universidade de aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Information Retrieval Engine

Implement a simple corpus reader, tokenizer, and Boolean indexer.
Tarefa 2

Curso	[8240] MI em Engenharia de Computadores e Telemática
Disciplina	[42596] Recuperação de Informação
Ano letivo	2016/2017

Alunos	[68021] Gabriel Vieira, gabriel.vieira@ua.pt
	[68779] Rui Oliveira, ruipedrooliveira@ua.pt
Grupo	5
Docente	Professor Sérgio Matos, aleixomatos@ua.pt

Aveiro, 26 de Outubro de 2016

Conteúdo

1 Enquadramento

Pretende-se através deste relatório expor sob forma escrita, o nosso desempenho e objetivos alcançados na primeira tarefa do trabalho prático desenvolvido no âmbito da unidade curricular de Recuperação de Informação.

Neste trabalho pretende-se criar um mecanismo de recuperação de informação. Este será composto de vários módulos, são eles: *corpus reader* , *document processor*, *tokenizer*, *indexer*,*searcher* e *ranker*.

Na primeira iteração do projecto, prodeceu-se à sua modelação, definindo e descrevendo as classes, métodos (principais) e o fluxo de dados.

Nesta segunda iteração do projeto pretende-se relatar o que foi feito atendendo à modelação da primeira parte, explicar com mais detalhe cada uma das classes e suas funções e todo o fluxo existente neste trabalho.

O fluxo de operações aplicado a um conjuntos de documentos apresenta-se no diagrama seguinte.

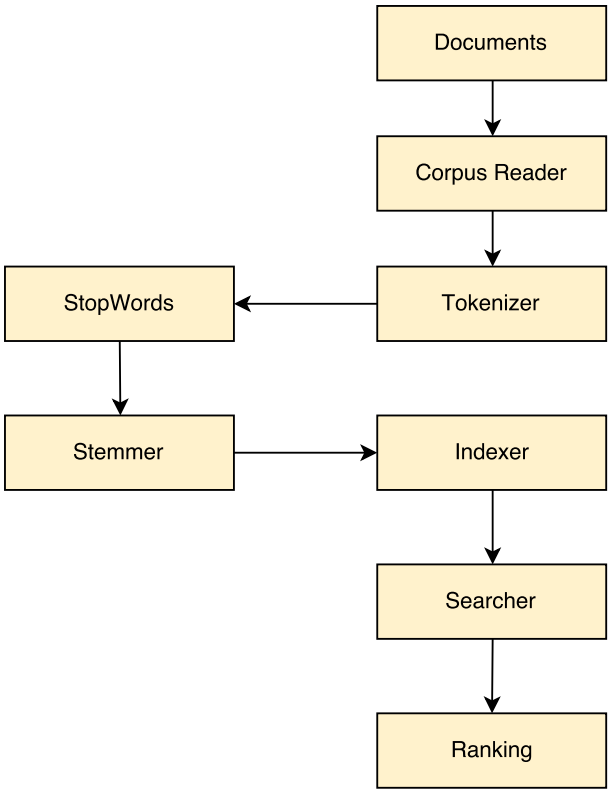

Figura 1: Diagrama geral

2 Arquitetura geral

Nesta secção iremos apresentar todos os módulos do nosso projeto e um esquema que representa o fluxo geral de execução do mesmo.

2.1 Componentes

2.1.1 *CorpusReader*

Este módulo trata da abertura, leitura e pré-processamento de cada um dos documentos presentes no conjunto de documentos a processar.

2.1.2 *Tokenizer*

Este módulo tem como função formar um conjunto de tokens (Array de tokens) com valor semântico recorrendo à sequência de caracteres que é fornecida pelo módulo *corpus reader*. Numa fase inicial, após o Tokenizer os termos serão imediatamente indexados através do Indexer sem sofrer qualquer processamento, apenas mais tarde e após garantirmos o total funcionamento(Tokenizer+Indexer), o processo passará pelos dois módulos que a seguir apresentamos.

2.1.3 *StopWords*

Esta classe lê o ficheiro de stop words fornecido e adiciona-as a uma lista. Esta lista é depois mais tarde usada para se comparar com um array de tokens e verificar se uma palavra é ignorada ou não.

