PROCESSAMENTO DIGITAL DE IMAGEM - LISTA DE EXERCÍCIOS 03

Os scripts que serão apresentados nesse documento foram desenvolvidos em "Python" (versão 3.5.2), para a execução dos mesmos será necessário utilizar as dependências "Numpy" (versão 1.13) e "Opencv" (versão 3.3), e para reutilização de código foi criada a classe "Utils" apresentada no apêndice A.

Problema 1.1:

a) Quantos tons de cinza existem no sistema de cor RGB na qual cada imagem RGB possui 8-bit?

Uma imagem RGB de 8-bit, possui 256 tons de cinza representados de 0 (preto) até 255 (branco). Cada canal de uma imagem RGB possui 255 tons separados para os valores da cor vermelha, verde e azul.

b) Numa imagem RGB, os componentes R, G, e B da imagem tem perfil horizontal de intensidade demostrado no diagrama da Figura 1. Que cor nós veríamos no meio da coluna desta imagem?

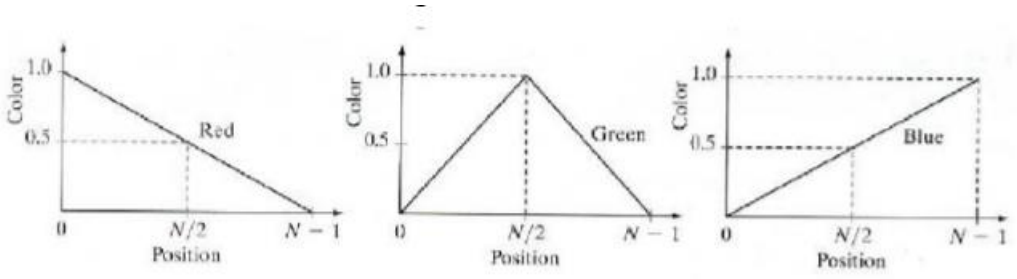


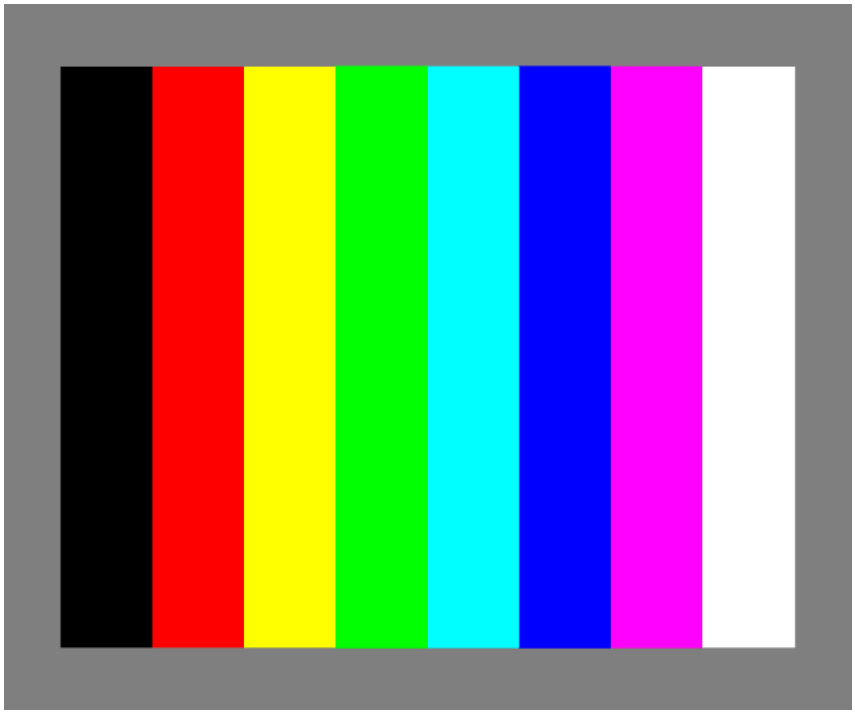
Figura 1: Perfil horizontal das componentes R, G e B.

Como cada canal de uma imagem RGB possui 255 tons, o perfil horizontal da componente vermelha representa a metade (128), o verde inteiro (255) e azul metade (128), gerando a seguinte cor representada na imagem a seguir.

A cor foi criada utilizando o script 1.1:

```
import Utils

tamanho = 500
imagem = Utils.criaImagem(tamanho,tamanho)

for linha in range(tamanho):
    for coluna in range(tamanho):
        imagem[linha, coluna, 0] = 128
        imagem[linha, coluna, 1] = 255
        imagem[linha, coluna, 2] = 128

Utils.mostraImagem(imagem)
```

Problema 1.2: Considere a imagem na Figura 2. Todas as cores têm valor máximo de saturação e intensidade.



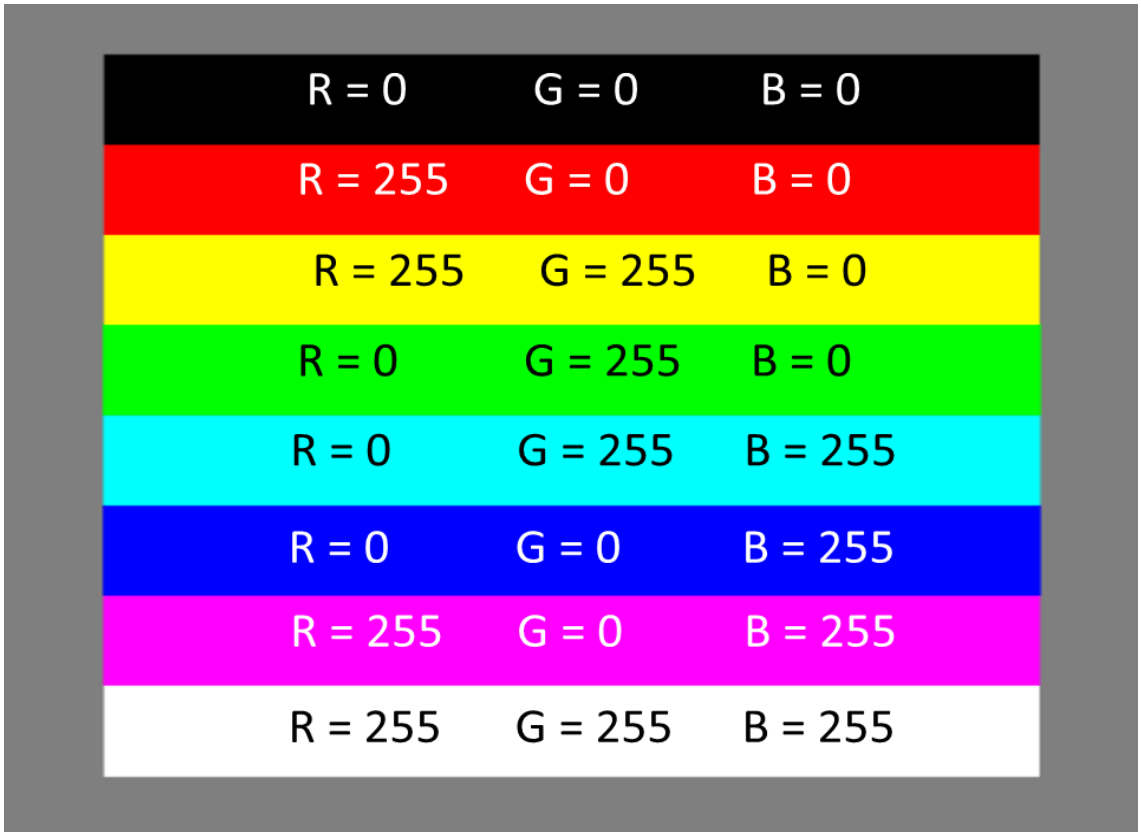
a) Desenhe as componentes RGB da Figura 2 como elas apareceriam em um monitor monocromático.

