Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática
Mestrado em Engenharia Informática

Recuperação de Informação

1º Trabalho

Autores
David Ferreira, Nº 72219
Tiago Faria, Nº 73714

Índice

Resumo

Neste documento é feito um *overview* do sistema implementado. É exibida e explicada a sua estrutura com também são analisadas e descritas algumas decisões tomadas. Contém também especificações de como executar o programa, como também demonstra alguns pormenores sobre a sua eficiência.

Diagrama de Classes

O projeto é composto por um total de cinco packages e de treze classes. Cada package corresponde ao Corpus Reader, Indexer, Main, Tokenizers e Utils (utilidades), respetivamente.
Na figura seguinte é apresentado um diagrama de classes do sistema. Para uma visualização mais clara e específica, encontra-se disponível o ficheiro UML.png.

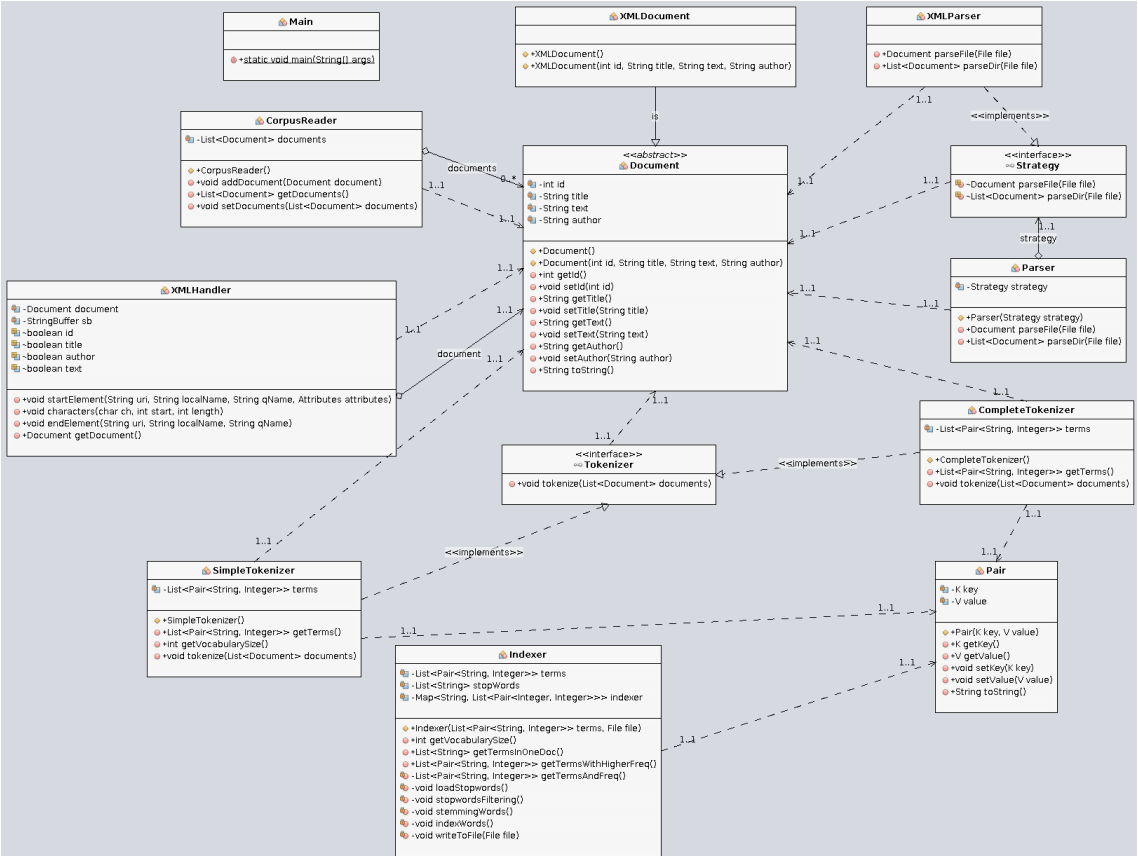

Figura 1 - Diagrama de Classes do projeto

Package Corpus Reader

O package CorpusReader é composto por classes que contribuem para a leitura, criação, edição e construção de ficheiros. Mais especificamente, o CorpusReader contem as seguintes classes:

