Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e Multimédia
Codificação de Sinais Multimédia

2º Semestre de 2019/2020

O ficheiro **Jupiter Notebook** com o relatório e código implementado deve ser submetido no Moodle até **26 de abril**. O nome do ficheiro deve ter o seguinte formato **"CSM_T41Gxx.ipynb"**, ou **"CSM_T42Gxx.ipynb"**, ou **"CSM_TNGxx.ipynb"** consoante a turma que estão a frequentar onde xx representa o número do grupo.

Este Trabalho explora os conceitos de compressão de dados sem perdas baseados na teoria da informação. Deve ter em consideração que as funções realizadas devem conter uma descrição, recomenda-se a utilização de células "markdown", para o efeito. No final deve apresentar uma tabela com todos os resultados extraídos.

1. Elabore uma função ("gen_huff_table") que gere uma tabela com o código binário para cada símbolo de um dado conjunto, usando o método de Huffman. Esta função deve ter como parâmetros de entrada um conjunto de símbolos e as suas probabilidades (ou em alternativa pode usar o número de ocorrências de cada símbolo, dado pelo seu histograma). Também pode em alternativa gerar não uma tabela mas outra estrutura de dados com os códigos pretendidos.

```
def gen_huff_table(symbol_list, symbol_prob):
    # code
    return symbol_code_table
```

2. Elabore uma função ("encode_huff") que dada uma mensagem (sequência de símbolos) e a tabela da ponto anterior, retorne uma sequência de bits com a mensagem codificada.

```
def encode_huff(message, symbol_code_table):
    # code
    return binary_codes
```

3. Elabore uma função ("decode_huff") que dada uma sequência de bits (mensagem codificada) e a tabela do ponto 1, retorne uma sequência de símbolos (mensagem descodificada). Garanta que a meensagem retornada por esta função é igual à mensagem que é dada como parâmetro de entrada da função "encode_huff".

```
def decode_huff(binary_codes, symbol_code_table):
    # code
    return message
```

4. Elabore uma função ("writeArray2File") que dada uma sequência de bits (mensagem codificada) e o nome do ficheiro, escreva a sequência de bits para o ficheiro.

```
def writeArray2File(filename, binary_codes, symbol_code_table):
    # code
```

5. Elabore uma função ("readFile2Array") que dado o nome do ficheiro, leia uma sequência de bits (mensagem codificada) contida no ficheiro.

```
def readFile2Array(filename):
    # code
    return binaryArray
```

6. Teste as funções elaboradas usando para o efeito a imagem fornecida ("LenaGray.tiff" em tons de cinzento).

   a) Gere o código usando a função realizada no ponto 1. Meça o tempo que demora a função.

   b) Meça a entropia e o número médio de bits por símbolo. Calcule a eficiência.

   c) Faça a codificação da mensagem contida no ficheiro (usando a função realizada no ponto 2). Meça o tempo que a função demora a fazer a codificação.

   d) Grave um ficheiro com a mensagem codificada, usando a função realizada no ponto 4. Veja o tamanho do ficheiro.

   e) Leia do ficheiro o conjunto de bits, usando a função realizada no ponto 5.

   f) Faça a descodificação da mensagem (usando a função realizada no ponto 3.) Meça o tempo que a função demora a fazer a descodificação.

   g) Compare a mensagem descodificada com a original e verifique que são iguais (erro nulo). MEça a taxa de compressão obtida.

Exemplo para o ficheiro com imagem

```
from time import time
from os import path
import cv2
import matplotlib.pyplot as plt

# Lê a imagem em níveis de cinzento
x = cv2.imread( "LenaGray.tiff", cv2.IMREAD_GRAYSCALE )

# Converte a imagem (matriz) numa sequência de números (array)
xi = x.ravel()

# Calcula o histogram
symbol_ocorr, symbols_list, patches = plt.hist(xi,256,[0,256])

# código alínea a) - Gera o código de Huffman
t0 = time()
symbol_code_table = gen_huff_table(symbol_list, symbol_ocorr):
t1 = time()
print "time:", t1-t0

# código alíneas b)

Entropy = # code
L = # code
Eficiency = # code

# código alíneas c) e d) - Codifica e grava ficheiro
binary_codes = encode_huff(xi, symbol_code_table)
writeArray2File(filename, binary_codes, symbol_code_table)
t2 = time()
print "time:", t2-t1

# código alíneas e) e f) Lê ficheiro e descodifica
binaryArray = readFile2Array(filename)
yi = decode_huff(binaryArray, symbol_code_table)
t3 = time()
print "time:", t3-t2

# código alíneas g)
```

```
erro = # code

size_ini = path.getsize("LenaGray.tiff")
size_end = path.getsize(filename)
print "taxa: ", 1.* size_ini / size_end

plt.show()
cv2.waitKey(0)
plt.close("all")
cv2.destroyAllWindows()
```