

## Compressão de Entropia

Na última aula prática vimos um exemplo simples de um padrão simples de representação de imagens, o chamado padrão CUIF Versão 1 (CUIF.1). Para realizar esta aula prática é necessário ter concluído a Prática III. Caso o seu grupo não concluiu, é necessário fazê-lo antes de realizar esta prática. Esta aula prática exigirá o cálculo do PSNR implementado na Prática III.

Nesta aula prática, veremos ao todo 3 versões do padrão CUIF:

- CUIF.1. Representa a imagem em um formato RGB;
- CUIF.2. Representa a imagem em um formato YCbCr.
- CUIF.3. Representa a imagem em um formato YCbCr e compactada usando usando codificação RLE.

O projeto Python da Prática IV contém 3 arquivos:

- praticaIV.py: contendo métodos para cálculo do PSNR, e uso da classe Cuif para conversão de imagens BMP em CUIF e vice-versa.
- Cuif.py: classe implementando métodos para converter imagens BMP em CUIF e vice-versa
- RLE.py: contém funções para compactar e descompactar dados usando a codificação RLE.

Analise estes arquivos, em especial o arquivo praticaIV.py.

### 1 CUIF.1

Como visto na aula prática anterior, o padrão CUI.1 é uma representação RGB separada em canais, de maneira similar ao BMP. Porém, diferente do BMP onde cada *pixel* aparece com seus canais BGR, o CUI.1 apresenta cada canal R, G e B completos em sequência. Ou seja, ao invés de codificar *pixel-a-pixel*, codifica-se canal-a-canal. Cada *pixel* utilizará 1 byte em cada canal.

**Exemplo de CUI.1, representado em um arquivo CUIF**

Vamos supor uma imagem com  $2 \times 2$  *pixels*:

red = (FF 00 00)<sub>16</sub>            green = (00 FF 00)<sub>16</sub>  
 blue = (00 00 FF)<sub>16</sub>            gray = (B7 B7 B7)<sub>16</sub>

Para este exemplo, há apenas um estudante no grupo, cuja matrícula é 99132042. Vejamos como fica a organização de um arquivo CUIF para armazenar essa imagem seguindo o padrão CUI.1:

byte	valor				significado
	3	2	1	0	
0	–	–	–	–	assinatura CUIF
4	–	–	–	1	versão do padrão CUI (CUI.1)
5	–	–	–	1	número de estudantes no grupo
6	2 <sub>10</sub>				largura
10	2 <sub>10</sub>				altura
14	99132042 <sub>10</sub>				matrícula do aluno no grupo
18	–	–	–	FF <sub>16</sub>	R pixel 0,0
19	–	–	–	00 <sub>16</sub>	R pixel 0,1
20	–	–	–	00 <sub>16</sub>	R pixel 1,0
21	–	–	–	B7 <sub>16</sub>	R pixel 1,1
22	–	–	–	00 <sub>16</sub>	G pixel 0,0
23	–	–	–	FF <sub>16</sub>	G pixel 0,1
24	–	–	–	00 <sub>16</sub>	G pixel 1,0
25	–	–	–	B7 <sub>16</sub>	G pixel 1,1
26	–	–	–	00 <sub>16</sub>	B pixel 0,0
27	–	–	–	00 <sub>16</sub>	B pixel 0,1
28	–	–	–	FF <sub>16</sub>	B pixel 1,0
29	–	–	–	B7 <sub>16</sub>	B pixel 1,1

## 1.1 Roteiro com a codificação CUIF.1

Faça os seguintes procedimentos iniciais:

1. Abra o arquivo praticaIV.py, e insira seu código de cálculo do PSNR. Se não for necessário na sua implementação, remova a função MSE().
2. Também no arquivo praticaIV.py modifique a lista `matricula` para conter o número de matrículas dos alunos de seu grupo.
3. Execute o praticaIV.py para geração dos arquivos CUIF na versão 1 e a conversão do arquivo CUIF em BMP. Também será impresso o PSNR entre a imagem BMP original e a imagem BMP obtida pela decodificação do arquivo CUIF.1. O valor do PSNR deve ser infinito. Se não for, verifique sua implementação.
4. Alterando o código entregue, codifique a imagem teste.bmp usando CUIF.1 e converta este arquivo CUIF em BMP (gerando o teste1.bmp). Também gere o PSNR deste arquivo bmp decodificado a partir do CUIF.1.