2.1.4 *Stemmer*

Este módulo tem como objetivo tratar as variações morfológicas das palavras, permitindo que apenas a palavra principal seja indexada, pois na maioria dos casos têm significados semelhantes. A implementação do stemmer irá recorrer ao *The Porter stemming algorithm* disponível através do link `http://snowball.tartarus.org/algorithms/porter/stemmer.html`

2.1.5 *Indexer*

Consiste numa estrutura de dados que tem como objetivo aumentar o desempenho da pesquisa, recorrendo à indexação dos termos de um dado conjunto de documentos. Neste caso, a estratégia utilizada será o Inverted Indexer que consiste em, para cada termo, guardar uma referência com a

identificação do documento em que está presente, bem como o seu número de ocorrências nesse documento.

Nesta primeira fase do projeto, pensamos que a melhor estrutura de dados a adotadar seria:

- **TokenFreqMap**: `HashMap<String, Integer>`

  Este tipo de dados herda as características de uma *HashMap*, em que a *key* corresponde ao *token* e o *value* à frequência com que o *token* aparece em todos os documentos existentes.

- **TokenIDDocFreqMap**: `HashMap<String<HashMap<Integer, Integer>`

  Este tipo de dados herda as características de uma *HashMap*, em que a *key* corresponde ao *token* e o *value* corresponde a uma outra *HashMap*. Nessa *HashMap* a *key* corresponde ao identificador do documento e o *value* ao número de ocorrências desse *token* no documento.

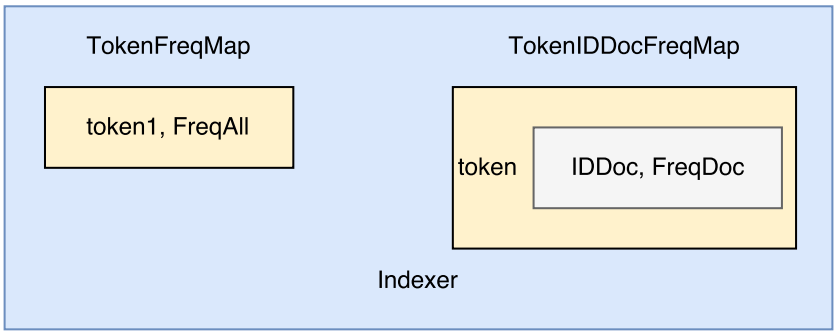

Figura 2: Fluxo geral de execução

Nota: Eventualmente, para outros corpus o tipo de dados da variável `IDDoc` terá que ser alterada, uma vez que, caso tenhamos `IDDoc` superiores a 2147483647 o tipo `Integer` não suportará. Em alternativa será utilizado o tipo `Long`.

2.1.6 *MemoryManagement*

Este módulo permitirá fazer uma gestão da memória, sendo que quando a memoria virtual excede certo limite, a informação é armazenada em disco. Através deste módulo é possível aceder à quantidade de memoria virtual consumida pela JVM.

Nota: Nesta iteração do projeto ainda não foi implementado qualquer mecanismo de gestão de memória, apenas foram criadas funções que nos retornam a memória RAM gasta até ao momento.

2.1.7 *Searcher*

Este módulo tem como objetivo fazer as pesquisas necessárias aos ficheiros resultantes da indexação, bem como filtrar o resultado das referidas pesquisas.

2.1.8 *Ranker*

Este módulo tem como objetivo ordenar os resultados provenientes do módulo de *Searcher*

2.1.9 *DocumentProcessor*

Este módulo tem como objetivo instanciar todo o fluxo associado à execução de um documento. Para além disso, o *document processor* irá iterar sob todos os documentos existentes num dado conjunto de documentos .

2.1.10 *SearcherProcessor*

Este módulo tem como objetivo instanciar todo o fluxo associado à pesquisa, isto é, aguarda pela receção de queries realizadas pelo utilizador, solicita ao Tokenizer a tokenização do conteúdo do documento e envia os vários tokens para um objeto do tipo Query.

