# Projeto de MPEI - Relatório

Universidade de Aveiro - Métodos Probabilísticos para Engenharia Informática

93289 - Alexandre Oliveira, 95228 - Lara Matos



#### Resumo

No âmbito da unidade curricular de Métodos Probabilísticos para Engenharia Informática foi proposto aos alunos a realização de um projeto cujo objetivo principal fosse a criação de uma aplicação que utilizasse algumas matérias lecionadas durante as aulas teórico-práticas da referida disciplina. Para isso, este grupo escolheu o problema 9 (desenvolvimento de uma aplicação para uma lista de letras de músicas) com as seguintes finalidades:

- Determinação das letras das músicas que contêm uma palavra específica ou um conjunto de palavras;
- Qual a letra mais similar a uma determinada letra;
- Quais os pares de letras que têm uma similaridade superior a um determinado valor.

Para o bom funcionamento da aplicação, é recomendável o uso de algoritmos MinHash, para a demonstração de um valor aproximado da similaridade de Jaccard.

Neste relatório, serão abordados items relacionados com o trabalho, tais como a maneira correta de correr os programas, os módulos implementados na aplicação, os testes efetuados e os resultados desses mesmos testes.

# Como correr os programas

Para a execução correta da aplicação deve-se fazer os seguintes passos:

- Num terminal, abrir o diretório Project\_MPEI\_93289\_95228 e em seguida o diretório src;
- Compilar a aplicação LetrasMusicais: javac LetrasMusicais.java;
- Executar a aplicação LetrasMusicais: java LetrasMusicais.

Esta app tem como função ler o ficheiro lyrics.txt constituído por 302 letras de músicas, tendo como ficheiro principal (main) o programa LetrasMusicais.java com a apresentação de um menu, e vai fazer os seguintes parâmetros:

- Leitura do ficheiro linha a linha;
- Remoção dos caracteres especiais;
- Adição dos títulos das músicas a uma hashtable pertencente à app;
- Listar as letras das músicas com palavras ou sequências de palavras iguais, com o auxílio do *BloomFilter*;
- Usando *MinHash*, a aplicação vai apresentar qual a letra de uma música que é mais similar a outra;
- Com a utilização de MinHash novamente, a app demonstrará os pares de músicas mais semelhantes.

Dá para escolher o número de *Strings* que se pretende usar, mas, infelizmente, quando pedidas demasiadas *Strings*, ocorre um erro no programa denominado por: *java.lang.OutOfMemoryError Java heap space*.

# Módulos implementados

O *Bloom Filter* é uma estrutura de dados que tem como funcionalidade a apresentação da solução probabilística rápida acerca da pertença de um elemento a um determinado conjunto ou não através da utilização de *Hash Functions*.

Neste caso em particular, desenvolveu-se o módulo BloomFilter com três funções hash:

- string2hash: gera um valor hash para cada String;
- insert: insere um elemento no filtro;
- isMember: devolve um valor boolean caso o elemento pertença ou não ao conjunto.

Desenvolveu-se igualmente um programa Shingle que comprime cada letra numa String e um módulo MinHash que, tal como explicado no capítulo anterior, apresentará a similaridade entre as letras das músicas, tendo como base o índice de Jaccard e o cálculo da respetiva distância de Jaccard, respetivamente são as funções jaccardSimilarity e jaccardDistance, e a criação de Sets com a função getHashSet.

### Testes efetuados e resultados

Realizaram-se os seguintes testes:

- Bloom Filter: testou-se o módulo *TestFilter* relativamente à criação, inserção e pertença de cada *String* a um determinado conjunto. Os testes foram todos realizados com sucesso, tendo como estudo um conjunto de nomes próprios;
- MinHash: testes ao módulo *TestMinHash* bem-sucedidos, com o cálculo da similaridade entre 5 *Strings* e um outro teste com *Strings random* em que comparam o valor da similaridade relativamente ao valor anterior.

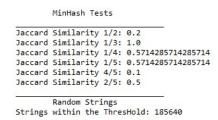


Figura 4.1: Teste - MinHash

```
false

Check Array words before insert: Should be False

Bloom Filter

true
true
true
true
false
false
false
false
false
true
false
false
false
true
false
true
false
```

Figura 4.2: Teste - Bloom Filter

Figura 4.3: Teste -  $Bloom\ Filter$ 

```
Tests
Test [igmhl]: false
Test [prpwd]: false
Test [midpp]: false
Test [hpsql]: true
Test [czkvo]: false
Test [cffmf]: true
Test [cguzl]: false
Test [hqsxy]: false
Test [klbwh]: false
Test [svccj]: false
Test [kgenf]: false
Test [ohumc]: false
Test [dnbjx]: false
Test [hwcbe]: false
Test [irxkx]: false
Test [dxjtj]: false
Test [wfnur]: false
Test [flkup]: true
Test [mlgwl]: false
Test [xcvpx]: false
```

Figura 4.4: Teste -  $Bloom\ Filter$