INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

ÁREA DEPARTAMENTAL DE ENGENHARIA DE ELECTRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES

Licenciatura em Engenharia Informática e Multimédia

Semestre 4

Codificação de Sinais Multimédia

TRABALHO Nº2

ALUNO:

Ricardo Azevedo, nº35667

2018-04-15

ÍNDICE

1 INTRODUÇÃO

Este trabalho tem como objetivo aprofundar os conhecimentos das aulas teóricas, ao aplicar em diversos ficheiros a codificação de Huffman.

O código tem como objetivos:

1. Gerar o código de Huffman com base nas probabilidades de cada símbolo;
2. Medir a entropia, o número médio de bits por símbolo e eficiência desse código;
3. Efetuar a codificação com base na tabela construída no ponto 1;
4. Gravar para um ficheiro, o conjunto de bits representativo da codificação do ficheiro inicial;
5. Ler esse conjunto de bits, escrito para ficheiro no ponto 4;
6. Fazer a descodificação, utilizando a tabela de Huffman gerada inicialmente;
7. Comparar o Input inicial, com o objeto de descodificação.

Para o desenvolvimento do código acima descrito foi utilizado o editor/compilador Spyder.

2 DESENVOLVIMENTO

2.1 Métodos/Funções

No código produzido foram criadas e utilizadas as seguintes funções por ordem no código:

- *cv2.imread* e *ravel* – Função para ler e decompor a image "lena.tiff" num array;
- *numpy.fromfile* – Função utilizada para ler os restantes ficheiros usados;
- *matplotlib.pyplot.hist* – Função utilizada para obtermos o número de ocorrências de cada símbolo;
- *concat* – Função gerada que vai atribuir uma probabilidade a cada símbolo com base no número de ocorrências do mesmo;
- *gera_huffman* e *tabCod* – Funções "principais" do código, pois é onde vai ser contruída a árvore/tabela de Huffman em que temos como base a lista de probabilidades retornada na função *concat*. O objetivo destas funções é atribuir o menor número de bits a símbolos com maior ocorrência e vice-versa;
- *codifica* – Função onde são substituídos os símbolos pelos bits correspondentes;
- *escrever* – Aqui é onde vai ser escrito para ficheiro a sequência de bits correspondente ao ficheiro inicial;
- *ler* – Função que lê um conjunto de bits escrito num ficheiro;
- *descodifica* – Com base nos bits lidos e na tabela gerada pelo código de Huffman vai descodificar o ficheiro atribuindo o correspondente símbolo à sequência de bits;
- *entropia* – Função que é usada para calcular a entropia;
- *bitsSimbolo* – Onde é calculado o número médio de bits por símbolo utilizado para cada símbolo;
- *eficiência* – Com base no número médio de bits por símbolo e da probabilidade de cada símbolo, é nesta função que é calculado a eficiência.

2.2 Resultados

3 CONCLUSÕES

4 ANEXOS

4.1 Código “huffmanCode.py”

```
# -*- coding: utf-8 -*-

from time import time
import cv2
import heapq as h
import matplotlib.pyplot as plt
import os.path as path
import numpy as np

dir = "Ficheiros/"
#_____________________________________
#file = "lena.tiff"
#file = "ecg.txt"
#file = "HenryMancini-PinkPanther.mid"
#file = "HenryMancini-PinkPanther30s.mp3"
#file = "ubuntu_server_guide.pdf"
file = "ubuntu_server_guide.txt"
#_____________________________________
codeFile = "huffman.bin"

tabela_codigo = []
cod_codif = []

def concat(y,leng):
    global cod_codif
    cod_codif = ['' for a in range(len(y))]
    oc = []
    for i in range(0,len(y)):
        prob = float(y[i])/leng
        if prob > 0:
            oc = oc+[(prob,i)]
    return oc
```

```
def gera_huffman(symbolProb):
    tree = list(symbolProb)
    h.heapify(tree)
    while(len(tree)>1):
        childR, childL = h.heappop(tree), h.heappop(tree)
        parent = (childL[0] + childR[0], childL, childR)
        h.heappush(tree, parent)
        h.heapify(tree)
    return tree[0]

def tabCod(tree, prefix = ''):
    global tabela_codigo

    if len(tree) == 2:
        tabela_codigo += [(tree[1],prefix)]
    else:
        tabCod(tree[1], prefix + '0')
        tabCod(tree[2], prefix + '1')

def codifica(data, tabela):
    seqBits = ''
    for i in range(0,len(data)):
        for j in range(0, len(tabela)):
            if data[i] == tabela[j][0]:
                seqBits += tabela[j][1]
    return seqBits

def descodifica(seqBits,tabela):
    data = []
    idx_seq = 0
    for i in range(1,len(seqBits)+1):
        for j in range(0,len(tabela)):
            if seqBits[idx_seq:i] == tabela[j][1]:
                data.append(tabela[j][0])
                idx_seq = i
    data = np.asarray(data,'uint8')
    return data
```

```
def splitStringByLength(string, length):
    subStrings = []
    for i in range(len(string)/length+1):
        subStrings.append(string[8*i:8*(i+1)])
    if len(subStrings[-1]) == 0:
        subStrings = subStrings[:-1]
    return subStrings

def escrever(seqBits, f):
    #escrever rw
    fileToWrite = open(f, 'wb')
    fileToWrite.write(seqBits)
    fileToWrite.close()

def ler(f):
    #read rb
    dataBin = open(f,"rb").read()
    return dataBin

def entropia(oc):
    Hs = 0
    for i in range(len(oc)):
        Hs = Hs + (oc[i][0] * np.log2(1.0/oc[i][0]))
    return Hs

def bitsSimbolo(tabela,oc):
    l = 0.0
    for i in range(len(tabela)):
        l = l + (len(tabela[i][1]) * oc[i][0])
    return l

def eficiencia(l,oc):
    L = 0
    for i in range(len(oc)):
        L = L + (oc[i][0] * l)
    return L

if __name__ == '__main__':
    # Imagem Lena
```

```
#x = cv2.imread(dir+file,cv2.CV_LOAD_IMAGE_GRAYSCALE)
#xi = x.ravel()
#_____________________________________________________________
# Outros ficheiros
xi = np.fromfile(dir+file,dtype=np.uint8)

y, bins, patches = plt.hist(xi,256,[0,255])

oc = concat(y,len(xi))
#print oc

t0 = time()
huffTree = gera_huffman(oc)
#print huffTree

tabCod(huffTree)
#print 'tabela_codigo',tabela_codigo

t1 = time()

seqBits = codifica(xi,tabela_codigo)
#print 'seq_bits',seqBits

escrever(seqBits,codeFile)

t2 = time()

receivedSeq = ler(codeFile)
#print 'recebido: ',receivedSeq
#print (seqBits == receivedSeq)

yi = descodifica(seqBits, tabela_codigo)

t3 = time()

size_ini = path.getsize(dir+file)
size_end = path.getsize(codeFile)

Hs = entropia(oc)
```

```
l = bitsSimbolo(tabela_codigo,oc)
L = eficiencia(l,oc)

print 'Tempo até gerar código de Huffman:',round(t1-t0,4)
print 'Entropia =',round(Hs,3)
print 'Número Médio Bits por Símbolo =',round(l,3)
print 'Eficiência =', round((Hs/L),3)
print 'Tempo de codificação =',round(t2-t1,4)
print 'Tamanho do ficheiro com mensagem codificada =',size_end
print 'Tempo de descodificação =',round(t3-t2,4)
print 'Input = Output?',(xi==yi).all()
print "Taxa de Compressão =",round((1.* size_ini / size_end),3)

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