Trabalho Prático

Métodos Probabilísticos para Engenharia Informática
DETI - Universidade de Aveiro

Rodrigo Rosmaninho, André Alves - Turma P5
(88802) r.rosmaninho@ua.pt, (88811) andr.alves@ua.pt

Dezembro de 2018

Relatório Final

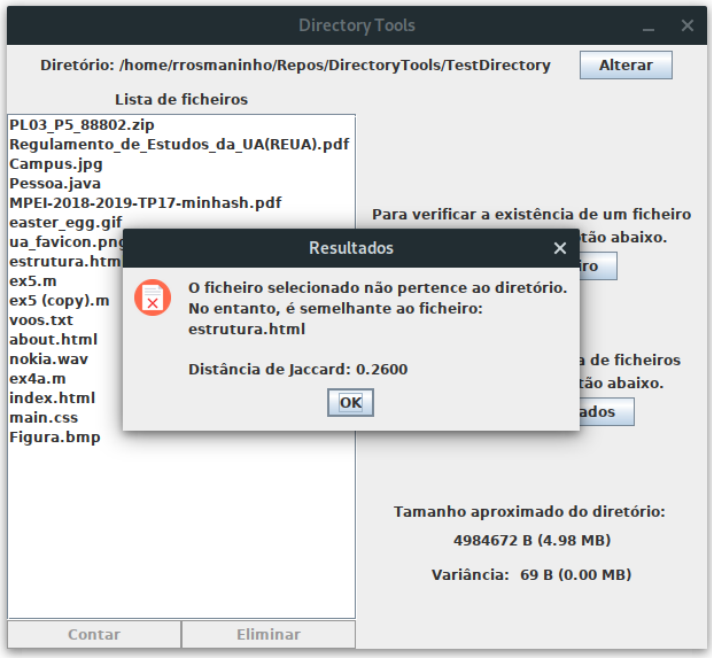
Capítulo 1

Módulos e Aplicação de Uso Conjunto

1.1 Aplicação 'Directory Tools'

Para demonstrar os módulos a trabalharem em conjunto, foi desenvolvida uma aplicação com interface visual (Java Swing) e que permite:

- Escolher um diretório no sistema cujos ficheiros vão ser lidos e adicionados ao Counting Bloom Filter após terem sido geradas as assinaturas de cada um (através do algoritmo de MinHash).
- Obter a soma de todos os bytes lidos (ou seja, o tamanho do diretório lido) através de contagem estocástica.
- Escolher um ficheiro e verificar se existe uma cópia deste no diretório que foi lido, mesmo que os nomes sejam diferentes (verificar presença no bloom filter). Mesmo que não exista é indicado, quando possível, qual o ficheiro do diretório lido é mais semelhante ao ficheiro escolhido (Distância de Jaccard com Locality Sensitive Hashing).
- Detetar e apresentar eventuais ficheiros duplicados no diretório lido, mesmo que tenham nomes diferentes (Distância de Jaccard com Locality Sensitive Hashing).
- Visualizar todos os ficheiros lidos, em forma de lista.
- Eliminar ficheiros do Counting Bloom Filter, para que estes deixem de ser contemplados em operações subsequentes ou exibidos na lista de ficheiros. (função remove())
- Contar o número de vezes que um determinado ficheiro foi adicionado ao Counting Bloom Filter. (função count())



Recomedam-se diretórios com tamanho não superior a 20MB de forma a minimizar o tempo de espera e a utilização de memória ao ser efetuada a leitura, adição ao Bloom Filter, e cálculo das assinaturas.

Para iniciar este programa, basta executar o ficheiro **DirectoryTools.jar** presente na raíz do projeto submetido, clicando neste duas vezes. É, no entanto, necessário instalar no mínimo a versão 8 do Java (JRE).

Exite também um diretório de demonstração, **TestDirectory**, com vários ficheiros de tipos diferentes e um outro diretório, **TestFiles** com alguns ficheiros semelhantes aos do TestDirectory, para que se possam testar as funcionalidades da aplicação.

1.2 Módulos

1.2.1 Counting Bloom Filter

A implementação do Counting Bloom Filter foi feita de forma genérica de forma a garantir que funciona para qualquer tipos de dados (String, int, byte[], ...).

Para além disso tanto o número de hash functions utilizadas, **k**, como o tamanho do filtro, **n**, são calculados de forma dinâmica em função do número de elementos que vão ser inseridos, **m**, e da probabilidade de ocorrência de falsos positivos desejada, **p**.

$$k = \frac{n}{m} * ln(2)$$

$$n = \frac{-m * ln(p)}{ln(2)^2}$$

Desta forma, o número de funções de hash é ótimo e está garantida uma percentagem de falsos positivos muito próxima da indicada ao programa. Tanto na aplicação de uso conjunto como nos testes, a probabilidade usada é 0.1%.

Devido ao facto de se tratar de um Counting Bloom Filter, foram implementadas as funções remove() e count().

1.2.2 Procura de elementos semelhantes (MinHash e Locality Sensitive Hashing)

À semelhança do Bloom Filter, este módulo também é genérico, funcional para qualquer tipo de dados.