Utilizando o “script1.2a”:

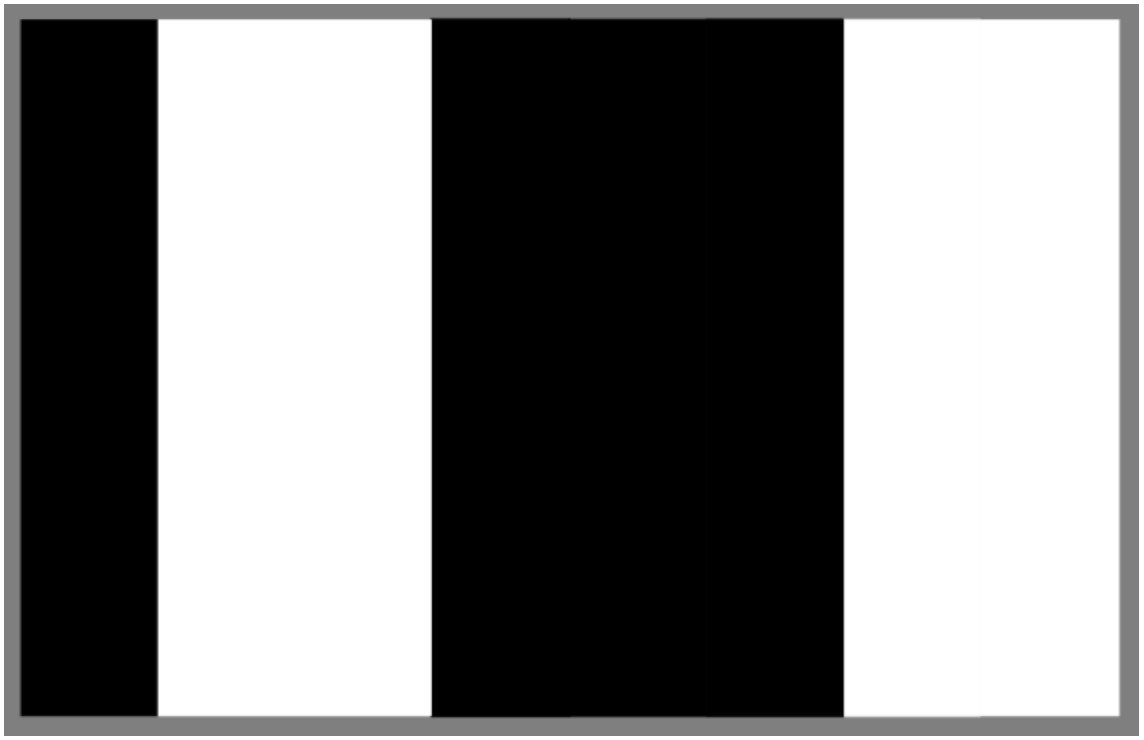
```
import Utils

imagem = Utils.leImagem("cores.tif")

B = imagem[:,:,0]
G = imagem[:,:,1]
R = imagem[:,:,2]

lista = []
lista.append(B)
lista.append(G)
lista.append(R)
Utils.mostraListaDeImagens(lista)
```

A imagem foi separada pelas suas componentes para representa-la respectivamente na sua forma monocromática, lembrando que a biblioteca “opencv” do script representa as cores na forma invertida “BGR”, a letra B (Blue) representa a cor azul, a letra G (Green) representa a cor verde e por fim a letra R (Red) representa a cor vermelhar.

Para explicar melhor à resolução do problema a imagem a seguir apresenta o valor do “RGB” das cores.

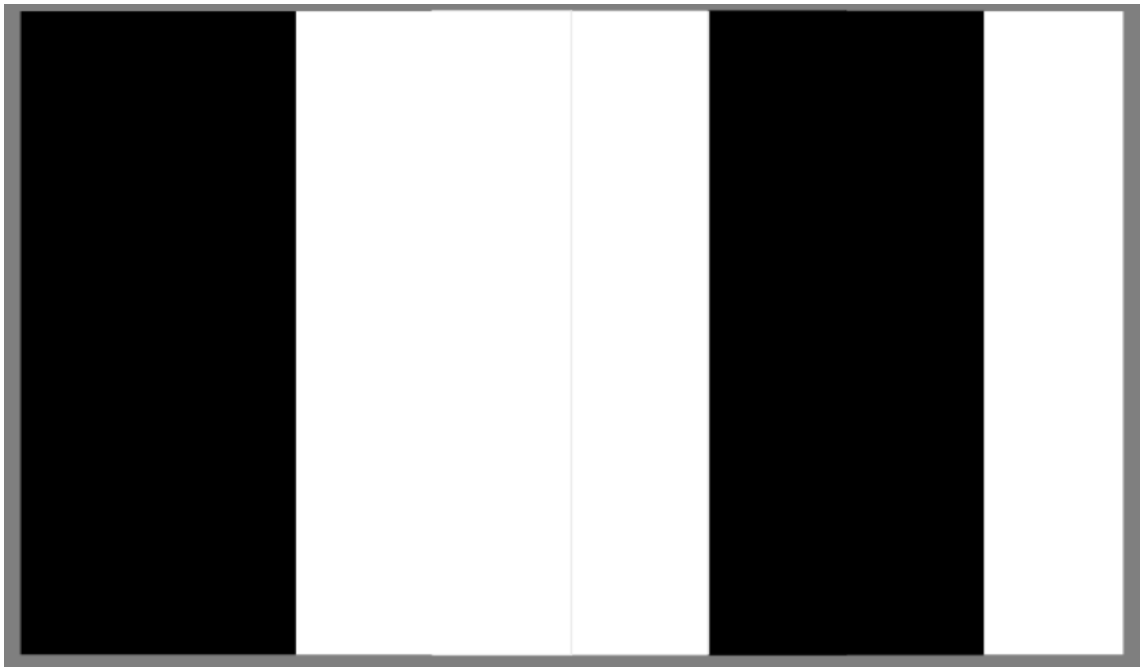


Para a componente vermelha (R):