2.2 Diagrama geral

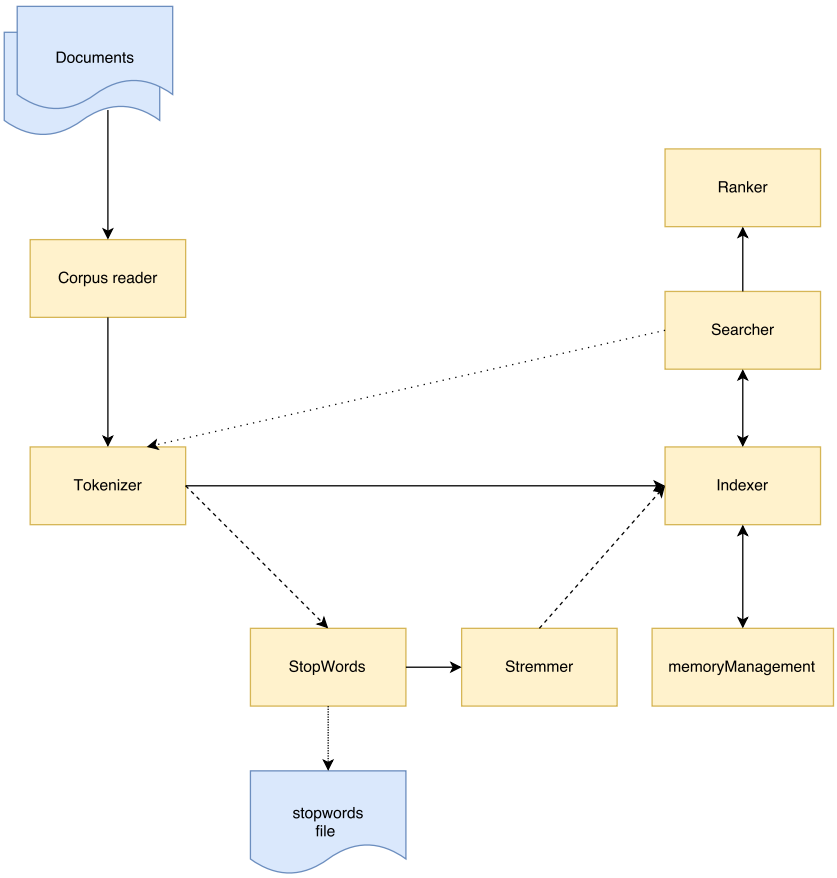

Figura 3: Fluxo geral de execução

A figura anterior mostra-nos de forma esquemática o fluxo de dados no nosso projeto.

O corpusReader irá encarregar-se de ler todos os ficheiros que se encontram num determinado diretório e de efetuar o seu pre-processamento. O texto pre-processado é dirigido para o nosso Tokenizer que se encarrega de transformar uma String num conjunto de termos válidos (tokens).

Numa primeira fase pretendemos que todos os termos sejam válidos, de modo a garantirmos o funcionamento do indexer, sendo enviados todos os tokens existentes no array de tokens. Posteriormente, iremos usar técnicas de stemming e um filtro de stopwords (setas representadas a tracejado).

O filtro de stopwords irá reduzir o array de tokens que anteriormente referimos, uma vez que irá comparar todos os candidatos a token com as pa-

lavras da lista de stopwords[1]. Todas as palavras não relevantes (stopwords) e posteriormente passado por um Porter Stemmer que converte palavras morfologicamente distintas com o mesmo significado para um token comum.

Relativamente ao Indexer, este é o componente mais complexo do nosso projeto. Faz dois tipos de indexação: um mapa de frequência e um mapa de referências. O primeiro consiste numa estrutura de dados que faz o mapeamento entre um termo e o seu número total de ocorrências em todo o projeto. O segundo faz o mapeamento entre número de ocorrências de um termo em cada um dos ficheiros lidos. No final da indexação todos os termos são guardados em dois ficheiros, um para a frequência e outro para as referências.

Em relação à pesquisa, o seu pipeline é iniciado pelo módulo SearcherProcessor. Quando este inicia, é carregado para memória o conteúdo do ficheiro resultante do Pipeline de indexação que consiste no mapeamento, entre a identificação do documento utilizada na indexação e a real identificação do documento. Quando o Pipeline está preparado para receber as queries vindas do utilizador, este bloqueia e aguarda para que estas sejam recebidas. Assim que uma query é recebida, esta passa pelo Tokenizer, onde o seu conteúdo é tratado e resultando daí um conjunto de tokens. Este conjunto pode ou não passar pelo filtro de StopWords bem como pelo Porter Stemmer. Antes da query ser iniciada, é criado um objecto do tipo Query para guardar o tipo e conteúdo da query feita pelo utilizador, passando o fluxo do pipeline do tokenizer para Query. Depois de tudo isto, cada query é enviada para o módulo Searcher que faz uso do método getTerms() que obtém a lista de termos a serem pesquisados.

