Mestrado integrado em Engenharia de Computadores e Telemática

Departamento de Electrónica, Telecomunicações e informática

2015/2016

Recuperação de Informação

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

GRUPO 04

Autores:

- Bernardo Ferreira (67413)
- Bruno Silva (68535)

Responsável:

- Prof. Sérgio Matos

26/10/2016

Índice

1. Descrição das Classes

Nesta Secção do documento vai ser realizada a descrição das classes usadas na implementação do motor de indexação booleana.

1.1. Classes Principais

1.1.1. Corpus Reader

O corpus reader é a classe responsável por percorrer um diretório recursivamente e devolver a lista de todos ficheiros e sua extensão, existentes em todo o diretório e respectivos subdiretórios.

Esta classe conta conta com os seguintes atributos:

- String[] ignoreExtensions - array que contém extensões de documentos que queiramos ignorar no processo.
- ArrayList<CorpusFiles> files - Array list que contém cada ficheiro e respetiva extensão presente no diretório.

Os métodos para esta classe são os seguintes

- readDir(String path) - método que lê o diretório fornecido no argumento e cria uma lista de ficheiros (CorpusFiles) com todos nos diretórios e subdiretórios.
- ArrayList<CorpusFile> getFiles() - Retorna lista de todos os ficheiros.
- CorpusFile getNextFile() - Retorna o próximo ficheiro, ou null caso não existam mais.

1.1.2. Document Processor

O document processor é a classe responsável por processar o documento consoante a sua extensão, retirar tags específicas do formato e criar e guardar um objeto da classe Document numa Array List.

Esta classe conta conta com os seguintes atributos:

- ArrayList<Documents> documents - Lista de todos os documentos presentes nos ficheiros válidos encontrados pelo CorpusReader.

Os métodos para esta classe são os seguintes

- processDocument(CorpusFile cfile) - método que vê a extensão do ficheiro recebido e envia para a função interna de processamento adequada à sua extensão.
- Document getNextDocument() - Função onde as threads aguardam a existência de novos documentos a serem processados.
- finishedProcess() - Função que notifica que já todos os documentos foram processados.

- String getDocumentContent(Document doc) - Funçao que retorna o conteúdo de um determinado documento.
- writeDocuments() - Função para escrever a correspondência entre o nosso id interno de documento com a localização real desse documento.

1.1.3. Tokenizer

O tokenizer é a classe responsável por receber o conteúdo do Documento processado e aplicar transformações para se obter uma lista de palavras cujo o resultado será as formas simples de cada palavra(passar para letra minúscula, retirar pontuação, aplicar transformações linguísticas etc.). Esta classe é instanciada para cada documento processado, permitindo assim, a paralelização na tokenização de documentos.

Esta classe conta conta com os seguintes atributos:

- ArrayList<String> tokens - Lista de todos os token presentes num documento.
- englishStemmer stemmer - Stemmer de termos para os transformar na sua forma simples.
- String [] stopWordsList - Array de stop words recebido no construtor desta classe.

Os métodos para esta classe são os seguintes:

- tokenize(String content, Document doc) - método que recebe o conteúdo de um documento, separa em termos, e aplica transformações, chamando para cada um deles a função transform. Por fim, guarda cada palavra(token) numa Array List de tokens.
- String transform(String term) - método que converte uma palavra para minúsculas e aplica o steamming e stopword filtering etc.

1.1.4. Indexer

O Indexer é a classe responsável por indexar cada palavra presente nos documento fornecidos. Esta classe pode ser instanciada para cada documento de modo a paralelizar o processo de indexação.

Esta classe conta conta com os seguintes atributos:

- Index index - Estrutura de dados para suportar o Índice que vai ser gerado a partir dos documentos.

Os métodos para esta classe são os seguintes

- indexToken(String[] tokens, int docId) - método para indexar todas as palavras(tokens) de um documento.
- writeIndex() - método para escrever para um ficheiro o index ordenado.

1.1.5. Searcher

O Searcher é a classe responsável pela pesquisa no Index por termos ou expressões e retorna quais os documentos onde as mesmas se encontram. Para esta entrega, esta classe não é necessária, não tendo, portanto, sofrido alterações.

Esta classe conta conta com os seguintes atributos:

- Index index - Estrutura de dados que suporta o Índice que foi gerado a partir dos documentos.

