Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Algoritmos e Estruturas de Dados
Semestre de Inverno 2017/2018
Segunda série de problemas – Segunda Parte

Observações:

- Data de entrega: **20 de Novembro de 2017**.

1. Problema: Indexação de Documentos

Com o desenvolvimento tecnológico do mundo moderno, a quantidade de informação disponível em formato digital tem vindo a aumentar. Todos os dias são gerados volumes enormes de dados de diferentes tipos e natureza, sendo cada vez mais importante a gestão e organização dessa informação para que a sua localização seja rápida e eficiente. Para isso, são usados ferramentas e mecanismos que endereçam essa necessidade, suportando operações de indexação e recuperação de documentos.

Pretende-se assim, a implementação de uma aplicação que dado um conjunto F = f_1, f_2, …, f_n de n ficheiros com informação não estruturada (texto), em que n > 0, efetue a sua indexação (simplificada), de forma a permitir a sua recuperação pelo método *Rank-Based Similarity Search*. Este método permite, ao fazer uma pesquisa por um conjunto de palavras chave, identificar os ficheiros cujo texto tenha mais similaridades (tendo em conta os termos e respetivo número de ocorrências) com as palavras chave indicadas.

A indexação e obtenção do *ranking* documentos tem as seguintes duas fases de processamento:

1. Fase de indexação. Pretende-se nesta fase duas operações:

   - A separação do texto em termos (palavras com significado), ignorando as designadas *Stopwords*, ou seja, os artigos definidos e indefinidos (“o”, “os”, “a”, “as”, “um”, “uns”, …), preposições (“em”, “por”, “para”, …) e conjunções (“e”, “se”, “que”, “me”, …). Considere que para a realização deste trabalho, dispõe de um ficheiro com um subconjunto de *stopwords*.

   Os ficheiros são representados através de estruturas de dados, designadas por (f_j), compostos pelos respetivos termos distintos $\{t_1, t_2, t_3, \dots, t_n\}$, associados aos seus números de ocorrências $\{x_1, x_2, \dots, x_n\}$.

   Exemplo:
   f_1 = {formiga formiga abelha} corresponde a {(formiga, 2), (abelha, 1)}
   f_2 = {cão abelha cão porco cão formiga cão} corresponde a {(cão, 4), (abelha, 1), (porco, 1), (formiga, 1)}
   f_3 = {gato antílope cão enguia raposa} corresponde a {(gato,1), (antílope,1), (cão,1), (enguia,1), (raposa,1)}

   - Calculo da dimensões dos documentos, segundo o método *Rank-Based Similarity Searc.*

   $d_j = \sqrt{x_1^2 + x_2^2 + x_3^2 + \cdots + x_n^2}$

   $|d_j|$ – É a dimensão do documento f_j, sendo $\{x_1, x_2, x_3, \dots, x_n\}$ o número de ocorrências de cada termo $\{t_1, t_2, t_3, \dots, t_n\}$.

2. Fase de obtenção do ranking. Pretende-se nesta fase, dado um conjunto de palavras chave, obter os documentos que mais se assemelham, ou seja, cujo conteúdo tenha mais similaridade com as palavras chave indicadas num ficheiro (*query*). O resultado deverá ser uma lista por ordem decrescente de *ranking* de similaridades.

   A similaridade entre uma *query* (q) e a dimensão de um documento (d_j) é dada por:

   $$\cos(q, d_j) = \frac{\sum_{k=1}^{n} w_{jk}}{|q|\,|d_j|}$$

   Em que:
   n – É o número de termos da *query*
   w_{jk} – É o número de ocorrências (*weight*) do k-ésimo termo da *query* no documento d_j
   $|q| = \sqrt{x_1^2 + x_2^2 + x_3^2 + \cdots + x_n^2}$ – É a dimensão da *query*, sendo $\{x_1, x_2, x_3, \dots, x_n\}$ o número de ocorrências de cada termo (normalmente na *query* cada termo ocorre uma vez)