Por fim temos o Ranker (ou Ranking) que vai ser responsável por, tendo em conta a lista de termos a serem pesquisados, ordenar o resultado da pesquisa efetuada no módulo Searcher.

[1]Palavra que é removida antes ou após o processamento de um texto em linguagem natural

3 Mudanças em relação à primeira iteração

Neste relatório é relatado com mais pormenor cada uma das classes e suas funções, visto que como foi implementada a primeira parte do projecto já existe um conhecimento mais vasto sobre cada uma das classes.

Na subsecção *Classes e métodos* foi adicionada uma descrição mais pormenorizada sobre o que foi feito em cada uma das classes e as suas funcionalidades. Também foi atualizado o diagrama de classes do respetivo trabalho, como se pode ver na secção seguinte de nome *Diagrama de classes*. O diagrama foi atualizado de acordo com os métodos usados em cada uma das classes bem como também os seus atributos. As alterações no diagrama só são percetíveis nas classes *CorpusReader*, *DocumentProcessor*, *StopWords*, *Stemmer*, *Indexer*, *TokenFreqMap* e *TokenIDDocFreqMap* pois apenas estas classes foram as necessárias para cumprir com sucesso esta iteração do projeto.

Foi também adicionada uma nova classe ao trabalho para especificar a localização do ficheiros, bem como o ID do ficheiro concatenado com o ID do documento, pois prevê-se que possa existir o mesmo DocID em ficheiros diferentes.

4 Diagrama de classes

Na figura seguinte apresenta-se o diagrama de classes do nosso projeto.

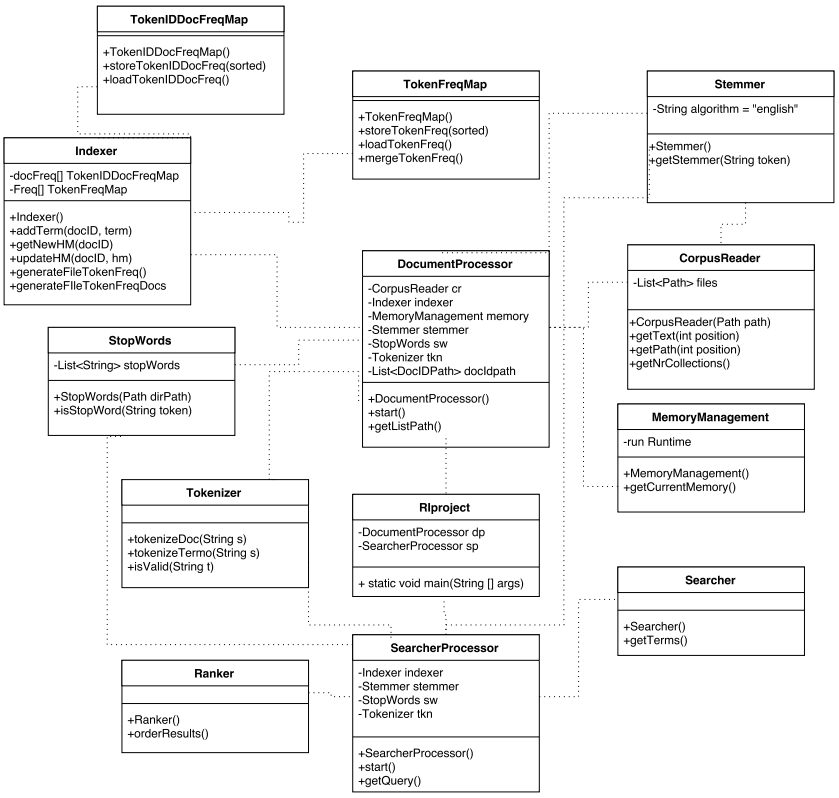

Figura 4: Diagrama de classes

4.1 Classes e métodos

4.1.1 RIproject (main)

Classe principal do projeto. É aqui onde são instanciadas as classes DocumentProcessor e SearcherProcessor que iremos descrever de seguida.

4.1.2 DocumentProcessor

Principais métodos e sua descrição:

- **start()**: método responsável pela iniciação do pipeline de todo o processo. Este método é invocado pela main classe para se iniciar o processo de iteração na coleção de ficheiros existente.

- **getListPath()**: método que retorna a lista dos caminhos onde os ficheiros de extensão "arff"se situam concatenados com o ID de cada ficheiro.