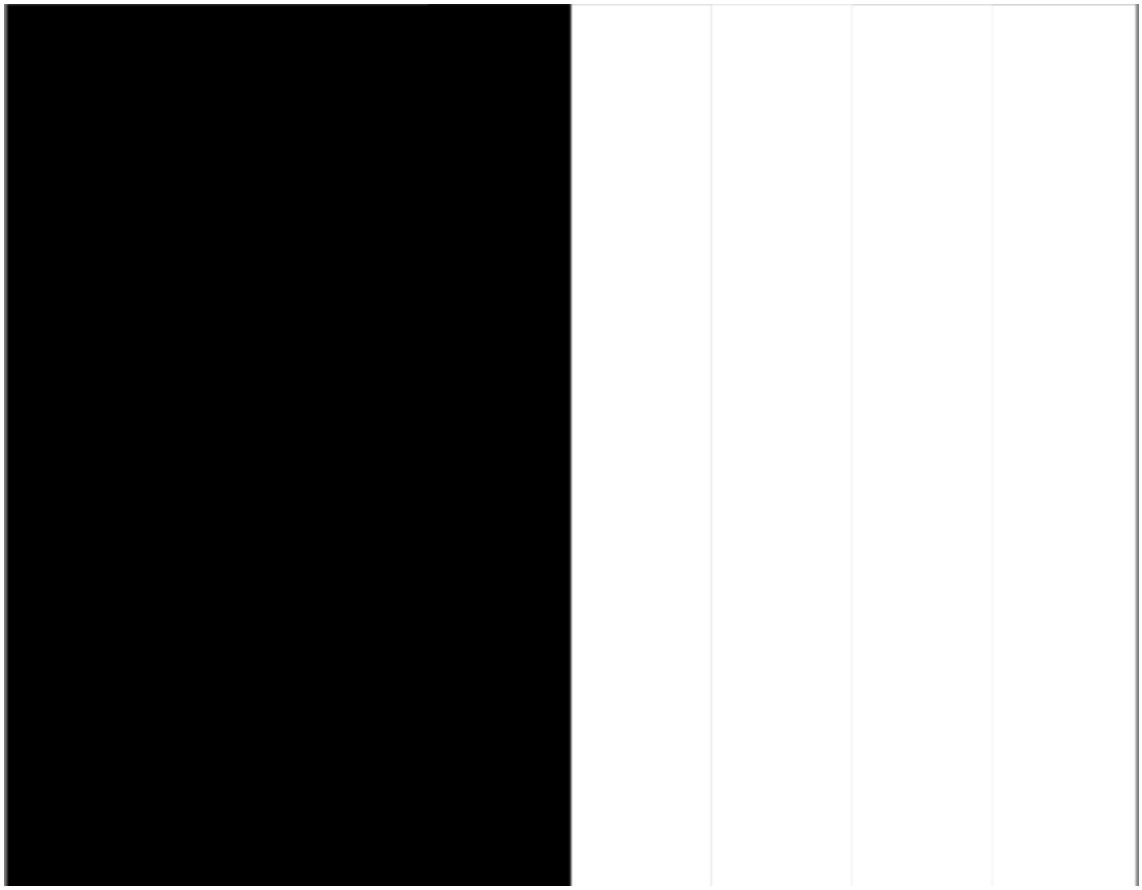
As cores que apresentam o componente vermelho ficam brancas e as que não possuem ficam pretas.

Para a componente verde (G):



As cores que apresentam o componente azul ficam brancas e as que não possuem ficam pretas.

Para a componente azul:



As cores que apresentam o componente verde ficam brancas e as que não possuem ficam pretas.

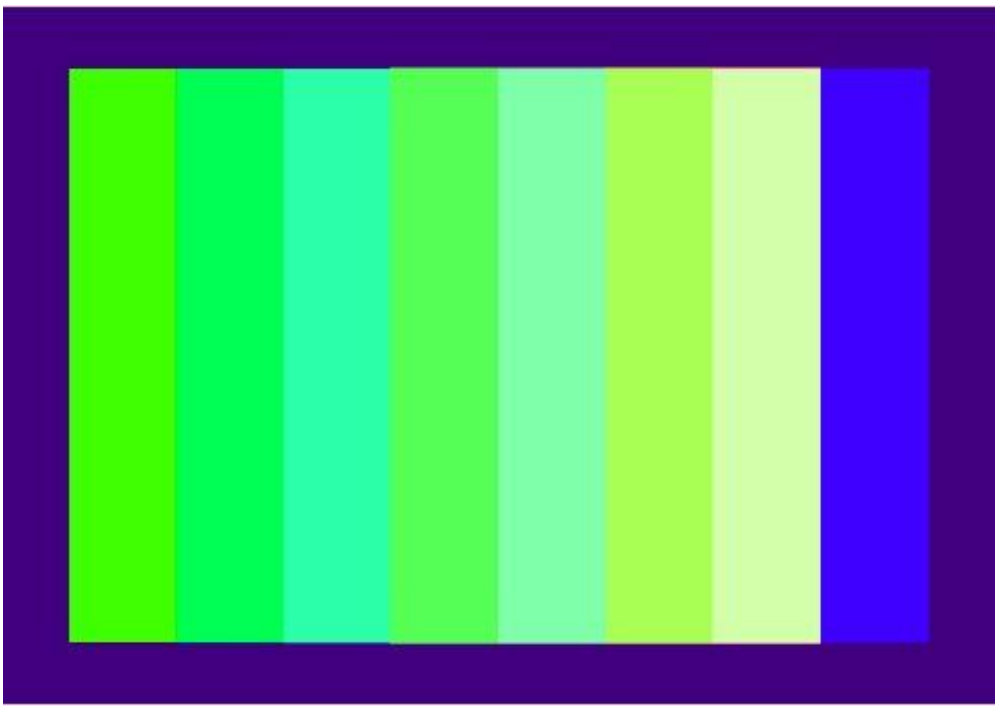
b) Desenhe as componentes HSI da Figura 2 como elas apareceriam em um monitor monocromático.

O sistema HSI é representado pela Tonalidade (Hue), Saturação (Saturation) e intensidade (Intensity).

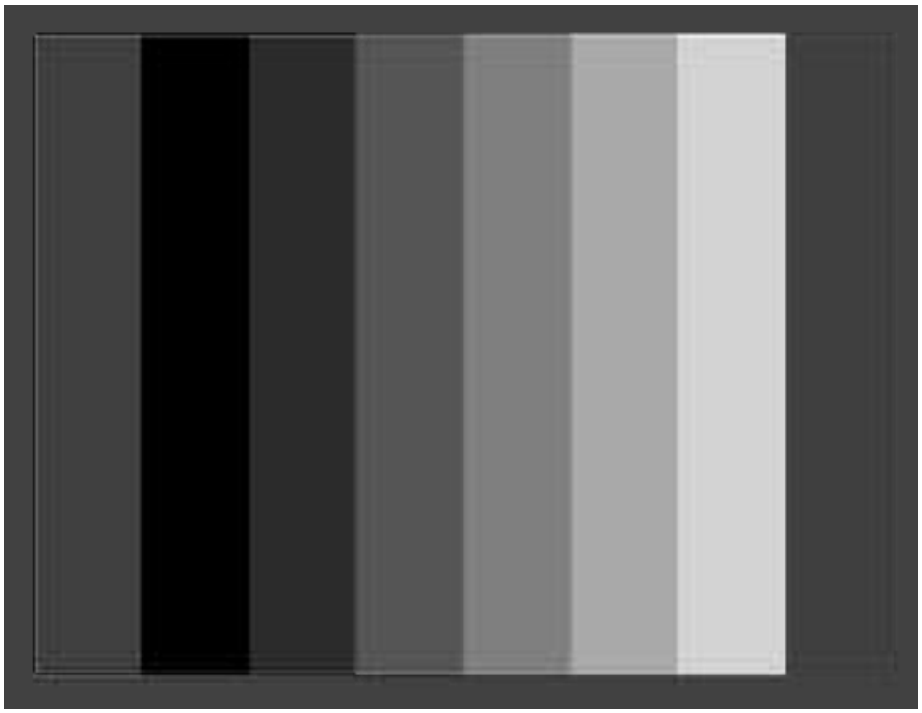
A biblioteca opencv, não possui função de conversão para o sistema HSI, assim a imagem anterior foi convertida no matlab utilizando o script do link a seguir:

http://www.mathworks.com/matlabcentral/fileexchange/13630-rgb-to-hsi-conversion?focused=5084558&tab=function

O resultado da conversão da imagem original é representado pela imagem a seguir:

