

## Compressão de Entropia

### 1 Padrão CUIF

Na última aula vimos um exemplo simples de um padrão simples de representação de imagens, o chamado padrão CUIF Versão 1 (CUIF.1). E vimos duas ferramentas de conversão. Caso necessário, você pode consultar a prática III para verificar o formato.

Segue abaixo o formato do cabeçalho CUIF, como já visto na aula prática anterior.

Offset	Tamanho	Descrição
0	2	assinatura (identificador), deve ser 5431 <sub>10</sub>
2	2	versão do padrão CUI
3	1	número de estudantes no grupo (NUMBER_OF_STUDENTS)
4	4	largura da imagem (em pixels)
8	4	altura da imagem (em pixels)

Após o cabeçalho, há uma lista de identificadores dos alunos no grupo. Cada identificador (ID) ocupará 4 bytes. Para esta disciplina, será utilizado o número da matrícula de cada aluno como ID. Note que o número de IDs no arquivo deve estar definido corretamente no cabeçalho (NUMBER\_OF\_STUDENTS).

Após 12 bytes do cabeçalho + 4 × NUMBER\_OF\_STUDENTS bytes, estarão os dados da imagem.

### 2 Baixando projeto para a aula prática

Nesta aula prática, veremos mais 3 versões do padrão CUIF. Então ao todo teremos 4 versões:

- CUIF.1. Representa a imagem, sem compressão, em um formato RGB;
- CUIF.2. Representa a imagem, sem compressão, em um formato YCbCr.
- CUIF.3. Representa a imagem em um formato YCbCr, com compressão usando codificação de Huffman.
- CUIF.4. Representa a imagem em um formato RGB, com compressão usando codificação RLE.

Para esta aula prática, baixe o projeto Java no Moodle. Este projeto contém 3 pacotes:

- huffmancoding: que implementa a codificação de Huffman. Não há necessidade de se estudar este pacote, bastando entender apenas os métodos `br.ufsc.ine5431.huffmancoding.HuffamnCoding.compression` e `br.ufsc.ine5431.huffmancoding.HuffamnCoding.decompression` que fazem a compactação e descompactação usando a codificação de Huffman;
- rlecoding: que contém a classe `RLECoding`, com seus métodos para compactação e descompactação usando a codificação RLE;
- praticaiv: pacote contendo as classes usadas nas aulas práticas.

Lembrando que usamos duas ferramentas para conversão:

1. `bmp2cuif`: para conversão de BMP para CUIF. A sintaxe deste comando é `java bmp2cuif -version 1 input.bmp output.cuif`, onde a versão pode variar de 1 a 4;
2. `cuif2bmp`: para conversão de CUIF para BMP. A sintaxe deste comando é `java cuif2bmp input.cuif output.bmp` ;

### 3 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 *raster*. 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	–	–	5431 <sub>10</sub>		assinatura CUIF
2	–	–	–	1	versão do padrão CUI (CUI.1)
3	–	–	–	1	número de estudantes no grupo
4	2 <sub>10</sub>				largura
8	2 <sub>10</sub>				altura
12	99132042 <sub>10</sub>				matrícula do aluno no grupo
16	–	–	–	FF <sub>16</sub>	R pixel 0,0
17	–	–	–	00 <sub>16</sub>	R pixel 0,1
18	–	–	–	00 <sub>16</sub>	R pixel 1,0
19	–	–	–	B7 <sub>16</sub>	R pixel 1,1
20	–	–	–	00 <sub>16</sub>	G pixel 0,0
21	–	–	–	FF <sub>16</sub>	G pixel 0,1
22	–	–	–	00 <sub>16</sub>	G pixel 1,0
23	–	–	–	B7 <sub>16</sub>	G pixel 1,1
24	–	–	–	00 <sub>16</sub>	B pixel 0,0
25	–	–	–	00 <sub>16</sub>	B pixel 0,1
26	–	–	–	FF <sub>16</sub>	B pixel 1,0
27	–	–	–	B7 <sub>16</sub>	B pixel 1,1

### 4 Roteiro da Parte I

Faça os seguintes procedimentos iniciais:

1. Baixar o projeto CUI no Moodle.
2. Modificar o valor da variável `numero_de_estudantes` (linha 47 do arquivo `bmp2cuif.java`) para o número de estudantes no grupo;
3. Atualizar o array (`id_estudantes`, linha 48, `bmp2cuif.java`) com seus números de matrícula dos alunos do grupo de alunos;
4. Gerar os arquivos `JazzMan2.cuif` e `JazzMan2.bmp` usando:

```
java bmp2cuif -version 1 JazzMan.bmp JazzMan1.cuif
java cuif2bmp JazzMan1.cuif JazzMan1.bmp
```
5. Corrigir o código projeto caso necessário (Caso não tenha feito, volte para a prática III).