Descrição: Neste trabalho, esta classe começa por ler os documentos que estão presentes no directório especificado. Percorre os documentos e com o auxílio da classe *CorpusReader* obtém o texto dos documentos presentes em cada ficheiro. Depois do *CorpusReader* ler todo o texto dos documentos e fazer a sua filtragem, este módulo vai de seguida enviar todo o texto para o *Tokenizer* para este "tokenizar"todo o texto, guardando cada termo num array de strings. Depois de tokenizar o texto, o Document Processor vai percorrer a coleção dos ficheiros e obter o ID de cada documento, para melhor especificar quais os termos que pertencem a um dado documento. Após obter o ID de cada documento, este módulo vai "tokenizar"apenas os termos de cada documento (já filtrados) usando o método *tokenizeTermo* da classe *Tokenizer*, guarda cada um deles num array de strings e depois vai percorrer esse array. Ao percorrer esse array, vai verificar se cada token é valido, ou seja, no nosso caso para ser válido tem de ter mais de 2 caracteres e conter texto. Se o termo não for válido é descartado. Caso seja, então vai-se verificar se esse termo está contido na lista de stop words, usando o método *isStopWord* da classe *StopWord* (explicado mais à frente). Se esse termo constar na lista de stop words, então é descartado, caso contrário vai ser feito o seu stemming usando o módulo *Stemmer* que vai ser responśavel por obter o stemmer especificado, que neste caso é o inglês. Depois de tudo isto, então o termo é indexado e ao fim de percorrer todas os termos, é feita a contagem dos termos por documento e a listagem dos ID's dos documentos onde os termos aparecem, usando os módulos *TokenFreqMap* e *TokenIDDocFreqMap*.

4.1.3 CorpusReader

Principais métodos:

- **getText()**: Sempre que o DocumentProcessor inicia a leitura de um documento, solicita a este método o conteúdo de um novo documento. O novo ficheiro é lido, recorrendo a Stream que permite fazer o pré-processamento de cada linha.
- **getPath(int position)**: método que é responsável por retornar o path de cada ficheiro de acordo com a posição que este se encontra no directório.
- **getNrCollections()**: método que retorna a número total de ficheiros a serem lidos.

Descrição: O Corpus Reader vai ler cada documento e fazer a filtragem do texto linha a linha, usando o Stream disponibilizado pelo Java 8. Em cada linha são eliminadas as tags < >, os M[0-9], pontuações, caracteres pouco relevantes para indexar, a letra é passada toda para minúscula e também filtra todos os espaços e ignora o texto que começa com o carácter "@"que foi pedido pelo docente para ignorar. Para além disto, o Corpus Reader tem um método que retorna o número de coleções (número de ficheiros arrf a serem lidos no directório), bem como um método que retorna uma string com o método.

4.1.4 Tokenizer

Principais métodos:

- **tokenizeDoc(String s)**: faz uso de uma expressão regular para trocar o conteúdo dos caracteres, depois faz-se a separação da String num array de Strings linha a linha
- **tokenizeTermo(String s)**: faz uso de uma expressão regular para trocar o conteúdo dos caracteres, depois faz-se a separação da String num array de Strings termo a termo.
- **isValid(String t)**: valida o conteúdo final dos tokens, andes da sua indexação.

Descrição: Este módulo faz a tokenização do texto e dos termos. Usando o método *tokenizeDoc* a tokenização do texto é feita linha a linha e depois guardado num array de Strings, já o método *tokenizeTermo* a tokenização é feita termo a termo e cada termo é guardado num array de strings. Para além destas funções, este módulo tem ainda um método que verifica se um termo é ou não válido de nome *isValid* ao qual já foi explicada a sua função na descrição do Document Processor.

4.1.5 StopWords

Principais métodos:

- **isStopWord(String token)**: verifica se o token que irá ser indexado é ou não uma stop word. Se for uma stop word, então o termo é ignorado, caso contrário prossegue para a indexação.
- **getSize()**: retorna o tamanho da lista de stopwords.

- **getStopWords()**: retorna a lista de stop words.

Descrição: Este módulo inicialmente vai ler a lista de Stop Words, disponibilizada pelo docente. Este módulo depois contém um método booleano de nome *isStopWord* que vai verificar se o termo que é passado por argumento está contido na lista de stop words ou não. Caso não esteja contido informa, o Document Processor que pode avançar para a próxima etapa, caso contrário é descartado.