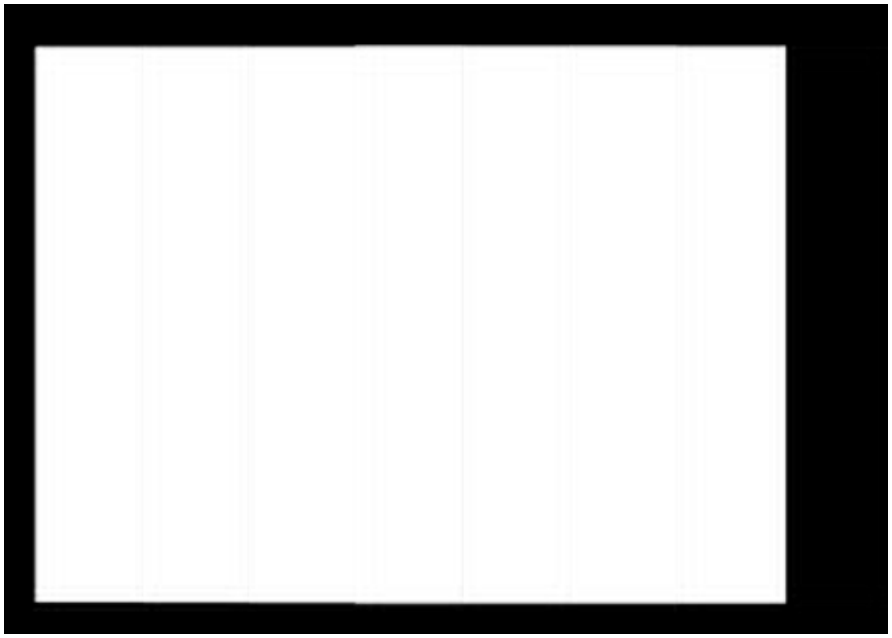


Resultado da imagem representada apenas no componente de Tonalidade (H):



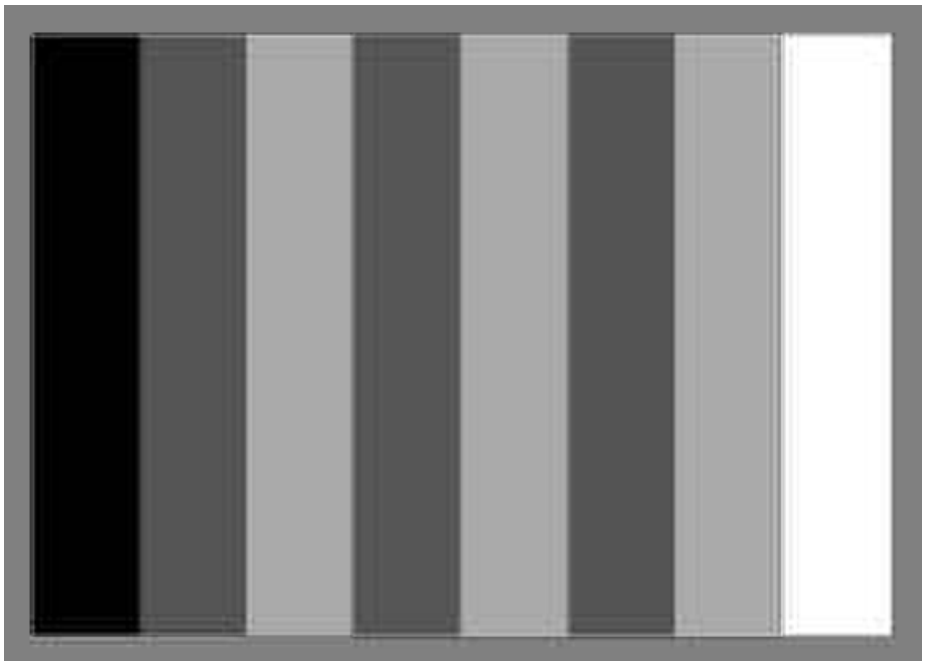
Essa componente representa a tonalidade de cinza utilizada para cada cor da imagem origina, essa característica define variantes de uma cor.

Resultado da imagem representada apenas no componente de Saturação (S):



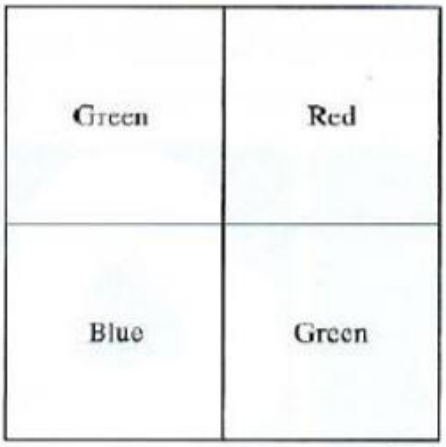
O componente de saturação dá uma medida de quanto à cor é diluída com luz branca.

Resultado da imagem representada apenas no componente de Intensidade (I):

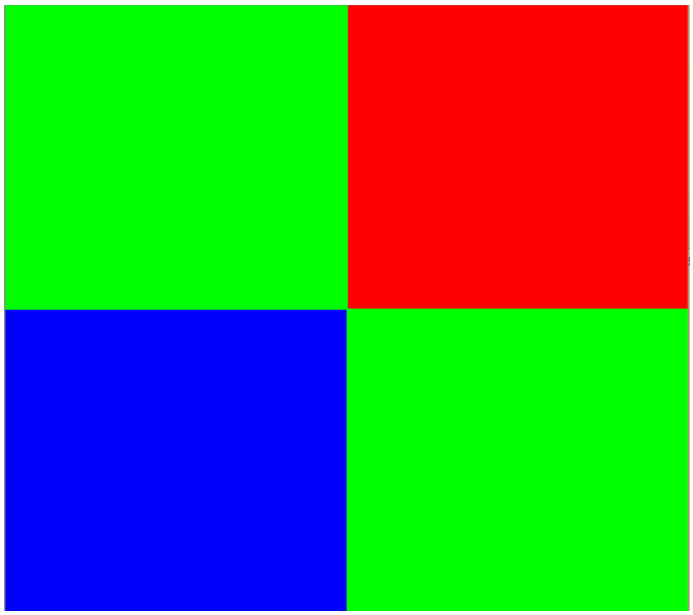


Quanto maior a intensidade, mais branco é a representação da cor, as cores que estão representadas por cinza escuro é feito a média de 255/3, e as cores de cinza claro, por ser uma mistura de cores primárias é feito a média de (255+255)/2.

Problema 1.3: Considere a imagem colorida de tamanho 500 x 500 da Figura 3, onde os quadrados correspondem às cores azul, vermelho e verde “puras”.



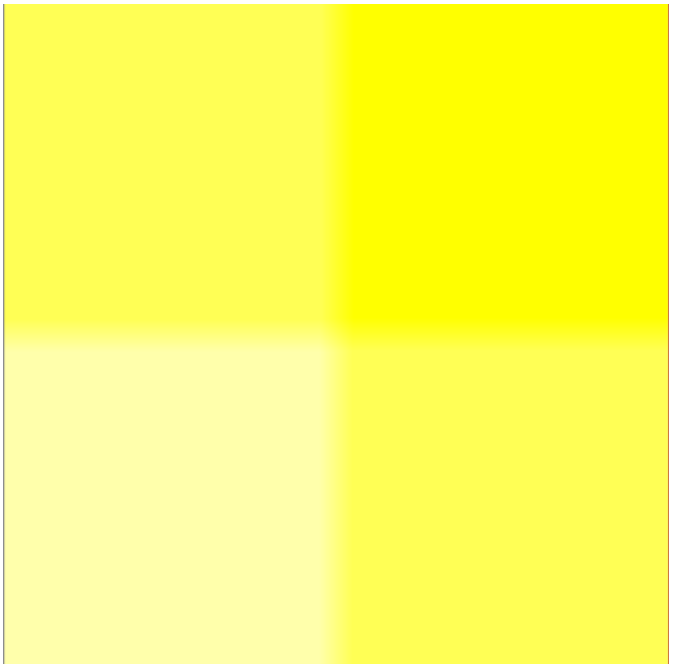
Utilizando o scripit1.3 para gera a imagem.



