

Representação digital de imagens

1 Introdução

Um **padrão de representação/compressão** é formado por uma série de regras, que definem como os dados serão representados digitalmente. Assim, um padrão não apresenta em si as técnicas para a codificação, mas sim define a organização dos dados. Para ser compatível com o padrão, o codificador deve, portanto, gerar um fluxo de dados no formato definido.

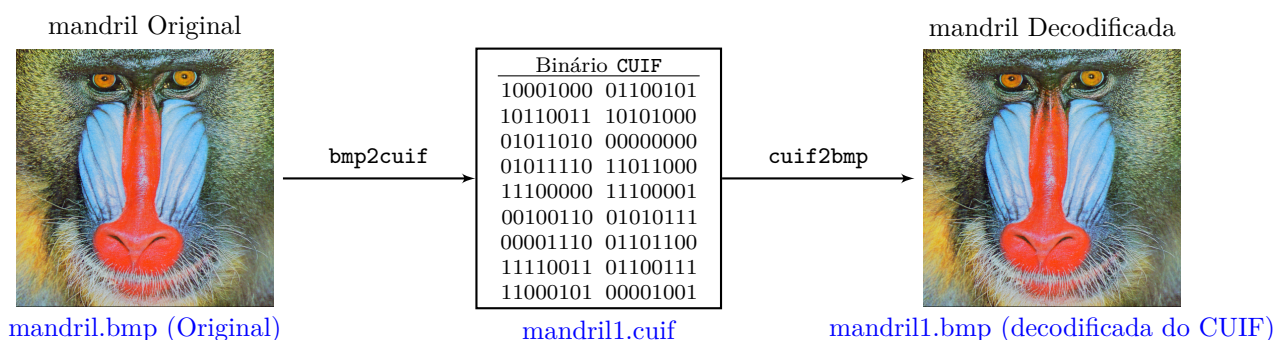
Nesta sequência de aulas práticas sobre imagens, desenvolveremos uma série de padrões de representação/-compressão de imagem digital. Em cada aula prática, iremos incrementar nosso padrão, gerando assim novas versões. Chamaremos nosso padrão inicial de **CUI.1: C**Ustom Image versão 1; Já seu formato de arquivo será chamado de ***C*Ustom *I*mage *F*ormat** (ou **CUIF**). Assim, a cada nova versão teremos um novo padrão. Porém o formato de arquivo será o mesmo, independente do padrão.

Para visualizarmos os efeitos da compressão, devemos utilizar algum formato de representação de imagens conhecido, e fornecer meios de converter entre um padrão e outro. Um formato de arquivo comum é o chamado bitmap, ou BMP. A vantagem de usarmos tal formato é sua capacidade de representação de imagens sem compressão e, portanto, sem distorções. Assim, teremos uma baseline para comparação: tanto de taxa de compressão quanto de qualidade.

Nesta aula prática está disponível o arquivo Cuif.py que implementa parcialmente funções para manipulação de arquivos CUIF:

1. **bmp2cuif**: para conversão de BMP para CUIF. No caso do código Python entregue, é necessário instanciar um objeto cuif da classe Cuif, e em seguida chamar o método `cuif.save()` para salvar o arquivo cuif;
2. **cuif2bmp**: para conversão de CUIF para BMP. No caso do código Python entregue, é necessário instanciar um objeto cuif, e chamar o método `cuif.saveBMP()`. Para instanciar este objeto Cuif pode-se utilizar o construtor da classe, ou então chamar o método estático `Cuif.openCUIF(<nome do arquivo>)`.

Assim, o fluxo para visualização do efeito da codificação através do padrão CUI é o seguinte:



Fonte da imagem: <http://sipi.usc.edu/database/database.php?volume=misc&image=12#top>

Para isso, devemos primeiramente entender os dois formatos de arquivo.

2 Formato BMP

O formato BMP tem um cabeçalho no início do arquivo. Tal cabeçalho provê as informações necessárias para a interpretação dos dados do arquivo. A Tabela 1 apresenta a descrição do cabeçalho BMP. Por exemplo, o campo *offset* do cabeçalho BMP, nos bytes 10-13, indica a posição onde termina o cabeçalho e começa a parte de dados da imagem.

Para manter o projeto simples, trataremos apenas de BMPs de 3 bytes/*pixel* e sem nenhuma compressão. Vamos considerar esta restrição para apresentar o modo como os pixels são codificados no bitmap (bitmap aqui é a matriz de pixels, aqui representada pelos componentes RGB).