4.1.6 Stemmer

Principais métodos:

- **getStemmer(String token)**: pretende identifiar variações morfológicas de palavras e transforma-las numa mesma palavra comum. Este método recebe um token e caso seja possível de transformação essa operação é efetuada. É retornado o token transformado ou original (recebido como argumento).
- **getAlgorithm()**: retorna o algoritmo de stemming a ser usado. Neste caso é usado o "english".

Descrição: Este módulo vai obter o stemmer para os termos usando Port Stemmer da biblioteca *SnowBall*. No módulo, foi necessário especificar o algoritmo a usar, que neste caso foi o "englishStemmer"já com todas as regras de stemming implementadas, sendo apenas necessário saber onde e como usar. Para que esta biblioteca seja reconhecida, foi necessário incluir no projecto e nas dependências do Maven o jar do snowball (`snowball.jar`).

4.1.7 Indexer

Principais métodos:

- **addTerm()**: método usado para adicionar termos às estruturas de dados, caso estes não existiam. Se existirem a estrutura de dados é actualizada com a nova informação recolhida.
- **getNewHM(int docId)**: caso uma key não tenha um value, este é adicionado. Ou seja, se aparece um termo que ainda não tem nada na sua lista de postings, então é adicionada a esta lista o ID do documento que contém este token.

- **updateHM(int docId, hashmap<integer,integer> hm)**: caso um termo aparece mais que nos documentos, então é adicionado o ID do documento correspondente à lista de postings.

- **generateFileTokenFreqDocs()**: método responsável por gerar o documento de texto com a seguinte impressão: token, freq, postings list, conforme é pedido para esta parte do trabalho.

Descrição: Este módulo é responsável por fazer a indexação dos termos. A função *addTerm* é a responsável por fazer tal indexação, usando as funções das classes *TokenFreqMap* e *tokenIDDocFreq*. Quando o termo é indexado, conta o número de vezes que o token aparece, usando a função *TokenFreqMap* e na outra estrutura conta também o número de documentos em que o token aparece e faz uma listagem dos DocIDs em que este token aparece. Antes de indexar o termo nesta segunda estrutura, é feita a verificação se este termo (key) tem um value correspondente. Caso não tenha, então usa a função *getNewHM* para obter um value correspondente ao token (key), mas caso tenha já um value, esse value irá ser actualizado usando a função *updateHW*. Por fim, as funções *generateFileTokenFreq* e *generateFileTokenFreqDocs* vão gerar os ficheiros de saída, no qual o ficheiro de nome *TokenFreq.txt* localizado no path **outputs/TokenFreq.txt** contém o token e a frequência com que este aparece nos documentos na totalidade. Já o ficheiro de nome *TokenFreqDocs.txt* localizado no path **outputs/TokenFreqDocs.txt** contém o termo, o número total de vezes que este aparece e a lista de DocIDs em que aparece.TokenFreqMap e TokenIDDocFreqMap As estruturas de dados criadas para o armazenamento do indexer terá pelo menos os seguintes métodos:

TokenFreqMap

- **storeTokenFreq()**: método usado para escrever conteúdo indexado da estrutura `TokenFreqMap` no ficheiro de *output*.

- **loadTokenFreq()**: ler conteúdo indexado na estrutura `TokenFreqMap`

TokenIDDocFreqMap

- **storeTokenIDDocFreq()**: método usado para escrever conteúdo indexado da estrutura `TokenIDDocFreqMap` no ficheiro de *output*.

- **loadTokenIDDocFreq()**: ler conteúdo indexado na estrutura `TokenIDDocFreqMap`

DocIDPath

Esta classe permitirá aceder aos atributos de um dado ficheiro, sendo para isso apenas necessário o ID do documento.

4.1.8 SearcherProcessor

Principais métodos:

- **start()**: método responsável pela iniciação do fluxo de todo o processo de pesquisa. Este método é invocado pela main classe para se iniciar o processo de pesquisa.
- **getQuery()**: método permite ao programa receber uma query de entrada inserida pelo utilizador.

4.1.9 Searcher

Principais métodos:

- **getTerms()**: método para obter a lista dos termos a serem pesquisados

4.1.10 Ranker

Principais métodos:

- **orderResults()**: este método é responsável por ordenar o resultado da pesquisa, vinda do módulo Searcher.

