

# Departamento de Eletrónica, Telecomunicações e Informática

**Métodos Probabilísticos para**

**Engenharia Informática**

Ano Letivo 2018/2019

2º Ano, 1º semestre

Turma P5

Alexey Kononov, nº mec: 89227  
(alexeykononov@ua.pt)

Rodrigo Santos, nº mec: 89180  
([rodrigo.l.silva.santos@ua.pt](mailto:rodrigo.l.silva.santos@ua.pt))

Introdução:

O objetivo deste projeto é desenvolver um conjunto de blocos que desempenhem as funções de contador estocástico, Counting Bloom Filter e um processo de deteção de similaridade entre conjuntos utilizando o conceito de MinHash, em *software*, e testar esses blocos tanto separadamente como em conjunto num programa de teste final.

Contador Estocástico:

A nossa implementação do contador estocástico (ContadorEstocastico.java), conta com probabilidade estabelecida pelo utilizador (countProb) o número estimado de elementos de um conjunto, sendo esse conjunto acedido através do seu *Path* ou String com o valor do Path, passado como argumento ao contador no momento da sua inicialização.

Para a correta utilização do teste é necessário abrir o programa (TestContadorEstoc.java) e colocar as localizações exatas dos ficheiros phone\_dataset.csv e pg26017.txt onde solicitado.

O teste a este elemento é realizado através da contagem parcial, do número de linhas que certo ficheiro possui. O número total de linhas (FinalCount) será depois estimado através da multiplicação do número de linhas contadas (counter) pelo inverso de countProb. No teste é também possível verificar a diferença entre o número de linhas real e o número estimado calculando assim o erro. Esta diferença deveria diminuir á medida que o valor de countProb aumentasse, porém pode não se verificar dado que é um método probabilístico.

Counting Bloom Filter:

O Counting Bloom Filter que desenvolvemos (CountingBloomFilter.java) recebe como argumentos a quando da sua inicialização um inteiro “m”, número de elementos esperados, e um double “p”, que representa a probabilidade de Falsos Positivos (PfalsPos) do Bloom Filter. A dimensão do Bloom Filter é calculada de seguida com recurso a estes argumentos através da formula da probabilidade desejada de Falsos Positivos, substituindo o valor ótimo de K (número de funções de Hash) por (n\*ln(2)/m). De seguida é calculado o valor de K. Para a inserção dos elementos no Bloom Filter é calculado o valor de Hash para os elementos e o contador desse elemento é incrementado em 1.

Para a correta utilização do teste é necessário abrir o programa (TestBloomFilter.java) e colocar a localização exata do ficheiro phone\_dataset.csv onde solicitado.

Os testes realizados a este bloco (TestBloomFilter.java) foram realizados com a base de dados que utilizamos para o programa conjunto, são colocadas as marcas dos telemóveis no Bloom Filter e verificamos a pertença de algumas marcas á base de dados utilizada bem como o número de ocorrências dessa marca no Bloom Filter.

Deteção de similaridade (MinHash):

Neste bloco (SimilarityFinder.java) verificamos a similaridade de 2 conjuntos, com recurso a MinHash. Este bloco é inicializado com o número de Funções de Hash (numHash) adequado para resultados aceitáveis. No processo de procura de similaridade entre os conjuntos é obtido um HashMap com informação sobre a pertença dos shingles aos conjuntos, são obtidas as assinaturas dos conjuntos e com as assinaturas é calculada a similaridade somando o número de MinHash iguais e dividindo pelo número de Funções de Hash.

Para o teste realizado a este bloco (TestSimilarityFinder.java), comparámos 2 conjuntos com especificações de telemóveis diferentes e de seguida comparámos um dos conjuntos com ele próprio, os resultados da similaridade entre os conjuntos bem como o número de Funções de Hash utilizadas são impressos no terminal.

Bloco de Construção de Shingles:

O módulo Shingling.java tem uma função estática getShingle() que recebe 2 parâmetros, uma String e um inteiro e retorna um array com os Shingles. A dimensão de cada shingle é determinada pelo parâmetro do tipo inteiro. Para o correto funcionamento o inteiro tem de ser menor do que o número de caracteres da String.

Para testar, fizemos o programa de teste TesteShingle.java, que dada uma String, obtém shingles da mesma, cada um de 5 caracteres. Na aplicação conjunta uma possível aplicação era obter shingles de cada modelo de telemóvel para introduzir nos Sets que são comparados. Após realização de vários testes, verificamos que os resultados obtidos sem uso deste bloco foram significativamente melhores, o que determinou a não utilização deste módulo no programa conjunto.

Programa de Teste Final:

Para a correta utilização do teste é necessário abrir o programa (TesteGlobal.java) e colocar a localização exata do ficheiro phone\_dataset.csv onde solicitado.

Na nossa aplicação conjunta (TesteGlobal.java) desenvolvemos um programa que conta o número de elementos esperados a serem inseridos no Bloom Filter que no caso da aplicação é o número de linhas (telemóveis) que existem na base de dados (phone\_dataset.csv). O Counting Bloom Filter desenvolvido é inicializado com probabilidade de Falsos Positivos de 0.05 e servirá de armazenamento para as especificações dos telemóveis, como não há telemóveis iguais na base de dados a função de Contador do Bloom Filter não é utilizada na medida em que todos os elementos terão valor 1. O utilizador terá de 2 opções:

1. Introduzir apenas um Modelo de telemóvel para verificar a sua pertença á base de dados.
2. Introduzir um conjunto de especificações separadas por vírgulas para procurar na base de dados, utilizando o bloco SimilarityFinder, telemóveis com uma similaridade superior ao limiar de 0.6.

Dependendo do *input* introduzido o programa imprimirá, para o caso 1) a pertença do modelo introduzido á base de dados e para o caso 2) uma lista de telemóveis similares.

A aplicação continuará a correr enquanto o utilizador não inserir “exit”.