## 5 CUIF.2

Nosso padrão CUI.2 é bastante similar ao CUI.1. Porém, ao invés de representar os *pixels* em RGB, a representação será em YCbCr, de acordo com a recomendação ITU-R BT.601. O documento descrevendo tal recomendação está disponível em [https://www.itu.int/dms\\_pubrec/itu-r/rec/bt/R-REC-BT.601-7-201103-I!!PDF-E.pdf](https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.601-7-201103-I!!PDF-E.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	–	–	5431 <sub>10</sub>		assinatura CUIF
2	–	–	–	2	versão do padrão CUI (CUI.1)
3	–	–	–	1	número de estudantes no grupo
4	2 <sub>10</sub>				largura
8	2 <sub>10</sub>				altura
12	99132042 <sub>10</sub>				matrícula do aluno no grupo
16	–	–	–	FF <sub>16</sub>	Y pixel 0,0
17	–	–	–	00 <sub>16</sub>	Y pixel 0,1
18	–	–	–	00 <sub>16</sub>	Y pixel 1,0
19	–	–	–	B7 <sub>16</sub>	Y pixel 1,1
20	–	–	–	00 <sub>16</sub>	Cb pixel 0,0
21	–	–	–	FF <sub>16</sub>	Cb pixel 0,1
22	–	–	–	00 <sub>16</sub>	Cb pixel 1,0
23	–	–	–	B7 <sub>16</sub>	Cb pixel 1,1
24	–	–	–	00 <sub>16</sub>	Cr pixel 0,0
25	–	–	–	00 <sub>16</sub>	Cr pixel 0,1
26	–	–	–	FF <sub>16</sub>	Cr pixel 1,0
27	–	–	–	B7 <sub>16</sub>	Cr pixel 1,1

## 6 Roteiro da Parte II

Faça os seguintes procedimentos com o CUIF.2:

1. Baixar a recomendação BT.601, consultem<sup>1</sup> as Seções 2.5.1 e 2.5.2 e implementem a conversão RGB → YCbCr no método `rgb_to_ycbcr_base_double` do arquivo `ColorSpace.java` (linha 15);
2. Gerar um arquivo chamado `JazzMan2.cuif` usando:

```
java bmp2cuif -version 2 JazzMan.bmp JazzMan2.cuif
```
3. Gerar um arquivo chamado `JazzMan2.bmp` usando o comando a seguir e abra o arquivo bmp. Caso haja algum problema na imagem, é necessário verificar a conversão RGB para YCbCr implementada.

```
java cuif2bmp JazzMan2.cuif JazzMan2.bmp
```

## 7 Calculando Ruído

Na última aula prática vimos que mesmo uma simples conversão entre espaços de cor pode gerar erros na codificação. Tal erro surgiu na codificação da mensagem original (Ori), que será enviada por um canal ou

<sup>1</sup>Vejam também, no material da disciplina, os slides 73 e 84 do Cap. 2 e o slide 87 do Cap. 3.

armazenada e posteriormente decodificada. Como houve erro, a mensagem decodificada (Dec) será diferente de Ori. Tomemos como exemplo o seguinte esquema de troca de mensagens:



Há diversas formas para medir o erro. Em geral, erros pequenos são menos perceptíveis e, portanto, costuma-se usar alguma métrica objetiva que evidencie os erros maiores. Este é o caso da Média dos Erros Quadráticos (MSE – *Mean Squared Error*). Considerando que tanto Ori quanto Dec tenham tamanho  $n$ , cada símbolo pode ser indexado, respectivamente, como  $ori_i$  e  $dec_i$ , onde  $1 \leq i \leq n$ . Vamos assumir que os símbolos sejam valores numéricos, assim, a MSE é definida como:

$$\text{MSE}(\text{Ori}, \text{Dec}) = \frac{1}{n} \sum_{i=1}^n (ori_i - dec_i)^2 \quad (1)$$