## 2 CUIF.2

Nosso padrão CUIF.2 é bastante similar ao CUIF.1. Porém, ao invés de representar os *pixels* em RGB, a representação será em YCbCr, de acordo com o formato JPEG File Interchange Format. O documento descrevendo as conversões de RGB para YCbCr e vice-versa está disponível em <https://www.w3.org/Graphics/JPEG/jfif3.pdf>.

Segue na sequência o formato do arquivo CUIF.2

**Exemplo de CUI.1, representado em um arquivo CUIF**

Vamos supor uma imagem com  $2 \times 2$  *pixels*. Para este exemplo, há apenas um estudante no grupo, cuja matrícula é 99132042. Vejamos como fica a organização de um arquivo CUIF para armazenar essa imagem seguindo o padrão CUI.1:

byte	valor				significado
	3	2	1	0	
0	–	–	–	–	assinatura CUIF
4	–	–	–	1	versão do padrão CUI (CUI.1)
5	–	–	–	1	número de estudantes no grupo
6	$2_{10}$				largura
10	$2_{10}$				altura
14	$99132042_{10}$				matrícula do aluno no grupo
16	–	–	–	$FF_{16}$	Y pixel 0,0
17	–	–	–	$00_{16}$	Y pixel 0,1
18	–	–	–	$00_{16}$	Y pixel 1,0
19	–	–	–	$B7_{16}$	Y pixel 1,1
20	–	–	–	$00_{16}$	Cb pixel 0,0
21	–	–	–	$FF_{16}$	Cb pixel 0,1
22	–	–	–	$00_{16}$	Cb pixel 1,0
23	–	–	–	$B7_{16}$	Cb pixel 1,1
24	–	–	–	$00_{16}$	Cr pixel 0,0
25	–	–	–	$00_{16}$	Cr pixel 0,1
26	–	–	–	$FF_{16}$	Cr pixel 1,0
27	–	–	–	$B7_{16}$	Cr pixel 1,1

## 2.1 Roteiro com a codificação CUIF.2

Faça os seguintes procedimentos com o CUIF.2:

1. Baixar <https://www.w3.org/Graphics/JPEG/jfif3.pdf>, e leia a seção *Conversion to and from RGB*, em seguida analise a implementação desta conversão no arquivo ColorSpace.py.
2. Usando o código entregue, codificar as imagens lena.bmp e teste.bmp no formato CUIF.2, nomeando os arquivos para lena2.cuif teste2.cuif. Em seguida, converta estes arquivos Cuif em bmp, nomeando estes arquivos em lena2.bmp e teste2.bmp. Também faça o código necessário para cálculo do PSNR do CUIF.2 para estas duas imagens.

## 3 Cuif.3

Na implementação de RLE disponibilizada (no arquivo RLE.py), a compressão se baseia na supressão de repetição de símbolos de tamanho de 1 byte (8bits). Nesta implementação, o bit mais significativo de cada byte foi utilizado como flag para indicar se o byte representa um símbolo único (sem repetição) ou contém o número de repetições do próximo byte. Assim, caso o bit mais significativo for 0, o byte representa um símbolo único. Caso este bit seja 1, o restante dos bits indica o número de repetição, e o próximo byte é o símbolo que se repete.