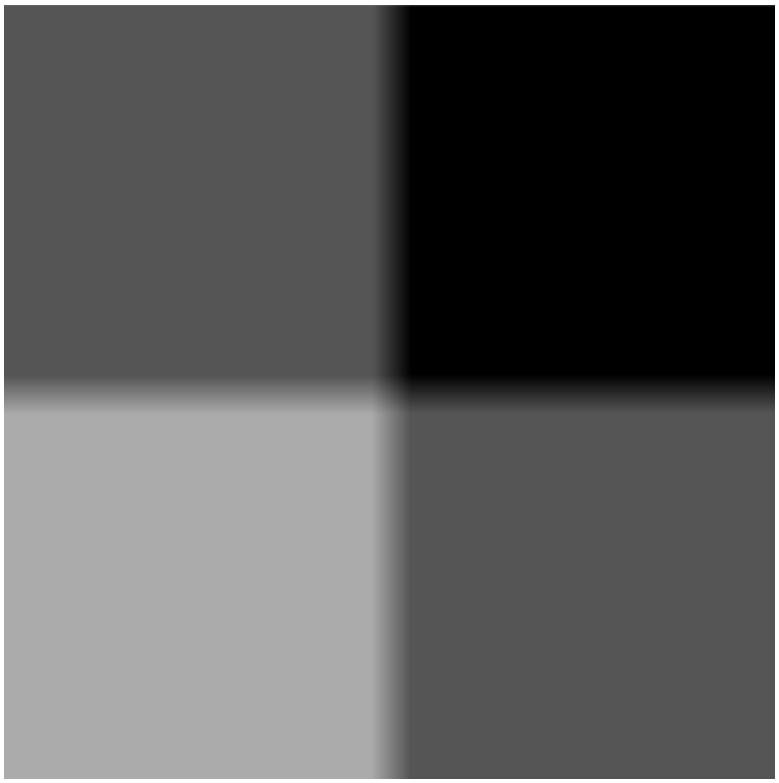
a) Suponha que convertamos a imagem de RGB para HSI. Em seguida, passamos um filtro da média aritmética de tamanho 25 x 25 na componente H da imagem e convertemos a imagem de volta para o RGB. Como a imagem resultante se parece?

Como informado anteriormente à biblioteca “opencv” não possui função para o sistema HSI, assim a imagem original foi convertida para um sistema similar, HSV que no lugar de Intensidade (I) utiliza a representação de valor (V) que define o brilho da cor.

A conversão de RGB para HSV é representada na imagem a seguir.



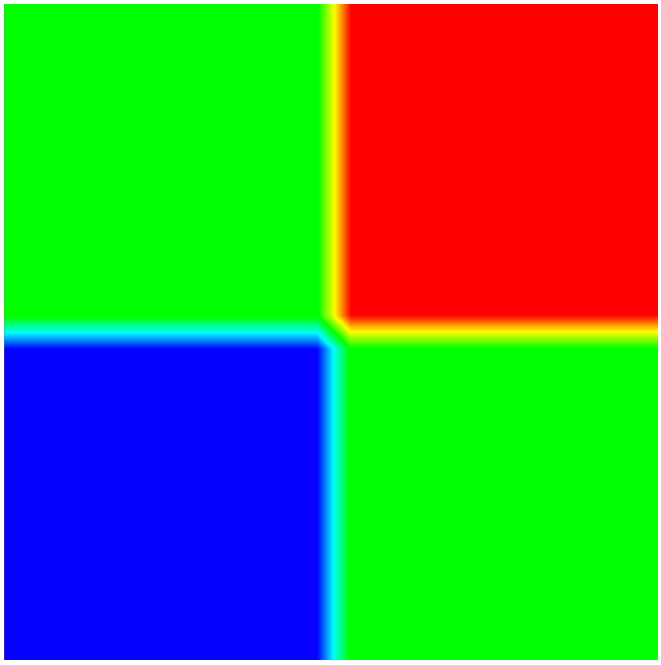
Resultado para a componente Tonalidade (H), explicado no exercício anterior:



Resultado para a componente H utilizando o filtro de média 25x25:

Este filtro faz a média na matriz da imagem entre 25 linhas e 25 colunas de pixel, tornando esse efeito borrado na imagem devido o cálculo de média entre os diferentes tons de cinza da imagem.

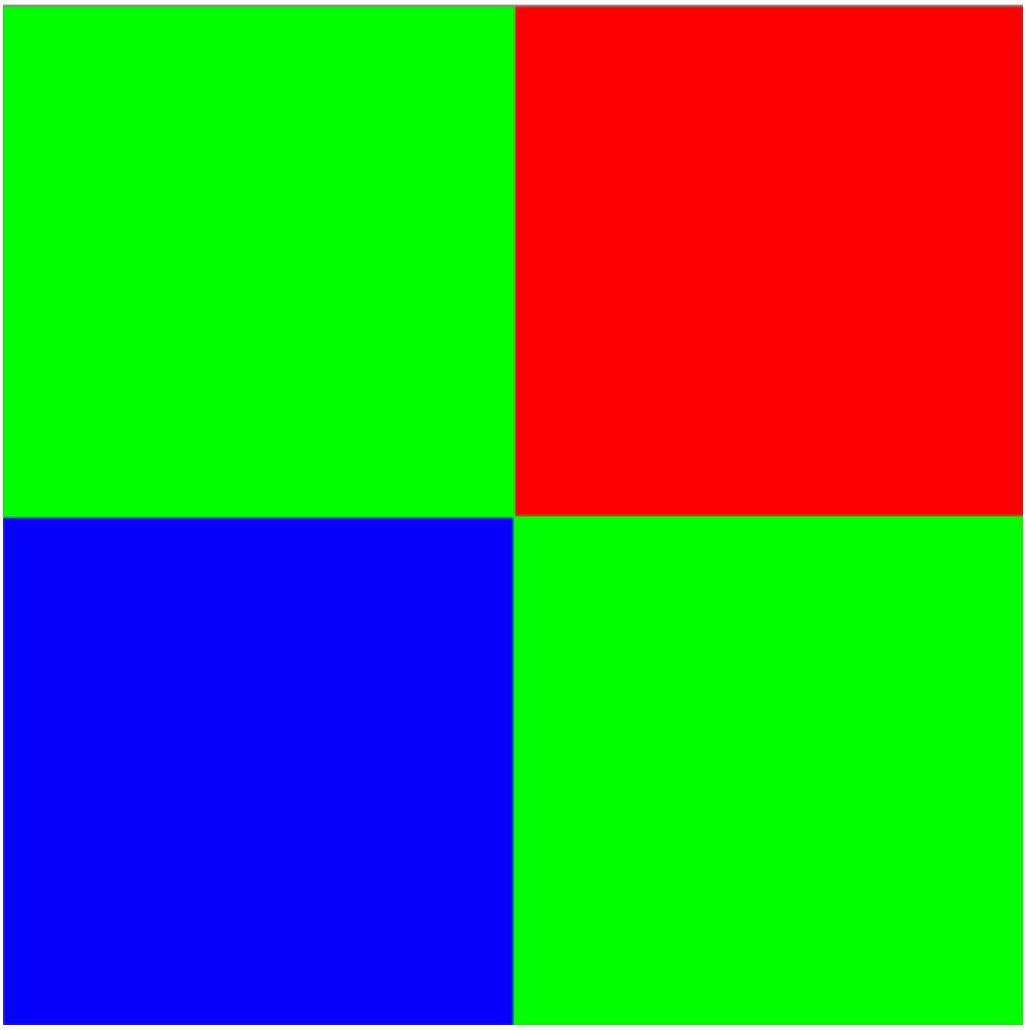
A imagem a seguir representa o resultado final do problema:



Nota-se que a mesma possui traços em volta dos quadrados, isso é devido a combinação com a componente H ilustrada anteriormente e também a sensação de das bordas borradas devido o filtro de média.

b) Repita o procedimento em (a) para a componente S. Como a imagem resultante se parece?