Outra métrica bastante utilizada, principalmente na compressão de sinais é a chamada Relação Sinal-Ruído de Pico (PSNR – *Peak Signal-to-Noise Ratio*), definida (em dB) como:

$$\text{PSNR}(\text{Ori}, \text{Dec}) = 10 \times \log_{10} \left( \frac{(2^b - 1)^2}{\text{MSE}(\text{Ori}, \text{Dec})} \right) \quad (2)$$

Onde  $b$  é o número de bits por símbolo. Quanto maior o PSNR melhor é a qualidade da imagem, ou seja, menores são os erros. Notem que quando Ori e Dec são iguais, isto é, quando não há erros,  $\text{PSNR} = \infty$ . Utilizaremos a PSNR para medir os erros causados em nossos padrões CUI.

## 8 Roteiro da Parte III

1. Analise a classe *PSNR.java* que calcula o PSNR a partir de um BMP original e um BMP decodificado a partir de um arquivo CUIF.x.
2. Implemente os métodos *mae* e *psnr* da classe *PSNR.java* (linha 203) para retornar o PSNR obtida pela conversão RGB para YCbCr.
3. Teste a implementação usando o comando a seguir, sendo que JazzMan1.bmp foi decodificado a partir do CUIF.1. Como os píxeis são iguais, o valor do PSNR deveria ser infinito.

```
java PSNR JazzMan.bmp JazzMan1.bmo
```

## 9 Questões a serem respondidas no relatório

Entregue no relatório o código fonte com as implementações realizadas e com as respostas das seguintes questões:

**Questão 1.** Há alguma compressão entre CUI.1 e CUI.2? Expliquem.

**Questão 2.** Indique o PSNR comparando a imagem original com as imagens obtidas a partir dos arquivos CUIF.1 e CUIF2. Há perdas nos dados da imagem na conversão  $\text{RGB} \rightarrow \text{YCbCr} \rightarrow \text{RGB}$ ? Explique porque.

## 10 Cuif.3 Compactado usando Codificação de Huffman

Vimos na aula teórica como funciona a codificação de Huffman para uma mensagem cujas distribuições de probabilidade são conhecidas (ou são calculadas). Devemos lembrar que a codificação de Huffman está em média a no máximo 1 bit/símbolo da compressão ótima sem perdas, que é relacionada à entropia dessa mesma mensagem. Dado um alfabeto  $\mathcal{A}$ , com  $n$  símbolos e distribuição de probabilidades  $\text{PMF} = \{p_1, p_2, \dots, p_n\}$ , a entropia  $H(\mathcal{A})$ , em bits, pode ser definida como:

$$H(\mathcal{A}) = \sum_{i=1}^n \left( p_i \times \log_2 \left( \frac{1}{p_i} \right) \right) \quad (3)$$

Para não ocorrer erros de ponto flutuante, nossa implementação de Huffman utiliza histogramas para obter a distribuição de símbolos. Basicamente, para cada símbolo há uma contagem de sua ocorrência. Ou seja, um histograma é um mapeamento  $histogram(x)$  entre o símbolo  $x$  e o número de ocorrências de  $x$  em uma mensagem  $m$ . Por exemplo, supondo a mensagem  $m = \{3, 3, 2, 5, 1, 1, 1, 2, 3, 5, 1, 4, 3, 3, 3, 3\}$ ,  $histogram(.)$  pode ser visto como:

Tabela 1: Exemplo de histograma

símbolo	# de ocorrências
1	4
2	2
3	7
4	1
5	2
total	16

Outra forma de visualizar histogramas é através de um gráfico, onde o eixo das abscissas apresenta os símbolos, enquanto o eixo das ordenadas apresenta a frequência de ocorrência.