- **Document:** classe abstrata que tem o intuito de representar um documento como um objeto. O mesmo utiliza atributos como o id, título (title), texto (text) e autor (author), contendo sets e gets para cada um.
- **XMLDocument:** extensão do Document, orientado para ficheiros XML.
- **Strategy:** vista a possibilidade de conseguir carregar ficheiros com estruturas diferentes, optou-se por utilizar o padrão Strategy. Assim, esta interface, com jus ao nome do padrão, contém os métodos necessários para fazer o parse a um único ficheiro (método parseFile) ou a uma diretoria (método parseDir).
- **Parser:** seguindo a estrutura do padrão descrito anteriormente, o sistema contém a classe Parser de modo a ser possível utilizá-lo. Assim, o Parser trata de usufruir os métodos pré-definidos no Strategy.
- **XMLParser:** por forma a especificar o parsing para ficheiros XML, criou-se o XMLParser. A classe implementa os métodos do Strategy com a estrutura e definições dos ficheiros do tipo XML.

- **XMLHandler:** o XMLHandler foi criado com o intuito de fazer o parsing de ficheiros XML, utilizando o parser SAX. Com os três métodos principais startElement, characters e endElement, é possível criar objetos dos documentos lidos.
- **CorpusReader:** a gestão dos documentos é feita no CorpusReader. A classe é composta por um conjunto de documentos onde é possível retornar uma lista ou adicionar mais objetos do mesmo tipo (addDocument).

Package Indexer

O package indexer contém apenas uma classe do mesmo nome. Recebendo como parâmetros uma lista de termos e um ficheiro de saída, é criado um indexer. Para tal, é necessário fazer um stopword filtering e um stemming, ações essas que correspondem aos métodos stopwordsFiltering e stemmingWords, respetivamente. Após a execução dos dois métodos anteriores, o resultado da estrutura do indexer é guardada num ficheiro (método writeToFile), nomeado previamente pelo utilizador na inserção de argumentos na linha de comandos.
A classe contém ainda métodos que retornam informações pedidas como o tamanho do vocabulário, os dez primeiros termos (ordenados alfabeticamente) que aparecem num só documento e os primeiros dez termos que ocorrem com maior frequência.

Default Package

No default package está contida a classe Mais que permite a execução da aplicação. Nela é verificado o número de argumentos e o programa corre de acordo com o seu conteúdo.

Package Tokenizers

Os dois tipos de tokenizers pedidos estão embutidos neste package. A estrutura pedida no segundo ponto do enunciado corresponde à classe SimpleTokenizer. Esta classe trata um só documento ou um conjunto deles, separando os termos por espaços e modificando todo o seu conteúdo para caracteres de letras minúsculas. No seu processo, são removidos todos os caracteres não-alfabéticos e só são aceites termos de extensão maior ou igual a três.
Por outro lado, o CompleteTokenizer processa o texto, dividindo-o em termos que têm especial atenção a caracteres especiais. Neste proceso já não há a restrições relativas aos dígitos e ao número mínimo de caracteres do termo.
Ambas as classes descritas anteriormente implementam a interface denominada Tokenizer. Esta contém um método denominado por tokenize, onde é feita a tokenização dos termos.

Package Utils

A classe Pair é o único ficheiro contido neste package. Nela está implementada uma estrutura de dados que permite gerir pares de <chave, valor>, auxiliando a manipulação da informação.