O componente S tem como resultado uma imagem toda branca, assim a média com a imagem que é uma matriz de zeros não impacta na imagem final representada a seguir:



Problema 1.4:

a) Calcule a entropia da fonte cujas probabilidades dos símbolos estão definidas na Tabela 8.1 do livro texto.

1.4. a) Calcule a entropia da fonte cujas probabilidades dos símbolos estão definidas na tabela 8.1 do livro texto.

$0,25 * \log_2 0,25 = -0,5$

$0,47 * \log_2 0,47 = -0,511$

$0,25 * \log_2 0,25 = -0,5$

$0,03 * \log_2 0,03 = -0,151$

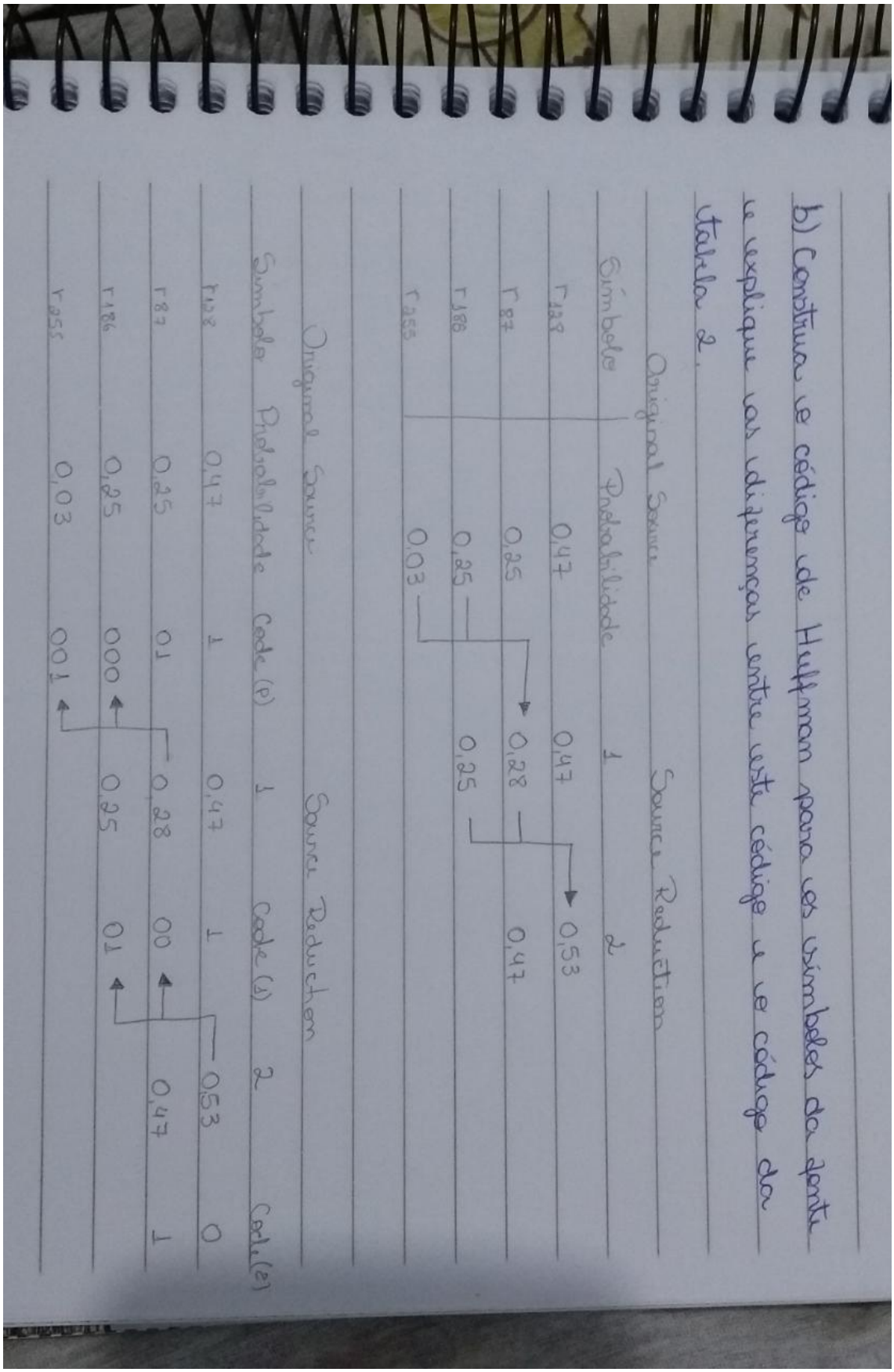
$H = -(-0,5 - 0,511 - 0,5 - 0,151)$

$H = -(-1,662)$

$\boxed{H = 1,662}$

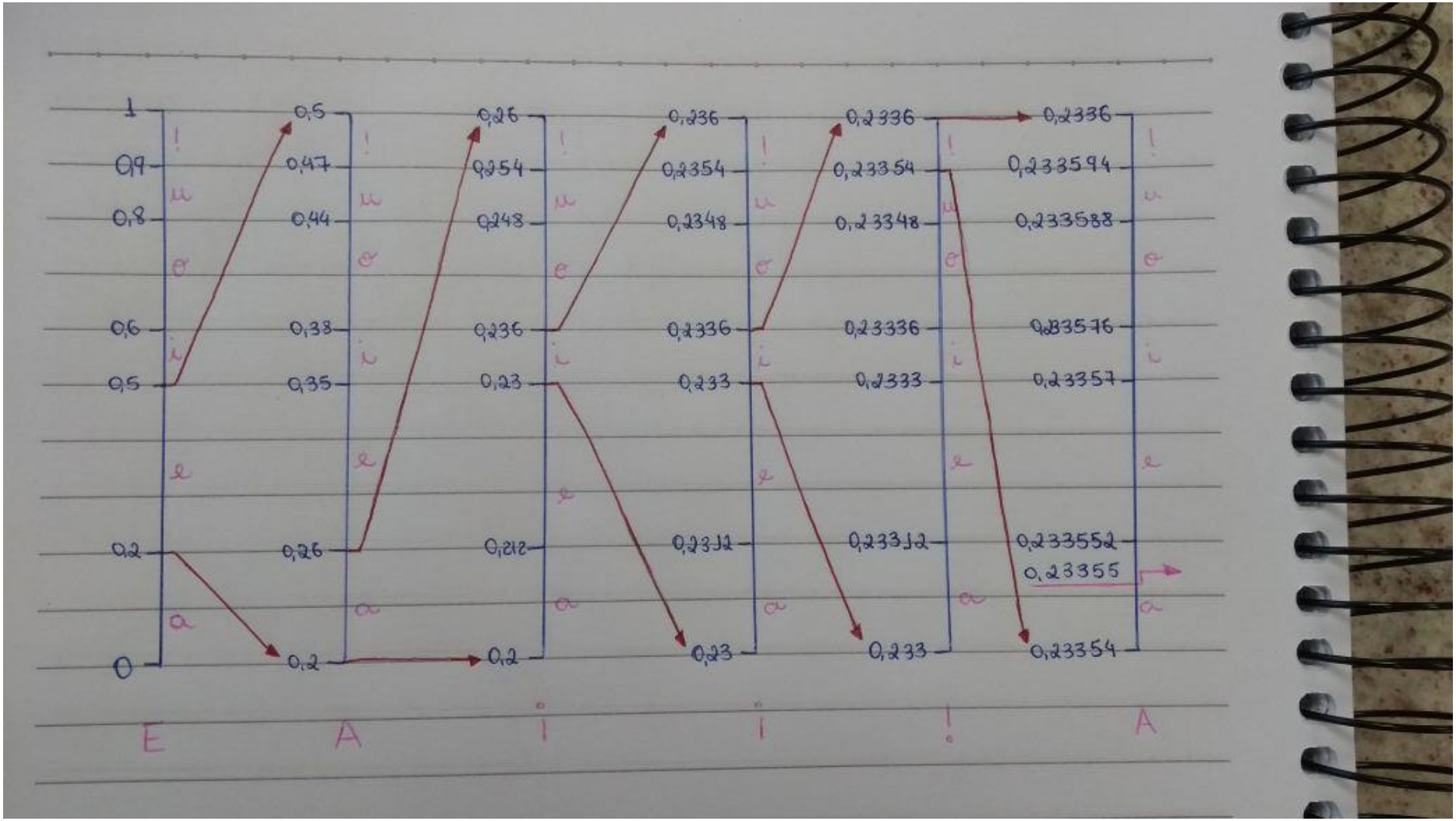
b) Construa o código de Huffman para os símbolos da fonte

