

# Lab work nº 3

## Relatório



universidade  
de aveiro

Teoria Algorítmica da Informação

Gil Teixeira

João Relva

Vera Oliveira

Janeiro 2022

## Índice

<b>1</b>	<b>Introdução .....</b>	<b>3</b>
<b>2</b>	<b>Desenvolvimento .....</b>	<b>4</b>
2.1	Design .....	4
2.2	charizam.py .....	5
2.3	accuracy.py .....	5
<b>3</b>	<b>Resultados .....</b>	<b>6</b>
<b>4</b>	<b>Conclusões .....</b>	<b>7</b>
<b>5</b>	<b>Project Repository .....</b>	<b>8</b>

# 1 Introdução

Neste segundo trabalho prático da unidade curricular de Teoria Algorítmica da Informação, os alunos tinham como objetivo desenvolver um programa cujo objetivo seria a identificação de músicas com base num pequeno segmento (*sample*) de uma música, por exemplo, 10 segundos.

O princípio base para o desenvolvimento deste programa seria calcular a NCD - *Normalized Compression Distance* - entre todas as representações das músicas presentes na base de dados e o *sample* a analisar. A música identificada estaria associada ao menor valor resultante do cálculo da NCD. Para realizar a compressão foram utilizados os seguintes compressores: *gzip*, *bzip2* e *lzma*. A base de dados de músicas e os *samples* contêm 50 ficheiros *.wav* para realizar os testes à aplicação. Existem samples com vários tempos de duração (5, 12 e 30 segundos) e também com ruído, de maneira a testar a robustez do programa.

O programa *GetMaxFreqs* foi nos previamente fornecido e tem a função de identificar as frequências mais significativas de um ficheiro *.wav* e de seguida criar um ficheiro *.freqs* que permite representar o áudio (assinatura).

Foi desenvolvido também um programa para realizar a classificação da aplicação calculando a *accuracy*.

Este relatório contém a explicação do código desenvolvido, os vários resultados obtidos e as conclusões finais.

O trabalho desenvolvido foi disponibilizado num repositório no GitHub.

## 2 Desenvolvimento

### 2.1 Design

Para o desenvolvimento deste trabalho continuámos a utilizar a linguagem de programação usada nos trabalhos anteriores (Python) e todo o código foi implementado e testado em Linux.

Considerando todo o contexto do trabalho e as tarefas que deveriam ser implementadas, foram criados dois ficheiros *.py*: *charizam.py*, *accuracy.py*. O *charizam.py* permite a identificação de músicas com base num *sample* que é passado como argumento do programa. Existem 3 compressores que podem ser utilizados durante a execução do programa para o cálculo da NCD que permitirá a identificação das músicas, pelo que o tipo de compressor a utilizar também deverá ser passado como argumento do programa:

- -g : gzip
- -b : bzip2
- -l : lzma

O Projeto contém 7 pastas:

1. samples - 50 ficheiros *.wav* com duração de 12 segundos que representam um segmento de uma música
2. 05sSmpl - 50 ficheiros *.wav* com duração de 5 segundos que representam um segmento de uma música
3. 30sSmpl - 50 ficheiros *.wav* com duração de 30 segundos que representam um segmento de uma música
4. noisySp - 50 ficheiros *.wav* com presença de ruído que representam um segmento de uma música
5. database - 50 músicas diferentes (*.wav*)
6. DbFreqs - Ficheiros *.freqs* que representam todas as músicas da base de dados (database)
7. GetMaxFreqs - Contém o programa que permite criar os ficheiros *.freqs*

Este programa permite também a utilização de uma *flag* “-u”, que permite atualizar a pasta DbFreqs.

O programa *accuracy.py* permite realizar a classificação da aplicação calculando a *accuracy*. O programa recebe como argumento o nome da pasta referente ao tipo de *samples* que serão utilizados no cálculo da *accuracy* e o tipo de compressor a utilizar.

Existe ainda um ficheiro excel (samples.xlsx) que contém a associação dos *samples* às músicas a que pertencem.

As instruções para a execução dos vários programas estão presentes no ficheiro “*README*” do repositório do projeto (link na secção “*Project Repository*”).