Diagrama Data Flow

A próxima figura representa os passos de execução do sistema:

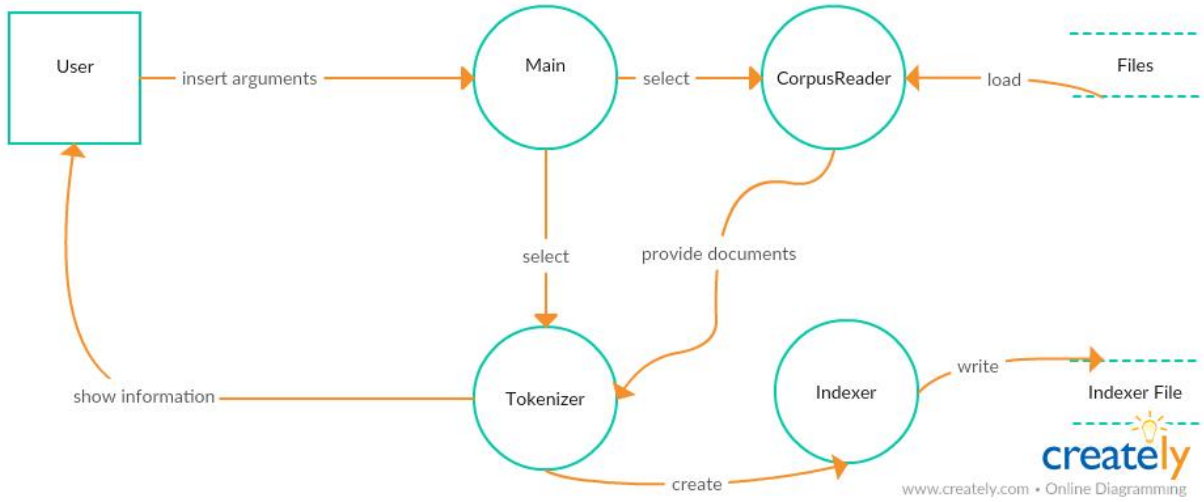

Figura 2 - Data Flow Diagram do sistema

Resumidamente, o sistema funciona da seguinte forma:

1. O utilizador insere três argumentos, indicando o ficheiro de leitura, o tipo de tokenizer a utilizar e o ficheiro onde será guardado o indexer.
2. Dado o caminho do ficheiro ou a diretoria de leitura, o Corpus Reader lê todos os ficheiros e faz o respetivo parsing.
3. Mediante a escolha do tokenizer por parte do utilizador, o sistema efetua a tokenização dos documentos lidos.
4. Após o passo anterior, é criado um indexer onde também é realizado o stopwording e stemmer. Por fim, o indexer é gravado num ficheiro.
5. São mostradas ao utilizador informações como o tamanho do vocabulário, os dez primeiros termos registados num só documento e os termos mais frequentes.

Execução do Programa

Como já foi dito anteriormente, para correr o sistema são necessários três argumentos. No primeiro campo, é necessário colocar o caminho de um único ficheiro ou o de uma diretoria. Se for colocado o caminho desta última, o sistema lerá um a um todos os ficheiros contidos dentro na mesma.
No segundo argumento, é pedida a escolha entre o tokenizer simples (ponto dois) ou o mais completo (ponto quatro). A diferença está entre inserir t1 e t2, respetivamente.
Por último, é pedido um nome que servirá para criar o ficheiro de saída que conterá o indexer. Este ficheiro não pode ter o nome de outro já existente.
O programa é executado através da classe Main.

Exemplos de argumentos de entrada:

- Ficheiro único com tokenizer simples: **[file.txt t1 index.txt]**
- Diretoria com tokenizer completo: **[cranfield t2 index.txt]**

Livrarias Externas

É de realçar o uso do Java 8 no projeto. À exceção do Java, o sistema usa uma dependência externa, o Porter Stemmer. Tal livraria pode ser encontrada em http://snowball.tartarus.org/download.html.

Medidas de Eficiência

Por forma a perceber a eficiência do sistema, foram feitas três experiências distintas relativas ao tempo total de indexação, memória utilizada durante a indexação e o tamanho do indexer em disco. Ambas foram executadas com os diferentes tokenizers, apresentando os seus resultados nos próximos tópicos.
Realça-se para o facto de estes valores serem subjetivos de máquina para máquina.