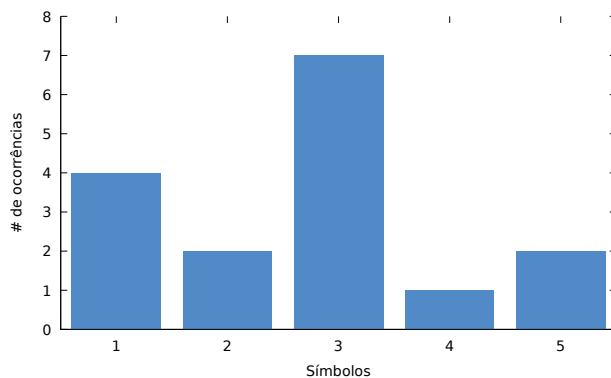
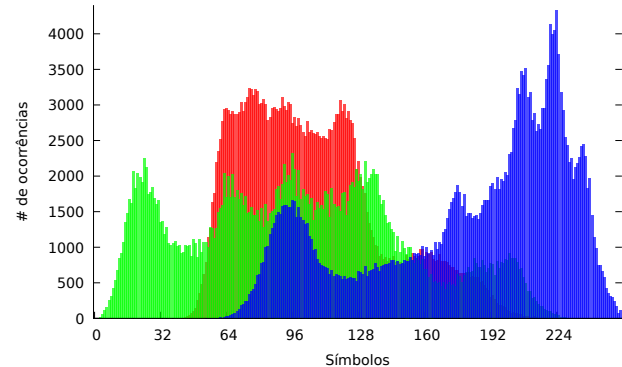
(a) Exemplo de histograma de  $m$ .(b) Exemplo de histograma dos canais R, G e B da imagem `JazzMan.bmp`.

Figura 1: Exemplos de histogramas.

Assim, pode-se utilizar os histogramas das mensagens para criar as árvores de Huffman.

Na implementação disponibilizada, o CUIF.3 converte a representação RGB para YCbCr antes de executar a codificação de Huffman. Note que a codificação de Huffman em si não gera descarte de informação, por ser uma técnica de codificação sem perdas.

## 11 Roteiro da Parte IV

1. Execute os comandos a seguir para gerar arquivo `JazzMan.cuif` na versão 3 e decodificar esta imagem novamente para bitmap. Observe o console, imprimindo o histograma e a tabela de codificação de Huffman.

```
java bmp2cuif -version 3 JazzMan.bmp JazzMan3.cuif
java cuif2bmp JazzMan3.cuif JazzMan3.bmp
```

2. Obtenha o Erro PSNR do arquivo CUIF.3 decodificado, comparando com o CUIF.1 e CUIF.2

```
java PSNR JazzMan.bmp JazzMan3.bmp
java PSNR JazzMan2.bmp JazzMan3.bmp
```

## 12 Questões a serem respondidas no relatório

Entregue no relatório as respostas das seguintes questões:

**Questão 3.** Calcule a taxa de compressão em relação ao arquivo JazzMan1.cuif (CUIF.1 sem compressão) e JazzMan3.cuif (CUIF.3 com Codificação de Huffman). Dica: A taxa de compressão da versão V em relação à versão X é  $\frac{\text{tamanho CUI.X}}{\text{tamanho CUI.V}} : 1$ . Por exemplo, se CUI.X resulta em 100 KB e CUI.V resulta em 50 KB, a taxa de compressão de V em relação a X é de 2:1. Comente também no relatório o princípio da compressão usada na codificação de Huffman, para isto analise o histograma e a tabela de códigos impressos.

**Questão 4.** Indique o PSNR comparando a imagem original JazzMan.bmp com a imagem obtida a partir do arquivo CUIF.3. Há perdas nos dados da imagem? Explique porque.

## 13 Cuif.4 Compactado usando RLE

Na implementação de RLE disponibilizada, 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:

Tabela 2: 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

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 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 1 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 (Eq. 2):

Tabela 3: Codificação RLE conforme implementada para CUIF.4

binário	significado
10000010	o próximo byte é repetido 2× (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× (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× (3, 3, 3, 3)
00000011	byte que é repetido: 3

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

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.

## 14 Roteiro da Parte V

1. Usando o bmp2cuif, gere, a partir da JazzMan.bmp arquivo no formato CUIF.4, chamado JazzMan4.cuif e em seguida gere o arquivo bmp a partir de cada um dos arquivos cuif (chamado JazzMan4.bmp).
2. Calcule a PSNR das codificações CUIF.4, usando para tal o arquivo JazzMan.bmp (original) e o arquivo JazzMan4.bmp.

## 15 Questões a serem respondidas no relatório

Entregue no relatório as respostas das seguintes questões:

**Questão 5.** Qual a taxa de compressão obtida com CUIF.4? ).

**Questão 6.** Indique a PSNR das codificações CUIF.4. Compare com a PSNR do CUIF3 e justifique.

**Questão 7.** Codifiquem as imagens JazzMan.bmp e lena.bmp usando CUIF.4. Qual imagem obteve maior compressão? Explique porque.