Considerando apenas imagens BMP sem compressão, na parte de dados do arquivo BMP, os *pixels* da imagem são codificados em sequência *raster*, que segue da esquerda para a direita e de cima para baixo. No arquivo BMP, os pixels iniciam na posição indicada pelo campo *offset*. Há duas características a serem observadas:

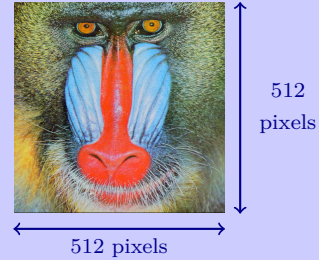
1. Os três canais de cor de cada *pixel* são codificados em sequência, sendo um byte por canal (**R**, **G** e **B**). Porém, ao invés de codificar **R**, **G** e **B**, a ordem adotada no BMP é **B**, **G** e **R**;
2. O comprimento de cada linha da imagem, em bytes, deve ser sempre múltiplo de 4. Caso a imagem não tenha uma resolução vertical múltipla de 4, inserem-se bytes com valor 0 até preencher a linha. Este procedimento é denominado de *padding*.
3. No raster (parte de dados do arquivo de imagem), a sequência de linhas da imagem estão invertidas em relação a visualização da imagem. Ou seja, a primeira linha da imagem no arquivo é na realidade a última linha da imagem.

Tabela 1: Especificação do Cabeçalho BMP

Offset	Tamanho	Descrição
0	2	assinatura (identificador), deve ser 4D42 ₁₆
2	4	tamanho do arquivo BMP em bytes (não é confiável)
6	2	reservado, deve ser 0
8	2	reservado, deve ser 0
10	4	offset, em bytes, até o início dos dados da imagem
14	4	tamanho da estrutura BITMAPINFOHEADER, deve ser 40 ₁₀
18	4	número de <i>pixels</i> na horizontal (largura)
22	4	número de <i>pixels</i> na vertical (altura)
26	2	número de planos na imagem, deve ser 1
28	2	número de bits por <i>pixel</i> (1, 4, 8, ou 24)
30	4	tipo de compressão (0=nenhuma, 1=RLE-8, 2=RLE-4)
34	4	número de bytes da imagem (incluindo <i>padding</i>)
38	4	resolução horizontal em <i>pixels</i> /m (não é confiável)
42	4	resolução vertical em <i>pixels</i> /m (não é confiável)
46	4	número de cores na imagem, ou zero
50	4	número de cores importantes, ou zero

Exemplo de cabeçalho BMP

Vamos usar a imagem `mandril.bmp` como exemplo e criar um cabeçalho de arquivo BMP. Tal imagem tem resolução 512×512 *pixels* e 24 bpp (bits por *pixel*). A imagem BMP descrita não terá nenhum tipo de compressão e não indexará cores.



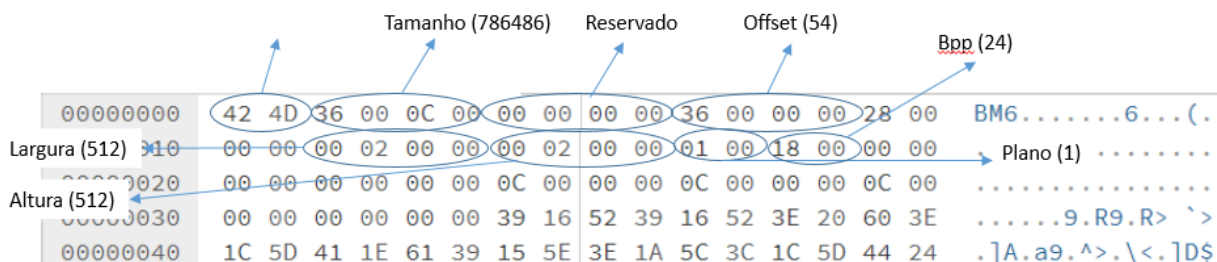
Devemos também calcular o tamanho do arquivo. Os dados ocuparão:

$$\# \text{ bits na imagem} = \text{largura} \times \text{altura} \times \text{bpp}$$

byte	valor		significado
	2	0	
0	–	4D42 ₁₆	assinatura bmp
2	786486 ₁₀		tamanho do arquivo
6	–	0	reservado
8	–	0	reservado
10	54 ₁₀		offset para o início do arquivo
14	40 ₁₀		fixo
18	512 ₁₀		largura
22	512 ₁₀		altura
26	–	1	planos (deve ser 1)
28	–	24 ₁₀	bpp
30	0		sem compressão
34	786432 ₁₀		número de bytes
38	786432 ₁₀		<i>pixels</i> /m
42	786432 ₁₀		<i>pixels</i> /m
46	0		cores na imagem
50	0		cores importantes

Vejam na Figura 1 como o cabeçalho deve ser interpretado no hexedit.it.

Figura 1: Exemplo de um cabeçalho real



3 CUIF

De maneira similar ao formato BMP, o CUIF inicia com um cabeçalho apresentado na sequência.

Offset	Tamanho	Descrição
0	4	assinatura (identificador), deve ser "CUIF"
4	1	versão do padrão CUI
5	1	número de estudantes no grupo (Nestud)
6	4	largura da imagem (em pixels)
10	4	altura da imagem (em pixels)
14	4×Nestud	Matrículas dos estudantes (4 bytes cada)

Após o cabeçalho inicia a parte de dados da imagem. O modo como estes dados serão organizados depende da versão do padrão CUIF utilizado.