## 2.2 charizam.py

Este programa recebe como *inputs* obrigatórios um *sample* e o tipo de compressor a utilizar no cálculo da NCD. A classe “CHARIZAM” é responsável pelo funcionamento do programa, classe essa que é usada também no *accuracy.py*. A aplicação utiliza a função *createFreqsFile()* que em conjunto com o programa *GetMaxFreqs* permite a criação de todos os ficheiros *.freqs* para as músicas da base de dados. Esses ficheiros são agrupados e movidos para uma pasta que é criada com o nome “*DbFreqs*” (*createFreqsFolder()*, *moveToFreqsFolder()*). Ao executar o programa, se a pasta *DbFreqs* já existir não serão criados novos ficheiros *.freqs* para as músicas, uma vez que os mesmos já se encontram disponíveis (*isDbFreqs()*). No entanto é possível passar como argumento uma *flag* (-u) que permite atualizar esta pasta criando novos ficheiros (*updateDbFreqs()*). De seguida é calculada a NCD entre cada música existente e o *sample* que se pretende analisar (*ncd()*, *createDistances()*), utilizando a seguinte fórmula:

$$\text{NCD}(x, y) = \frac{C(x, y) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}},$$

A distância de similaridade que é retornada após esse cálculo é adicionada a um dicionário (*distances*) e associada à música correspondente. A música a que o segmento pertence será aquela que tiver um menor valor de distância de similaridade associado. Durante a execução do programa são criados alguns ficheiros temporários que são eliminados quando o mesmo termina (*deleteFiles()*).

## 2.3 accuracy.py

Este programa recebe como *inputs* a pasta pertencente aos segmentos que serão analisados e ao compressor que será utilizado. Esta aplicação utiliza o programa anterior para identificar a que músicas pertencem todos os *samples* existentes na base de dados. É feita a comparação do *sample* com a música que foi identificada utilizando a função *classification()* que contém um dicionário com as associações corretas de cada *sample* a cada música. É guardado na variável *correct\_guesses* o número de vezes que foram identificadas as músicas corretas e de seguida calculada a *accuracy* através da seguinte fórmula:

```
accuracy = (correct_guesses / len(samples)) * 100
```

### 3 Resultados

Todos os testes foram realizados num *laptop* com um processador Intel Core i7-4558U e 8GB de RAM dentro de uma máquina virtual com o sistema operativo Kali Linux 5.14.16.

Os seguintes resultados foram obtidos através da utilização do programa *accuracy.py*. Foi analisado o tempo de execução de cada compressor na identificação de todos os *samples* e foi também analisado a *accuracy* de cada compressor, isto para cada diferente tipo de *sample*.

**Tempos de execução por compressor**

Pasta de Samples/Compressor	gzip	bzip2	lzma
<b>samples</b>	0:22.40	0:37.53	01:27.39
<b>05sSmpl</b>	0:18.70	0:31.31	01:36.46
<b>30sSmpl</b>	0:29.60	0:42.41	01:49.80
<b>noisySp</b>	0:20.59	0:32.65	01:33.98

**Accuracy por compressor**

Pasta de Samples/Compressor	gzip	bzip2	lzma
<b>samples</b>	100%	100%	66%
<b>05sSmpl</b>	100%	100%	12%
<b>30sSmpl</b>	100%	100%	98%
<b>noisySp</b>	100%	100%	52%

As seguintes imagens são exemplos de *outputs* do dois programas implementados:

```

$ python3 charizam.py samples/sample37.wav -g
Analyzing the sample...

-----GUESSED SONG-----
ProfJam - Agua de Coco

can object (guessed comp)

Execution time: 0:00:00.393258

```

```

$ python3 accuracy.py noisySp -b
Calculating Accuracy...

Sample Folder: noisySp
Compressor: bzip2

Accuracy: 100.0%

Execution time: 0:00:32.650067

```

## 4 Conclusões

Após a análise dos resultados foram retiradas as seguintes conclusões:

- O compressor *gzip* é o mais rápido enquanto o *lzma* é o compressor mais lento.
- Tanto o compressor *gzip* como o *bzip2* obtiveram uma *accuracy* de 100% para todos os tipos de *samples* (com várias durações e com ruído)
- O *lzma* foi o compressor com pior performance. Este tipo de compressor demonstrou ser bastante sensível ao ruído e à duração de tempo dos vários segmentos. Para segmentos de 12 segundos obteve uma *accuracy* de 66%, mas para segmentos com pouco segundos a *accuracy* desceu bastante. No aumento da duração dos *samples* já se verificou uma performance bastante melhor. Comparando os *samples* com 12 segundos com os *samples* com ruído (que têm a mesma duração de tempo – 12 segundos) verificou-se que a *accuracy* baixou ligeiramente.

Concluimos que para este tipo de soluções o *gzip* seja o melhor compressor. Apesar de o *bzip2* conseguir também identificar todos os segmentos, o *gzip* acaba por se tornar um pouco mais rápido.

## 5 Project Repository

- GitHub Repository: <https://github.com/bearkillerPT/TAI>