Para melhor compreender como funciona o RLE em nossa implementação, vamos analisar a mesma mensagem que abordamos anteriormente:  $m = \{3, 3, 2, 5, 1, 1, 1, 2, 3, 5, 1, 4, 3, 3, 3, 3\}$ . Esta, seria codificada em binário (1 byte por símbolo) como:

Já na codificação RLE proposta na implementação, a representação seria da forma a seguir:

Esta implementação suporta apenas entradas de até 127 símbolos, por utilizar 7 bits para codificar os símbolos (primeiro bit é usado como flag de repetição). Como as imagens consideradas representam valores de Y, Cb e CR com 1 byte (valores de 0 a 255), o valor do byte é dividido por 2 na codificação e multiplicado por 2 na decodificação. Na implementação, foi utilizado o deslocamento de um bit a direita na codificação e deslocamento

Tabela 1: Codificação binária de  $m$  considerando 8 bits/símbolo.

símbolo	binário
3	00000011
3	00000011
2	00000010
5	00000101
1	00000001
1	00000001
1	00000001
2	00000010
3	00000011
5	00000101
1	00000001
4	00000100
3	00000011
3	00000011
3	00000011
3	00000011

Tabela 2: Codificação RLE conforme implementada para CUI.4

binário	significado
10000010	o próximo byte é repetido $2\times$ (3, 3)
00000011	byte que é repetido: 3
00000001	este byte é único, representa 2
00000010	este byte é único, representa o 4 (deveria ser 5)
10000011	o próximo byte é repetido $3\times$ (1, 1, 1)
00000001	este byte é repetido: 1 (não há erro)
00000001	este byte é único, representa o 2
00000001	este byte é único, representa o 2 (deveria ser 3)
00000010	este byte é único, representa o 4 (deveria ser 5)
00000000	este byte é único, representa o 0 (deveria ser 1)
00000010	este byte é único, representa o 4
10000100	o próximo byte é repetido $4\times$ (3, 3, 3, 3)
00000011	byte que é repetido: 3

de um bit a esquerda na decodificação. Assim, sempre que o byte representando um símbolo único for ímpar, haverá erro no bit menos significativo, que será zerado.

A mensagem decodificada seria:  $m' = \{3, 3, 2, 4, 1, 1, 1, 2, 2, 4, 0, 4, 3, 3, 3, 3\}$ . Note que há erros! Usando a Equação do MSE, apresentado na Prática III, podemos estimar a MSE entre  $m$  e  $m'$ :

$$MSE(m, m') = \frac{1}{16} \times (1^2 + 1^2 + 1^2 + 1^2) = \frac{1}{16} \times 4 = \frac{1}{4}$$

Através da MSE, podemos obter a PSNR:

$$PSNR(m, m') = 10 \times \log_{10} \left( \frac{(2^8 - 1)^2}{MSE(m, m')} \right) = 10 \times \log_{10} \left( \frac{65025}{1/4} \right) = 10 \times \log_{10}(260100) = 54,1514dB$$

Vimos então que, mesmo a RLE sendo um método de codificação de entropia que por definição não gera erros, por conta de decisões de implementação foram inseridos erros. Durante a execução do roteiro a seguir, busquem analisar as imagens codificadas e verifiquem se é possível identificar esses erros.

### 3.1 Roteiro com a codificação CUIF.3

Faça os seguintes procedimentos com o CUIF.3:

1. Usando o código entregue, codificar as imagens lena.bmp e teste.bmp no formato CUIF.3, nomeando os arquivos para lena3.cuif teste3.cuif. Em seguida, converta estes arquivos Cuif em bmp, nomeando estes arquivos em lena3.bmp e teste3.bmp. Também faça o código necessário para cálculo do PSNR do CUIF.3 para estas duas imagens.

## 4 Relatório a ser entregue

**Questão 1.** Indique as taxas de compressão obtida pelas codificações CUIF.1, CUIF.2 e CUIF.3 para as imagens lena e teste (razão entre o arquivo original bmp e os arquivos cuif).

**Questão 2.** Compare as taxas obtidas pelas codificações CUIF.1, CUIF.2 e CUIF.3 para a imagem lena. Qual é a codificação gerou a maior taxa de compressão? Justifique porque.

**Questão 3.** Compare as taxas obtidas para as imagens lena e teste. Explique porque a taxa de compressão para o CUIF.3 para a imagem teste.

**Questão 4.** Indique o PSNR medido nas imagens lena1.bmp, lena2.bmp e lena3.bmp quando comparados com a imagem original lena.bmp. Justifique os valores obtidos, explicando a fonte dos ruídos gerados em cada codificação.