b) Construa o código de Huffman para os símbolos da fonte e explique as diferenças entre este código e o código 2 da tabela.



Problema 1.5: Decodifique a mensagem 0,23355 codificada utilizando o código aritmético, cujos símbolos têm as probabilidades apresentadas na Figura abaixo.

Symbol	Probability
a	0.2
e	0.3
i	0.1
o	0.2
u	0.1
!	0.1


1
0,9
0,8
0,6
0,5
0,2
0
0,5
0,47
0,44
0,33
0,35
0,26
0,2
0,25
0,254
0,248
0,236
0,23
0,212
0,2
0,236
0,2354
0,2348
0,2336
0,233
0,2312
0,23
0,2336
0,23354
0,23348
0,23336
0,2333
0,23312
0,233
0,2336
0,233594
0,233588
0,233576
0,23357
0,233552
0,23355
0,23354
E
A
I
I
!
A

(a) Explique (em detalhes) o funcionamento do JPEG, desenhando o seu diagrama de blocos.

O padrão JPEG utiliza um algoritmo denominado LOCO-I (Low Complexity Lossless Compression for Images), especifica vários processos de compressão para imagens estáticas. A figura a seguir mostra um diagrama de blocos do processo JPEG básico.

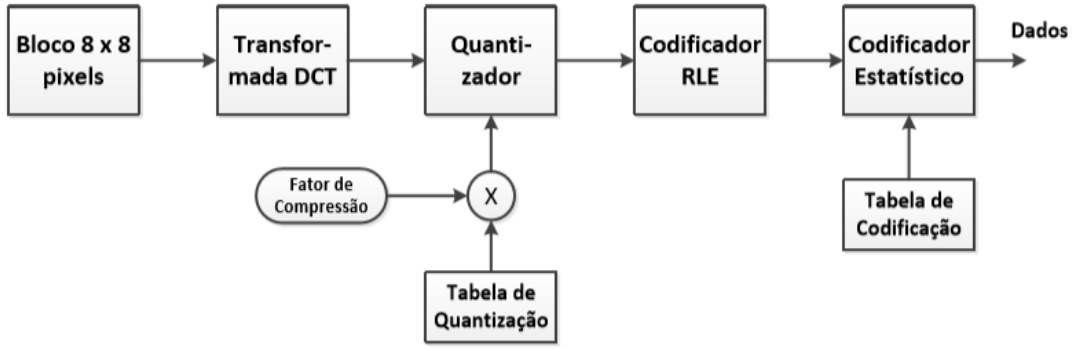


Figura 1.1 - Codificador JPEG

b) Explique (em detalhes) o funcionamento do MPEG, desenhando o seu diagrama de blocos.

O MPEG é utilizado para compressão de vídeo com áudio, sendo assim, vale considerar que um quadro difere muito pouco um do outro, e com isso pode ser usado na codificação de alguns deles. O MPEG trabalha com dois tipos de compressão. A primeira e baseada nas cores onde o algoritmo procura por agrupamento de pixels com a mesma cor e os substitui por um único código. Já a segunda etapa de compressão é um pouco mais complexa. A imagem é dividida em pequenos blocos de 16 x 16 pixels cada um, ao invés de simplesmente atualizar toda a imagem, a cada quadro são mudados os blocos que foram alterados de forma perceptível entre um quadro e outro.

Problema 2.1:

(a) Escreva um programa que converta uma imagem colorida em RGB para RGB de cores seguras.

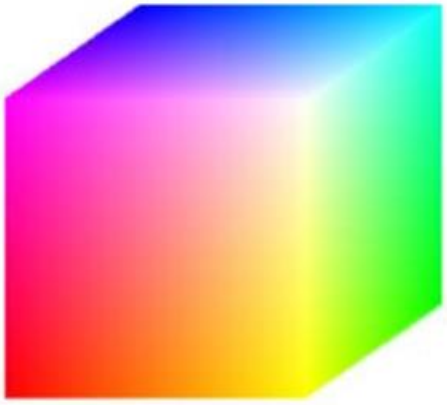
As cores segura são representadas pela tabela a seguir:

Sistema numérico	Equivalentes em cores					
Hexadecimal	00	33	66	99	CC	FF
Decimal	0	51	102	153	204	255

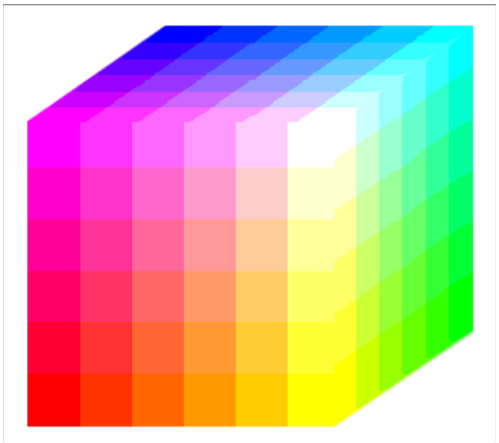
Segue o script2.1 para resolver a conversão:

```
import cv2
import numpy

import Utils

imagem = Utils.leImagem('fig_lista4_1.bmp')

dimensoes = numpy.shape(imagem)
linhas = dimensoes[0]
colunas = dimensoes[1]
dimensoes = dimensoes[2]

coresSeguras = [0, 51, 102, 153, 204, 255]


def trocaParaCorSegura(dimensao):
    pixel = imagem[linha, coluna, dimensao]
    if pixel >= 0 and pixel <= 43:
        pixel = coresSeguras[0]
    elif pixel >= 44 and pixel <= 86:
        pixel = coresSeguras[1]
    elif pixel >= 87 and pixel <= 129:
        pixel = coresSeguras[2]
    elif pixel >= 130 and pixel <= 172:
        pixel = coresSeguras[3]
    elif pixel >= 173 and pixel <= 215:
        pixel = coresSeguras[4]
    elif pixel >= 216 and pixel <= 255:
        pixel = coresSeguras[5]
    return pixel


for linha in range(linhas):
    for coluna in range(colunas):
        imagem[linha, coluna, 0] = trocaParaCorSegura(0)
        imagem[linha, coluna, 1] = trocaParaCorSegura(1)
        imagem[linha, coluna, 2] = trocaParaCorSegura(2)

Utils.mostraImagem(imagem)
```

(b) Baixe imagem abaixo e converta esta imagem para o formato RGB de cores seguras.



