

INF100 – Introdução à Programação I
Roteiro da Aula Prática 10 – 26 de novembro de 2020
Arranjos bidimensionais - Imagens
Valor: 5 pontos

Introdução

O processamento de imagens é uma das aplicações mais interessantes de arranjos bidimensionais. Conforme visto anteriormente, uma foto colorida guardada em um arquivo (.jpg, .png, etc.) pode ser representada em um programa por uma matriz de pixels. E cada pixel é representado por uma tupla de três componentes, representando as cores primárias (r = vermelho, g = verde, b = azul).

O arquivo `imagens.py` tem funções prontas para manipulação de um arquivo de imagem. Em particular, as funções que mais nos interessam são: `Imagem()`, `mostrar()`, `altura` e `largura`.

O arquivo `p10.py` já traz pronto o trecho de código usando os comandos que serão comentados a seguir, usando o estilo de comentário de um programa Python. Após o comentário é apresentado um exemplo de uso de cada função.

```
# importa a "biblioteca" imagens, o que permitira usar as funções lá definidas
import imagens
```

```
# lê o arquivo de nome arquivo.jpg (que deve estar no mesmo diretório do p10.py) a
# atribui para im os pixels da figura
im = imagens.Imagem('arquivo.jpg')
```

Portanto, **im é um arranjo bidimensional (ou uma matriz) de pixels**. Cada pixel é representado pelos componentes r, g, b, conforme mencionado no primeiro parágrafo. Um comando:

```
print( im[10][10] )
```

Irá exibir na tela algo do tipo:
(219, 240, 248)

Que representa a cor do pixel da posição 10, 10 (décima primeira linha e décima primeira coluna da imagem, armazenada em `im`).

Esta combinação dos componentes (r = 219, g = 240 e b = 248) corresponde a um pixel azul claro. Se apenas o componente b for alterado, por exemplo de 248 para 249, dificilmente alguém perceberia a mudança na cor. Lembrem-se que com este esquema de codificação, existem 2^{24} (ou 16.777.216) cores diferentes.

Esta informação é importante, pois nem tudo que nos parece ser branco, é realmente um pixel branco, cujos valores dos componentes r, g, b são (255,255,255). Da mesma forma nem todos os pixels que nos parecem pretos, têm de fato a combinação (0, 0, 0) que são os valores (r, g, b) do pixel preto.

```
# mostra na tela do computador a figura correspondente aos pixels da matriz im
im.mostrar() # Mostrar a imagem na tela
```

Observe que uma alteração nos pixels da matriz `im` será exibida pela função `mostrar()`. Dependendo da alteração ela será perceptível aos nossos olhos ou não.

No início do programa p10.py existe um comando para exibir a figura original, antes de qualquer processamento.

Se após este comando de exibição for executado o trecho abaixo:

```
# quadrado negro 10 x 10 pixels
for i in range ( 0, 10 ) :
    for j in range (0, 10 ) :
        im[i][j] = (0, 0, 0)
```

im.mostrar() **# este comando está (E DEVE FICAR SEMPRE) fora do comando repetitivo**

A nova exibição da imagem aparecerá com um quadrado PRETO de 10x10 pixels.

A leitura dos componentes de um pixel, conforme visto na videoaula e na aula teórica pode ser feita com o comando:

```
# lê os componentes de um pixel da imagem im. 0<= i <m e 0<= j <n
r, g, b = im[i][j]
```

Muitas vezes é necessário aplicar uma transformação em toda a imagem. Outras vezes, a transformação deverá ocorrer apenas com algumas linhas ou colunas. Assim, é importante saber quantas linhas e quantas colunas a imagem possui. Para isso podem ser usadas as funções descritas a seguir:

```
# atribui para a variável m o número de linhas da matriz im. O número de pixels da
# imagem na vertical
m = im.altura
```

```
# atribui para a variável n o número de colunas da matriz im. O número de pixels da
# imagem na horizontal
n = im.largura
```

Desta forma podemos usar m e n como limites para percorrer uma matriz de pixels, da mesma forma que fazíamos com outras matrizes pois, na nossa convenção m = número de linhas e n = número de colunas.

Instruções


Nesta prática DEVE ser utilizado o comando repetitivo *for* para manipulação/utilização dos elementos do arranjo bidimensional.

Nome do arquivo a ser entregue: **p10.py**

Importante: Como qualquer outra prática de INF100 você deve:

1. Criar o cabeçalho obrigatório.
2. Após finalizar o cabeçalho salve o arquivo com o nome correto
3. Leia as instruções até o final e, após finalizar sua leitura, inicie sua programação.

Obs.: Recomenda-se salvar o arquivo com certa frequência para não perder a digitação já feita em caso de uma falha na rede elétrica.