Os métodos para esta classe são os seguintes

- Dictionary searchTerm() - método para pesquisar por um termo que retorna um Dictionary.

1.2. Classes Auxiliares

1.2.1. CorpusFile

O CorpusFile é a classe responsável por guardar informação de cada ficheiro encontrado pelo CorpusReader.

Esta classe conta conta com os seguintes atributos:

- String path - Path do ficheiro encontrado.
- String extension - Extensão do ficheiro encontrado.

Esta classe contém getters para todos os seus atributos.

1.2.2. Document

O Document é a classe responsável por guardar toda a informação relativa a documento(localização, id, etc).

Esta classe conta conta com os seguintes atributos:

- static String tmpPath - Path temporário para a localização do ficheiro com o conteúdo do documento.
- static Int id - contador do número de documentos.
- Int docId - Id do documento.
- Int docStartLine - Linha em que o documento começa dentro do ficheiro.
- String filePath - Localização do ficheiro onde se encontra o documento.

Esta classe contém getters para todos os seus atributos.

1.2.3. Index

O Index é a classe responsável pela estrutura de dados usada para a indexação de palavras presentes nos documentos.

Esta classe conta conta com os seguintes atributos:

- HashMap<Integer, Dictionary> - Estrutura de dados do Índice.
- SortedSet<String> words - Estrutura de dados para armazenar as palavras do índice por ordem alfabética de modo a facilitar a impressão do mesmo.

Os métodos para esta classe são os seguintes:

- addTerm(String term, int doc) - método para adicionar um termo no índice em que aparece no documento. Se já existir, não cria uma entrada nova, mas sim, adiciona o documento à Posting List.
- Dictionary searchTerm(String term) - método para pesquisar por um termo que retorna um Dictionary.
- removeTerm(String term) - método para remover um termo do Índice.
- removeDocument(String Term, int doc) - método para remover um documento da posting list respetiva daquele termo.

1.2.4. PostingList

Classe que guarda a lista dos documentos em que um termo aparece.

Esta classe conta conta com os seguintes atributos:

- ArrayList<Integer> docs - Estrutura de dados do Posting.

Os métodos para esta classe são os seguintes:

- addPosting(int doc) - método para adicionar um novo posting.
- int postingListSize() - método para retornar o tamanho da lista dos postings.
- removePosting(int doc) - método para remover um posting.

Esta classe contém getters para todos os seus atributos.

1.2.5. Dictionary

Classe que guarda a informação do Índice, por exemplo, os termos, o número de documentos em que o termo aparece e um ponteiro para a lista de Postings.

Esta classe conta conta com os seguintes atributos:

- String term - Termo presente no dicionário.
- int nDocs - Número de documentos onde um determinado termo aparece.

- Posting postingList - Posting onde é guardada a lista dos documentos em que um termo aparece.

Os métodos para esta classe são os seguintes:

- addDocument(int doc) - método para adicionar um documento à posting list.
- removeDocument(int doc) - método para remover um documento à posting list.
- Dictionary getTerm() - método para retornar informações sobre o próprio termo.
- PostingList getPostingList() - método para retornar uma PostingList.

1.2.6. ArffProcessor

Classe que processa o tipo de ficheiro '.arff', processando corretamente todas as tags específicas deste tipo de ficheiro.

Os métodos para esta classe são os seguintes:

- ArrayList<Document> identify(CorpusFile file) - Método que para cada CorpusFile, separa o ficheiro em documentos.
- String process(Document doc) - Método para processar um Documento que teve origem num '.arff', retirando as tags específicas deste tipo de ficheiro.

1.3. Threads

1.3.1. DP_Worker

Esta thread será responsável por processar os documentos presentes num ficheiro. As threads deste tipo ficam a escuta de novos CorpusFiles identificados pelo CorpusReader para identificar os documentos presentes no mesmo.

1.3.2. TI_Worker

Esta thread será responsável por tokenizar e indexar cada palavra presente num documento. As threads deste tipo ficam a escuta de documentos identificados previamente, e tokenizam e aplicam as transformações necessárias a cada termo. De seguida, cada termo é indexado.