Resultado da imagem utilizando o script:



Problema 2.2:

a) Implemente um sistema semelhante ao apresentado na Figura 6.23. Neste sistema você pode especificar dois intervalos de valores de níveis de cinza para a imagem de entrada. Seu programa deverá gerar uma imagem RGB cujos pixels têm uma cor especificada correspondendo a um intervalo de níveis de cinza na imagem de entrada. O restante dos pixels na imagem RGB tem o mesmo valor de nível de cinza da imagem original. Você pode limitar as cores de entrada ao conjunto de cores apresentado conjunto de cores apresentado na Figura 6.4(a) do livro texto.

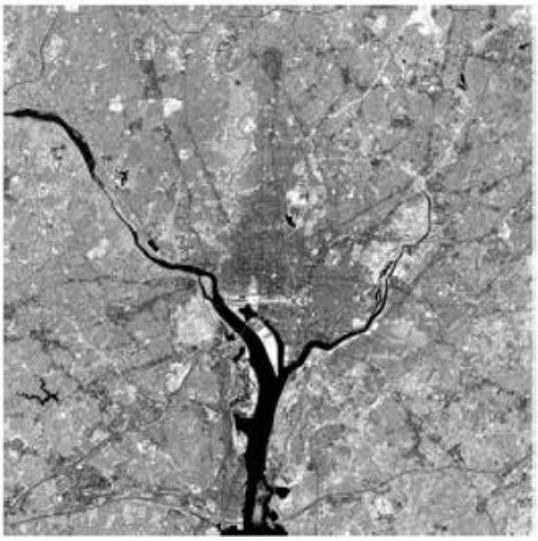
Para utilizar o Script 2.2, é necessário inserir os níveis de cinza que queria substituir pela cor desejada, exemplo: Primeiro nível de cinza = 0 e Segundo nível de cinza = 30.

Segue o script2.2:

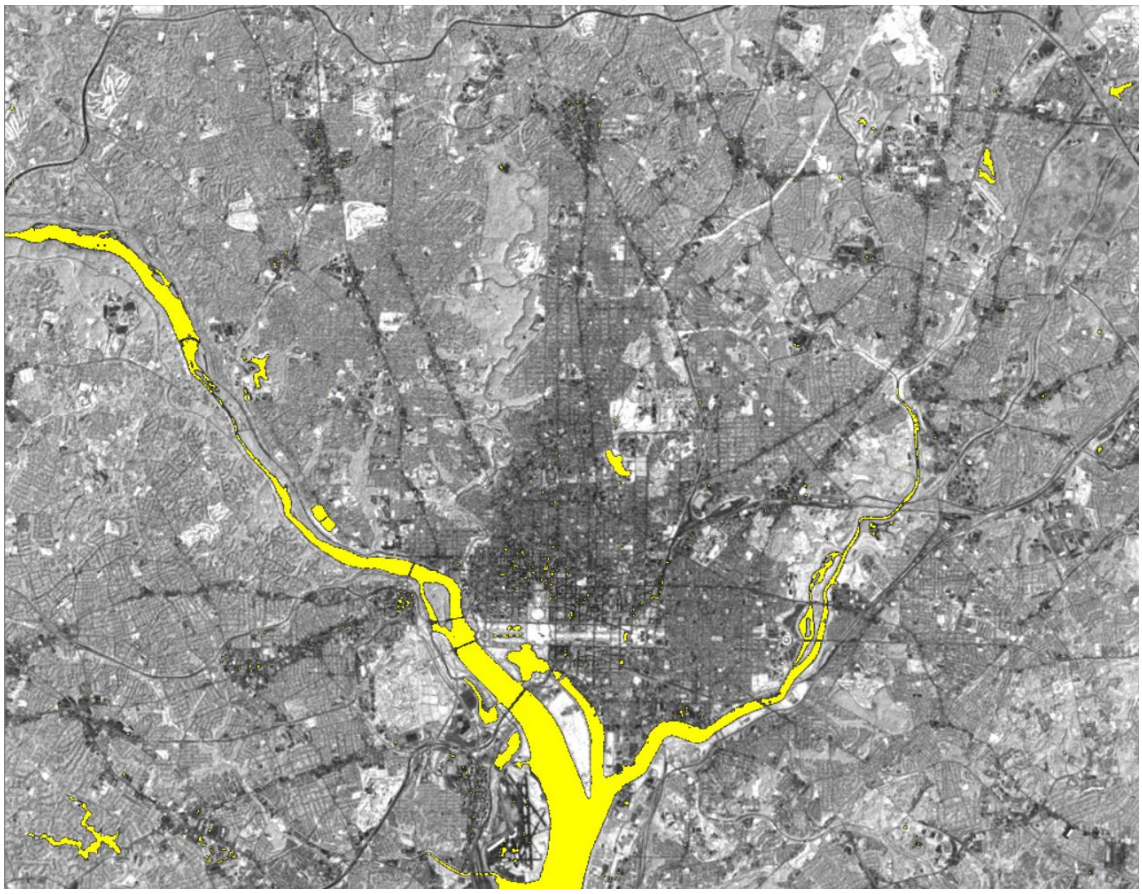
```
import numpy
import Utils
```

```
c1 = input("Primeiro nível de cinza: ")
c2 = input("Segundo nível de cinza: ")

imagem = Utils.leImagem("fig_lista4_2.bmp")
dimensoes = numpy.shape(imagem)

linhas = dimensoes[0]
colunas = dimensoes[1]
dimensoes = dimensoes[2]

for dimensao in range(dimensoes):
    for linha in range(linhas):
        for coluna in range(colunas):
            pixel = imagem[linha, coluna, dimensao]
            if pixel >= int(c1) and pixel <= int(c2):
                if (dimensao != 0):
                    imagem[linha, coluna, dimensao] = 255
                else:
                    imagem[linha, coluna, dimensao] = 0

Utils.mostraImagem(imagem)
```

b) Baixe a imagem abaixo (Figura 1.10 (4) do livro texto). Utilizando o programa desenvolvido em (a), processe a imagem de forma que o rio aparece amarelo e o restante da imagem conserve os tons de cinza originais. É aceitável ter alguns pontos isolados na imagem com cor amarela, mas este número deve ser baixo.



Resultado da imagem:



Apendice A – Utils

```
import cv2
import numpy


def criaImagem(linhas, colunas):
    imagem = numpy.ones((linhas, colunas), numpy.uint8)
    imagem = cv2.cvtColor(imagem, cv2.COLOR_GRAY2BGR)
    return imagem


def mostraImagem(imagem):
    cv2.imshow('imagem', imagem)
    cv2.waitKey(0)
    pass


def leImagem(imagem):
    path = "imagens/"
    return cv2.imread(path + imagem)
```

```
def getMonocromatico(blue, gree, red):
    monocromatico = (int(blue) + int(gree) + int(red)) / 3
    return monocromatico


def converteRGBParaHSV(imagem):
    return cv2.cvtColor(imagem, cv2.COLOR_BGR2HSV_FULL)


def converteHSVParaRGB(imagem):
    return cv2.cvtColor(imagem, cv2.COLOR_HSV2BGR_FULL)


def passaFiltroDeMedia(imagem, tamanho):
    return cv2.blur(imagem, (tamanho, tamanho))


def mostraListaDeImagens(list):
    posicao = 0
    for i in range(len(list)):
        cv2.imshow('Imagem ' + str(posicao), list[i])
        posicao += 1
    cv2.waitKey(0)
    pass
```