Tempo Total de Indexação

Utilizando a unidade de medida milissegundos, verificou-se que o programa ao utilizar o tokenizer simples demorou 646 ms a indexar.

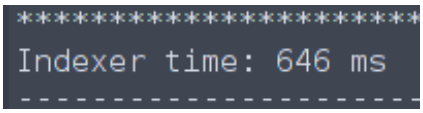

Figura 3 - Tempo de indexação do primeiro tokenizer

Por outro lado, a utilização do tokenizer mais complexo levou à demora de 1134 ms.

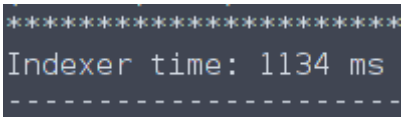

Figura 4 - Tempo de indexação do segundo tokenizer

Memória Utilizada

A quantidade de memória utilizada pelo programa ao usufruir dos serviços do primeiro tokenizer encontra-se representada na seguinte figura:

Figura 5 - Memória utilizada no primeiro tokenizer

Enquanto no outro tokenizer, o total de memória utilizada foi a seguinte:

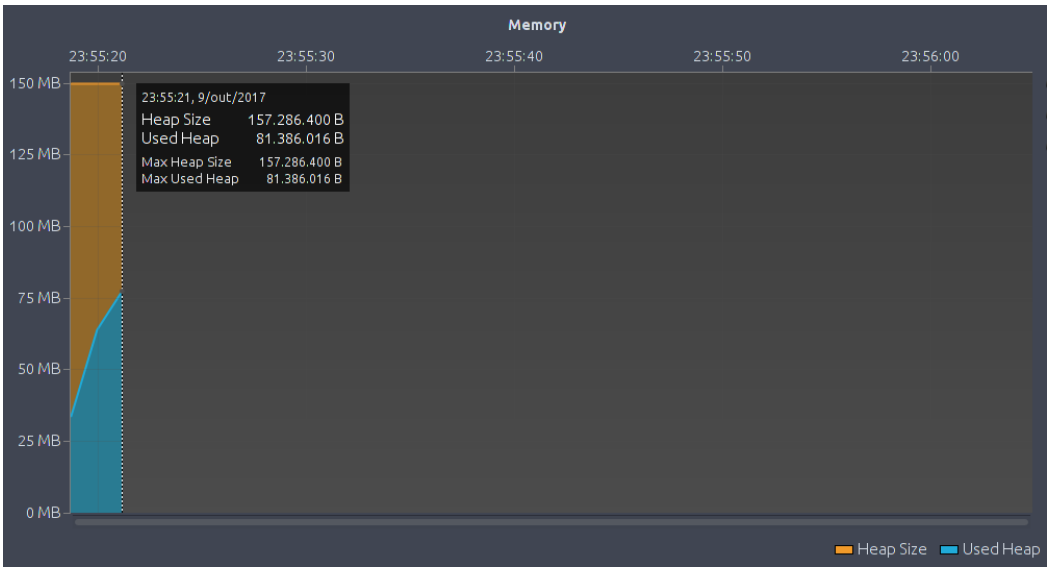

Figura 6 - Memória utilizada no segundo tokenizer

Tamanho de Indexação em Disco

Utilizando o indexer no primeiro tokenizer, gerou-se um ficheiro com um tamanho de 726,4 KB. O ficheiro do indexer do segundo tokenizer será mais leve devido à utilização do stopword filtering e do stemming.