   Para o exemplo de 1. e assumindo a *query* = {(formiga,1), (cão,1)}, obtêm-se as seguintes dimensões, para a *query* e para cada um dos documentos:

	formiga	abelha	gato	cão	enguia	raposa	antílope	porco	
q	1			1					$\|q\| = \sqrt{1^2 + 1^2} = \sqrt{2}$
d_1	2	1							$\|d_1\| = \sqrt{2^2 + 1^2} = \sqrt{5}$
d_2	1	1		4				1	$\|d_2\| = \sqrt{1^2 + 1^2 + 4^2 + 1^2} = \sqrt{19}$
d_3			1	1	1	1	1		$\|d_3\| = \sqrt{1^2 + 1^2 + 1^2 + 1^2 + 1^2} = \sqrt{5}$

Calculando as similaridades, obtém-se:

	formiga	abelha	gato	cão	enguia	raposa	antílope	porco	
q	1			1					
d_1	2	1							$\cos(q, d_1) = \frac{2}{\sqrt{2} * \sqrt{5}} = 0.63$
d_2	1	1		4				1	$\cos(q, d_2) = \frac{1+4}{\sqrt{2} * \sqrt{19}} = 0.81$
d_3			1	1	1	1	1		$\cos(q, d_3) = \frac{1}{\sqrt{2} * \sqrt{5}} = 0.32$

Resultando no seguinte *ranking* de similaridades:

q	formiga	cão
d_2	0.81	
d_1	0.63	
d_3	0.32	

Objetivo

Pretende-se o desenvolvimento de uma aplicação que dado um conjunto de ficheiros de texto, faça a sua indexação e obtenção do *ranking*, A indexação (ponto 1.) é realizada no início da execução da aplicação. A obtenção do *ranking* (ponto 2.) é realização através de um comando em que é dado um ficheiro que contém as palavras da *query*.

Parâmetros de Execução

A aplicação a desenvolver terá que suportar os seguintes parâmetros:

```
java RankBySimilarities stopwords.txt f1.txt f2.txt f3.txt
```

Durante a execução, a aplicação deverá suportar o seguinte comando:

- **`ranking query.txt`** – mostra o *ranking*, por ordem decrescente, dos documentos pelo método *Rank-Based Similarity Search*, dado um conjunto de palavras chaves contidas no ficheiro **`query.txt`.**

Para realizar este exercício é necessário definir no tipo de dados genérico **`HashMap<K,V>`** fornecido em anexo as variáveis de instância, os construtores e os métodos indicados, tendo em atenção que:

- **`public V put(K key, V value)`**-Associa o valor `value` à chave `key` no mapa. Se o mapa já contiver um mapeamento para a chave, substitui-o pelo novo valor. Esta operação deverá retornar o valor que se encontrava associado à chave, ou `null` caso não exista mapeamento para a chave;
- **`public V get(K key)`**- Retorna o valor ao qual a chave `key` está mapeada, ou `null`, se o mapa não contiver um mapeamento para essa chave;
- **`public V remove(K key)`**- Remove o mapeamento para a chave `key` no mapa, caso exista. Retorna o valor que estava associado à chave, ou `null` se o mapa não contiver um mapeamento para essa chave;
- **`public int size()`** - Retorna o número de mapamentos chave-valor presentes neste mapa.
- **`public Set<Map.Entry<K,V>> entrySet()`**- Retorna uma vista do conjunto dos pares chaves-valor contidos neste mapa. Um mapeamento chave-valor é especificado através do tipo **`java.util.Map.Entry<K,V>`**, através das seguintes operações:
  - **`K getKey()`** - Retorna a chave correspondente a esta entrada.
  - **`V getValue()`** - Retorna o valor correspondente a esta entrada.

Avaliação Experimental

Realize uma avaliação experimental do(s) algoritmo(s) desenvolvido(s) para a resolução deste problema. Apresente os resultados graficamente, utilizando uma escala adequada.