice1 = self.encontra_cor_prox(cor1[0], cor1[1], cor1[2],
cores_uso)
67     cor1 = cores_uso[indice1]
68     # troca cor 2 por uma das 256, a mais proxima
69     indice2 = self.encontra_cor_prox(cor2[0], cor2[1], cor2[2],
cores_uso)
70     cor2 = cores_uso[indice2]
71
72     self.texto += " " + str(indice1) + " " + str(indice2)
73
74     # Para cada pixel da celula define RGB
75     for i in range(pos, pos+self.tam_celula):
76         for j in range(pos2, pos2+self.tam_celula):
77             if(bitmap[i-pos][j-pos2] == 1):
78                 self.texto += " 1"
79             else:
80                 self.texto += " 0"
81
82     # Avan a pos na largura
83     pos += self.tam_celula
84     # Avan a pos na altura
85     if pos >= img_size:
86         pos = 0
87         pos2 += self.tam_celula
88
89     arq = open(self.nomeArq + "_ccc_compress.txt", "w")
90     arq.write(self.texto)
91     arq.close()

```

Listing 2. Compressão CCC com Python




Analisando-se a implementação nota-se que ela segue os passos do algoritmo que são explicadas na parte 3.2 deste artigo, porém o passo 6 é realizado primeiro, pois a criação da tabela de cores será realizada uma vez e utilizada apenas no último passo.

Desconsiderando a realocação do sexto passo, os demais são seguidos a risca, sendo que o algoritmo percorre a imagem em células de 4x4 pixels, calculando a luminescência dos pixels a partir dessa luminosidade separa os pixels em grupos, calcula a cor média do grupo e substitui pela cor mais semelhante dentro da tabela criada. Seguido a isto, ao fim de cada iteração será adicionado os dois índices das cores representativas e dos bits da célula, ao texto do arquivo comprimido, que já possuía as dimensões originais da imagem, assim como o número de cores da tabela, o tamanho da célula e todas as cores da tabela em si.

5. Conclusão

Neste trabalho foi apresentado a implementação de dois algoritmos de compressão de imagens em uma aplicação que visa o ensino, foi levado em consideração até mesmo diferentes tipos de algoritmos, como os com perdas de dados e os sem perdas. Foi explicado de forma clara o passo a passo de como cada algoritmo funciona e como trabalha, além de que com a execução da aplicação completa trará de forma eficiente ao usuário as noções de diferença entre os tipos de compressão de imagens e qual o momento mais propício para cada uma.

References

- [1] G. E. Blelloch, "Introduction to data compression," in <http://www.cs.cmu.edu/guyb/-realworld/compression.pdf>, 2013. 
- [2] A. H. Hussein, S. S. Mahmud, and R. J. Mohammed, "Image compression using proposed enhanced run length encoding algorithm," (Baghdad), pp. 1–14, 2011. 
- [3] G. Campbell, T. A. DeFanti, J. Frederiksen, S. A. Joyce, L. A. Leske, J. A. Lindberg, and D. J. Sandin, "Two bit/pixel full color encoding," pp. 215–223, 1986. 
- [4] M. Pins, *Extensions of the Color-Cell-Compression-Algorithm*. In: *Thalmann N.M. Thalmann D. (eds) Computer Animation '91*. Springer, Tokyo. 1991. 