5 Bibliotecas externas

- *The Porter stemming algorithm*: http://snowball.tartarus.org/

6 Estrutura do código

A estrutura do nosso projeto Maven encontra-se organizada da seguinte forma:

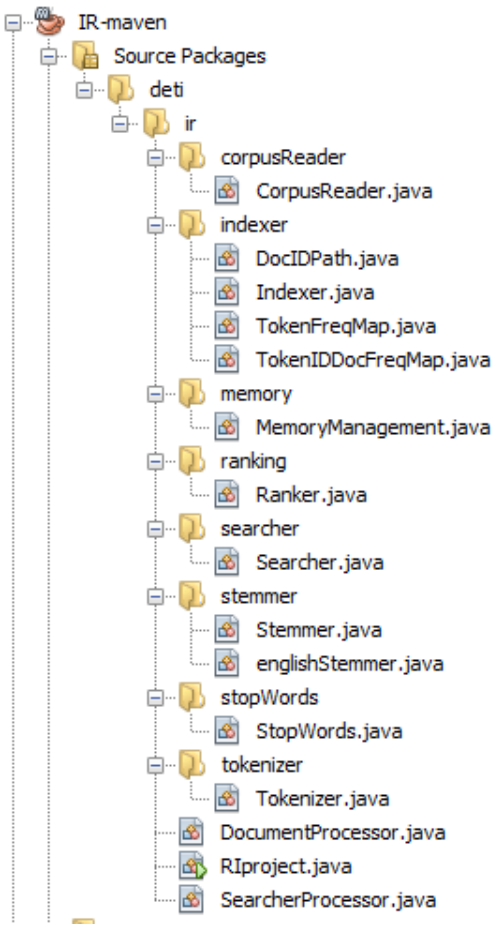

Figura 5: Estrutura do código

Na raiz do projeto Maven é possível encontrar três diretórios:

- `files-data`: pasta onde se encontram os corpus utilizados. Para esta fase do projeto apenas foi utilizado o `corpus-sample`. Também nesta pasta existe um ficheiro `stopwords_en.txt` que contém todas as stopwords.
- `outputs`: Nesta pasta são colocados os ficheiros resultantes do processo de indexação, com o nome `TokenFreqDocs.txt`

- `jars`: Nesta pasta estão colocados as bibliotecas compiladas necessárias para este projeto (Para este caso foi usado o jar do Snowball).

7 Execução

- Compilar e executar o nosso projeto recorrendo ao Netbeans.
- Por padrão, o nosso projeto irá gerar um ficheiro em `outputs` com os dados indexados.

8 Resultados

Para obtermos os tempos que demorou o processo de indexeção e respectiva escrita em ficheiro, foi utilizado o seguinte hardware:

- **Processador**: Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz, 2401 Mhz, 4 Núcleo(s), 8 Processador(es) Lógico(s)
- **Memória**: 8.00 GB

Para o corpus fornecido `sample_corpus.zip` obtivemos os seguintes tempos:

Teste	**Tempo de execução (s)**
1	3.438
2	3.359
3	3.379
4	3.367

Tabela 1: Tempos obtidos

O tempo médio de execução foi 3,373 segundos.

Os ficheiro de output onde é possível observar o resultado da indexação tem o seguinte formato:

`token;frequencia_com_que_token_ocorre;ID_Document1,ID_Document2,ID_Document3...`

Um exemplo apresenta-se a seguir:

`maxillofaci;3;119047674,111922015,112523235`

9 Repositório de desenvolvimento

https://github.com/ruipoliveira/IR-engine

10 Conclusões

Chegado ao final deste relatório, é nossa intenção efetuar uma retrospetiva da evolução do mesmo, tendo em conta os problemas com que nos deparámos, e principais conclusões retiradas.

Nesta iteração do trabalho, procedemos à implementação baseada na modulação que foi efetuada na iteração anterior, bem como à descrição mais detalhada de cada classe e seus métodos, uma vez que esta se tornou mais fácil depois de iniciarmos a implementação.

Apesar de não termos a certeza da modulação da iteração anterior, pensamos que esta tenha sido bem efetuada, uma vez que este projeto se encontrar funcional.

Durante a realização deste projeto, consideramos que entendemos os fundamentos do armazenamento e recuperação de informação. Com a modulação anteriormente efetuada, podemos concluir que esta foi uma mais valia para iniciarmos a implementação e percebermos todos os processamentos envolvidos.