 Não esqueça de preencher o cabeçalho com seus dados e uma breve descrição do programa.

Após certificar-se que seu programa está correto, envie o arquivo do programa fonte (**p10.py**) através do sistema do LBI.

Questão a ser Resolvida

- 1) No servidor do LBI, faça o download do arquivo **imagens.py**, além dos arquivos de nome **Pedestre.py**, com código (ainda incompleto) para abrir o arquivo de imagem de nome **Pedestre.png**.

Após baixar os 3 arquivos, altere o nome do arquivo Pedestre.py para p10.py, pois o seu trabalho deverá ser entregue com este nome (p10.py).

Para alterar o nome, após abrir o arquivo no IDLE use a opção “Save As” (Salvar Como) e dê o nome p10.py

Complete o programa do arquivo renomeado (p10.py) para que ele execute a seguinte tarefa:

- a) Mudar a cor da figura representando o pedestre e a seta ao seu lado de preto para VERDE, sem alterar a cor dos pixels do entorno.

O tom de cor verde deve ser próximo ao do exemplo, mas não exatamente igual. Dica: para produzir um tom de verde claro, o componente verde (g) deve estar mais próximo de 255 do que de 0. Para verde escuro é o contrário. Os componentes vermelho (r) e azul (g) não devem ser nem 0 (zero) e nem acima de 50.

A seguir são mostradas a figura original e a figura após processamento do programa a ser feito por você.



Imagem original



Imagem processada

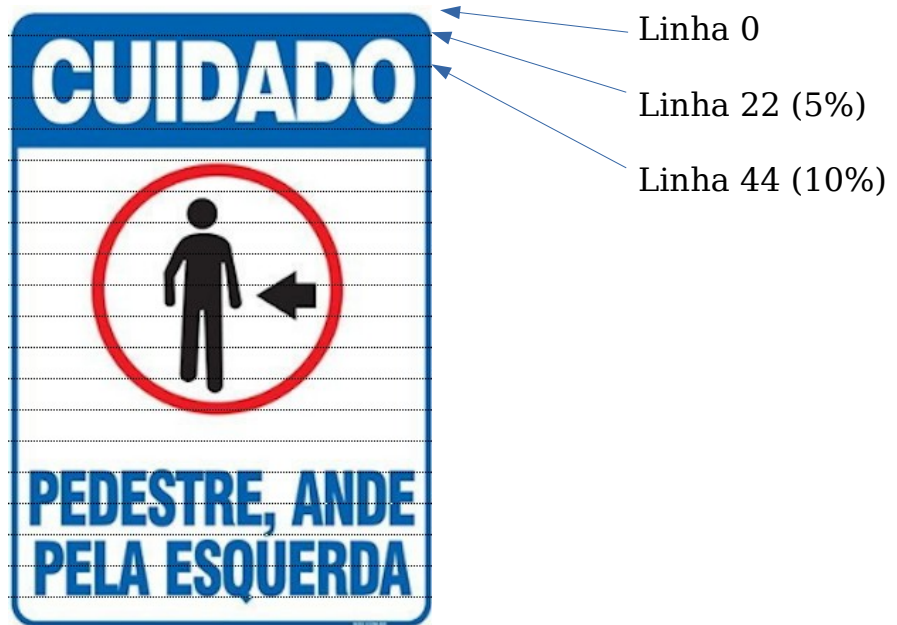
O algoritmo da solução pode ser descrito da seguinte forma:

```

para i = linha_inicial até linha_final :
    para cada coluna da linha i:
        lê os componentes de um pixel
        se "valores de (r, g, b)" == "tom de preto"
            altera valores de (r, g, b) para tom de verde
        atualiza pixel

```

Para auxiliar na identificação da região a ser considerada (linha_inicial e linha_final), a figura original foi processada para incluir uma linha tracejada de 5 em 5% da altura da imagem.



Conforme dito anteriormente o teste para verificar se o pixel é “tom de preto” não deve ser verificar se $r = g = b = 0$.

A figura foi analisada por um outro programa e a estatística mostrou que não existem pouquíssimos pontos totalmente pretos ($r=0, g=0, b=0$) e menos pixels totalmente brancos ($r=255, g=255, b=255$) do que nossa visão nos sugere.

Número de pixels: 440 linhas x 302 colunas = 132.880 pixels
 Pixels pretos: 448 ou (0.34%)
 Pixels brancos: 30429 ou (22.90%)
 Outras cores: 102003 ou (76.76%)

A saída do seu programa deve gerar uma imagem o mais parecido possível com a mostrada no exemplo.

Após certificar-se que seu programa está correto, envie o arquivo do programa fonte (**p10.py**) através do sistema do LBI.

A entrega deverá ser feita até às 20h20 (1h50 após o início da aula prática)