Já que a vasta maioria de ficheiros é composta por milhares ou milhões de bytes, as shingles de um elemento são geradas utilizando um comprimento de 15 bytes.

Em vez de usar uma matriz de shingles, a assinatura de cada elemento é calculada (com o algoritmo de MinHash e 300 Hash Functions) enquanto este é adicionado.

O Locality Sensitive Hashing (LSH) divide a assinatura do elemento em 20 bandas de 15 rows cada. Assim, elementos com semelhança 0.6 têm menos de 0.1% de probabilidade de serem considerados candidatos, enquanto que elementos com semelhança 0.9 têm mais de 0.99%.

$$S = (\frac{1}{b})^{\frac{1}{r}} = 0.819$$

1.2.3 Contador Estocástico

Foi utilizada uma variante do contador estocástico que incrementa com uma probabilidade de $\frac{1}{16}$, de forma a aumentar o número até ao qual se pode contar, mas sem que o erro na aproximação aumente de forma significativa.

1.2.4 Funções de Hash

Nos primeiros 2 módulos todas as hash functions utilizadas são geradas através de **Universal Hashing**, da seguinte forma:

$$f(x) = (ax + b)\ mod\ p$$

Em que **a** e **b** representam números gerados aleatóriamente e **p** um número primo 'grande' (exemplo: 1086216277). Cada função tem valores de **a** e **b** diferentes. Assim garante-se que as funções não são correlacionadas.

Capítulo 2

Testes

2.1 CountingBloomFilter

Para correr o Teste do Counting Bloom Filter abre-se o terminal e executa-se o TestBloomFilter.jar:
java -jar TestBloomFilter.jar (mínimo Java 8)
Foram Criados 3 Filtros

Filtro 1

No primeiro filtro adicionam-se 1000 Strings aleatórias com 40 caracteres. Gerou-se 10000 novas strings aleatórias e verifica-se se pertencem ao bloom filter A probabilidade de qualquer das novas strings é muito baixa(por ter 40 caracteres aleatórios), por isso, todas as strings que passem no teste são contadas como falsos positivos.

Filtro 2

No segundo filtro adicionaram-se 3 Strings aleatórias, também de 40 caracteres. Verifica-se se 2 das strings adicionadas estão no filtro. Verifica-se que uma strings diferente não está no filtro. Remove-se uma das strings e verifica-se se está no filtro.

Filtro 3

No terceiro filtro adicionaram-se 100 Strings aleatórias em que uma delas aparece várias vezes no filtro. Verifica-se quantas vezes essa string se encontra no filtro. Remove-se algumas vezes essa string e voltamos a ver a contar essa string no filtro e verificamos que a diferença se verifica.

2.2 SimilarFinder

Para correr o Teste da MinHash abre-se o terminal e executa-se o TestSimilarFinder.jar:
java -jar TestSimilarFinder.jar (mínimo Java 8)

Lê-se um ficheiro com várias as 5 primeiras estrofes dos Lusíadas e 5 strings para teste. Verifica-se a existência de de strings semelhantes a "*As armas e os Barões assinalados*". As Strings candidatas pelo Locality Sensitive Hashing são diferentes em apenas um caracter.

O Min Hash verifica destas strings qual a mais próxima e calcula a diferença de Jaccard.

De seguida verifica a existência de strings duplicadas. No ficheiro encontra-se um string duplicada em que é verificada a sua distância de Jaccard (0).

2.2.1 Locality Sensitive Hashing

Adiciona-se 10 mil strings aleatórias com 40 caracteres. Adicionam-se mais 1000 strings aleatórias e para cada uma é contado o número de strings(das originalmente adicionadas) candidatas a serem semelhantes usando o Locality Sensitive Hashing. O ideal é não existir candidatos.

2.3 StochasticCounter

Para correr o Teste da Contagem Estocástica abre-se o terminal e executa-se o TestStochasticCounter.jar:
java -jar TestStochasticCounter.jar (mínimo Java 8)

Realizamos 3 testes com o contador cada um para probabilidades diferentes(1/2, 1/16 e 1/32).

Testamos o contador para valores de base 10 (10, 100, 1000, até 100000000); Em cada valor verificamos o valor que é registado pelo contador, aproximamos à ao valor real. Vemos a diferença e calculamos o erro. Quanto mais alto é o valor real menor a probabilidade erro.

2.4 DirectoryTools

É um teste automático da aplicação de uso conjunto. Para correr o Teste abre-se o terminal e executa-se o TestDirectoryTools.jar:
java -jar TestDirectoryTools.jar (mínimo Java 8)

Lê-se todos os ficheiros presentes no diretório "TestDirectory"e calcula-se o seu tamanho através da Contagem Estocástica.

Testamos se outros ficheiros do diretório TestFiles pertencem ou se são parecidos aos do diretório de teste e calcula-se a respetiva distância de Jaccard. Verifica-se o mesmo com ficheiros removidos. Faz-se a contagem de quantas vezes os ficheiros são adicionados. Deteta-se quais os ficheiros que estão duplicados.