2. Processo de Tokenização

O processo de Tokenização, que escolhemos para a nossa implementação, assenta, primeiramente, nas seguintes regras aplicadas ao conteúdo todo:

- Os símbolos '-', '(' e ')' são substituídos por espaços, devido à ocorrência frequente neste corpus de várias palavras ligadas por hífens (Ex:

Eu-anthracene-9-carboxylic). A utilização de parênteses sem espaços antes ou depois, faria com que a palavra dentro de parêntesis fosse concatenada à palavra imediatamente antes(ou depois).
- Fazer split a um ou mais espaços consecutivos, de modo a obter um array com todas as palavras do documento(tokens).

De seguida são aplicadas as seguintes regras a cada palavra(token), antes de ser indexada:

- Passagem da palavra para minúsculas.
- Retirar todos os símbolos alfanuméricos (Ex: U.S.A -> USA).
- Retirar palavras de apenas um caracter.
- Retirar palavras presentes na lista de stop-words.
- Fazer o stemming das palavras.

O nosso processo de tokenização, indexa números do tipo '0.0001%' como 00001, de modo a que se esta percentagem for pesquisada, o resultado apontar apenas para os documentos que contéem exatamente '0.0001%' e não para todos os que contéem o número 1(Que seria o caso se guardássemos '0.0001%' como '1' no índice). No caso de um número ser seguido de unidades (Ex. '0.01L/s'), este é indexado como 001ls.

3. Data Flow

O nosso Information Retrieval Engine, contém uma main que é responsável por instanciar as classes principais referidas acima.

Para a execução deste programa são utilizadas threads de dois tipos, DP_Workers, responsáveis por processar os ficheiros presentes no diretório dos corpus, e TI_Workers, responsáveis por tokenizar e indexar cada documento. O programa inicia-se com uso do CorpusReader para a identificação dos vários ficheiros presentes nos diretórios (CorpusFile's) que, conforme são identificados, vão sendo processados pelas várias threads DP_Workers, ou seja, cada thread irá identificar os vários documentos de um ficheiro(consoante o tipo de ficheiro), e criar um Document para cada um. Conforme estes vão sendo criados, as threads do tipo TI_Workers vão consumindo cada documento, aplicado várias regras de tokenização, e indexando cada token.

No fim, é imprimida a lista de documentos e o index ordenado alfabeticamente.

4. Análise de resultados

Neste trabalho, fizemos primeiramente uma implementação single threaded, porém não ficamos satisfeitos com os resultados e decidimos utilizar threads para paralelizar a execução de algumas partes do programa. Observámos uma melhoria substancial no tempo de execução do programa, como é possível verificar nas imagens abaixo:

Imagens de 3 execuções para melhor estimar valores de execução:

As imagens abaixo são referentes à execução do código num ambiente multi thread com 10 threads de cada tipo descrito em cima em execução. Nestas execuções o corpus usado foi o “corpus-RI” disponivel na pagina do elearning da cadeira.

As 3 imagens seguintes mostram os valores de tempo de execução para um ambiente “single thread” em que só existe 1 thread de cada tipo. O corpus usado foi o mesmo que anteriormente.

```
Finished Reading Directory: 0.035 seconds.
Finished Processed Documents: 0.516 seconds.
Finished Indexing: 30.529 seconds.
Finished writing index to file: 31.2 seconds.
```

Olhando para os diferentes resultados podemos ver que a implementação multi thread consegue valores inferiores a metade do tempo mantendo os requisitos necessários de ordenação dos documentos na posting list.

Todos estes testes foram realizados num Macbook Pro 15 (mid 2012) e podem variar consoante o computador onde foram executados.

5. Execução do programa

Para executar corretamente o programa, é necessário, da primeira vez, fazer clean and build e só depois executar. Na main, existem ainda 3 parâmetros configuráveis: número de threads do tipo DP_Worker (default 10), número de threads do tipo TI_Worker (default 10) e diretório do corpus a ser indexado.

A execução deste programa gera dois ficheiros: um index.txt, que contém as palavras presentes no índice, número de ocorrências e os documentos em que essa palavra está presente, e um documento doc_dict.txt que contém o mapeamento entre os docID internos ao nosso programa e os ficheiros originais e identificador onde este se encontra.

6. Anexos

Por motivos de visualização, o diagrama de classes encontra-se num ficheiro à parte.