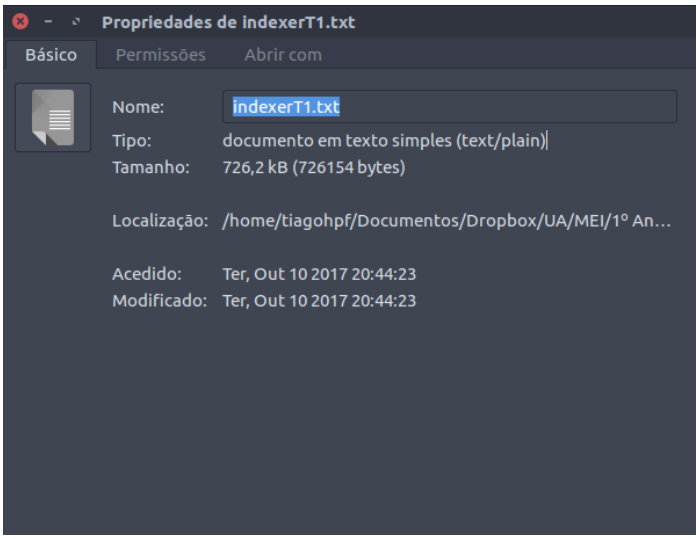

Figura 7 - Tamanho de indexação do primeiro tokenizer

Por outro lado, utilizando o segundo tokenizer com regras mais flexíveis sobre caracteres especiais e dígitos, criou-se um ficheiro de 569,3 KB.

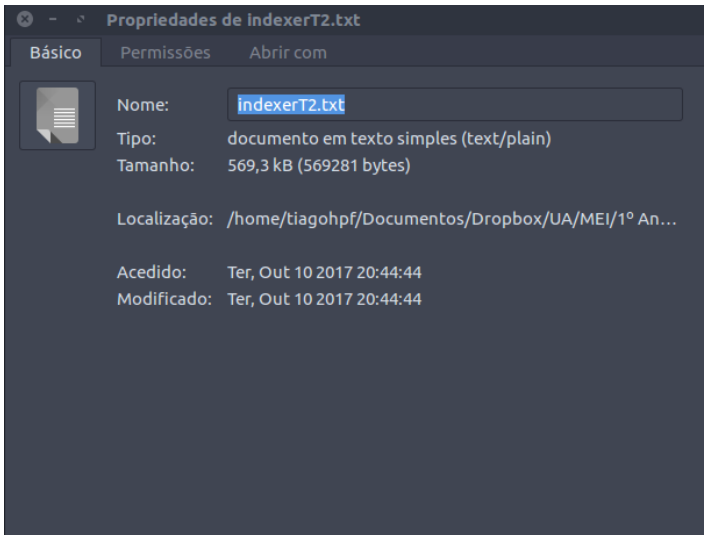

Figura 8 - Tamanho de indexação do segundo tokenizer

Conclusões

De grosso modo, o sistema cumpre com os objetivos traçados e corre sem nenhum imprevisto. Contudo, elementos como o segundo tokenizer precisam de alguns reparos. No caso da existência de alguns caracteres especiais, resulta uma tokenização nem sempre correta dos termos, especialmente em palavras compridas que contém o caracter '-'. É necessária uma melhor validação dos mesmos.

Quanto às decisões tomadas durante a implementação, realçam-se as seguintes:

- O padrão Strategy é utilizado com o intuito de poder gerir diferentes estruturas de documentos e fazer o respetivo parsing. Desta forma, o código encontra-se adaptável para mudanças num futuro próximo;
- O nome do ficheiro de saída do indexer tem de ser único, ou seja, não é permitido atualizar nenhum ficheiro já existente. Assim, é impedido que o ficheiro de entrada seja substituído pelo ficheiro de saída;
- O tokenizer mais complexo lida com vírgulas, hífens, pontos e números, não os desprezando. Aceita números decimais separados por ponto ou vírgula e mantém o hífen na palavra se a mesma for pequena (tamanho menor do que dez), caso contrário, é desprezado;
- Apesar de ser sintaticamente menos legível, foi escolhido o SAX parser em vez do DOM. O SAX só usa uma pequena parte do ficheiro na memória enquanto o DOM carrega-o todo.

Referências

[1] "Design Patterns - Strategy Pattern." [Online]. Available: https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm.

[2] "Java XML Parsers." [Online]. Available: https://www.tutorialspoint.com/java_xml/java_xml_parsers.htm.

[3] Mkyong, "How to read XML file in Java – (SAX Parser)," 2015. [Online]. Available: https://www.mkyong.com/java/how-to-read-xml-file-in-java-sax-parser/.