3.1 CUI.1

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 2×2 *pixels*:

red = (FF 00 00)₁₆  green = (00 FF 00)₁₆
blue = (00 00 FF)₁₆  gray = (B7 B7 B7)₁₆

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
18	–	–	–	FF ₁₆	R pixel 0,0
19	–	–	–	00 ₁₆	R pixel 0,1
20	–	–	–	00 ₁₆	R pixel 1,0
21	–	–	–	B7 ₁₆	R pixel 1,1
22	–	–	–	00 ₁₆	G pixel 0,0
23	–	–	–	FF ₁₆	G pixel 0,1
24	–	–	–	00 ₁₆	G pixel 1,0
25	–	–	–	B7 ₁₆	G pixel 1,1
26	–	–	–	00 ₁₆	B pixel 0,0
27	–	–	–	00 ₁₆	B pixel 0,1
28	–	–	–	FF ₁₆	B pixel 1,0
29	–	–	–	B7 ₁₆	B pixel 1,1

4 Roteiro

1. Baixe os arquivos py e imagens disponibilizados no moodle, e abra em seu ambiente de desenvolvimento preferido.
2. Abra o arquivo `mandril.bmp` em um editor hexadecimal, por exemplo, como aquele em <https://hexed.it/>, e responda a primeira questão do Relatório.
3. Abra o arquivo `praticaIII.py` e inclua na lista matrículas os números de matrículas dos estudantes no grupo;
4. Execute o arquivo `praticaIII.py` que converte a imagem `mandril.bmp` para CUI.1, e verifique que o arquivo `mandril1.cuif` foi criado.
5. Façam a conversão inversa (`cuif2bmp`) usando o arquivo `praticaIII.py` utilizando o código comentado.
6. Verifiquem se os números de matrícula de todos os alunos no grupo foram exibidos no console;
7. Abra as imagens `mandril.bmp` e `mandril1.bmp` com algum visualizador e verifique que as duas imagens são diferentes.

5 Calculando Ruído

A figura abaixo representa um esquema de codificação e decodificação. Na entrada, a mensagem original (Ori) é codificada e posteriormente decodificada, resultando em uma mensagem decodificada (Dec). Em uma codificação com perdas de informações, a mensagem decodificada (Dec) é diferente da mensagem original (Ori). Neste caso, diz-se que o codificador gerou ruído na mensagem original.



Há diversas formas para medir o erro gerado pelo codificador. Uma destas métricas é a 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:

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

Para determinar o MSE entre uma imagem Ori e uma imagem Dec, o tamanho de símbolos é a quantidade de componentes de cor de uma imagem (3 componentes por píxel). Assim, o tamanho n de uma imagem de largura w e altura h é igual a $3 \cdot h \cdot w$.

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:

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

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, $PSNR = \infty$. Utilizaremos a PSNR para medir os erros causados em nossos padrões CUI.

6 Relatório

Entregue via Moodle o relatório contendo as respostas das questões abaixo e o código modificado:

Questão 1. Abra o arquivo `mandril.bmp` no editor hexadecimal em <https://hexed.it/> e, analisando o formato do cabeçalho BMP apresentado na Seção 2, indique no relatório: qual é o valor dos campos *offset* e *tamanho do arquivo*? Quais são os valores dos componentes de cor R, G e B do primeiro pixel armazenado no arquivo?

Questão 2. Qual é o tamanho do cabeçalho do arquivo `mandril1.cuif` para seu grupo?

Questão 3. No arquivo `praticaIII.py` tem um função PSNR incompleta. Implemente esta função de maneira a calcular o PSNR passando como parâmetro a imagem original e uma decodificada. Implemente o cálculo do MAE e PSNR com base nas fórmulas da seção 5 .

Questão 4. Indique o PSNR comparando a imagem original `mandril.bmp` (original) com a imagem obtida a partir do arquivo CUIF.1 (`mandril1.bmp`). Para isto, utilize o comando `java PSNR mandril.bmp mandril1.bmp`

Questão 5. Indique o PSNR comparando a imagem original `mandril.bmp` (original) com a imagem obtida a partir do arquivo CUIF.1 corrigido (`mandril1.bmp`). Justifique a resposta do PSNR.

Questão 6. Compacte as imagens `mandril.bmp` e `mandril1.cuif` com zip. Qual a taxa de compressão obtida para os dois arquivos? Qual arquivo compactou mais? Explique porque deste resultado, ou seja, indique a vantagem de organizar os pixels nesta sequência definida pelo CUIF.1 (primeiro os valores de R, depois de G e finalmente de B) para a compressão baseada em RLE ou DPCM? Dica: relembre os princípios da compressão RLE e DPCM e compare a parte de dados de imagem do arquivo `mandril.bmp` e `mandril1.cuif` no editor hexadecimal.