UFRJ - Departamento de Ciência da Computação / IM
MAB605 – Recuperação da Informação – 2019/1

André Queiroz e Ingrid Canaane

Teste de um Sistema de Recuperação de Informação: Apache Lucene

Rio de Janeiro
2019

Introdução

O Lucene (hoje em sua versão 8.0.0) é uma biblioteca Java disponível em código aberto, sob o domínio da Apache Foundation. Suas APIs focam principalmente na indexação e consulta de textos (full text search). Suporta, além de Java, outras linguagens como Perl, Python, Ruby, C++, .NET (sendo necessário para estas o uso dos ports do Lucene).

As duas etapas principais são: indexação e pesquisa. A primeira processa os dados originais e gera uma estrutura de dados eficiente para a pesquisa baseada em palavras-chave. A segunda consulta o índice pelas palavras digitadas numa consulta, e então ordena os resultados pela similaridade do texto com a consulta.

Instalação e uso

Nesse projeto, usamos a versão do Lucene original, em Java. Logo, precisamos do Java Development Kit (JDK) instalado na máquina. Como ambiente de desenvolvimento (IDE), utilizamos o Eclipse. Após baixar e instalar ambos os softwares, baixamos a biblioteca do Lucene, também no seu site oficial. A biblioteca vem como um arquivo compactado, contendo vários arquivos .jar.

No Eclipse, criamos um novo projeto. Clicamos com o botão direito no projeto na aba “Packet Explorer” e clicamos em “Propriedades”. Em “Java Build Path”, clique em “Add External JARs…” e importe os seguintes arquivos .jar, a partir da pasta descompactada da biblioteca Lucene baixada no site oficial:

/queryparser/lucene-queryparser-8.0.0.jar
/analysis/common/lucene-analyzers-common-8.0.0.jar
/core/lucene-core-8.0.0.jar
/demo/lucene-demo-8.0.0.jar

Com isso o Lucene já estará funcionando no projeto.

Scoring

O Lucene implementa vários modelos de recuperação de informação, como Booleano, Vector Space Model (VSM), modelos probabilísticos, entre outros e é possível selecionar qualquer um desses para uso.

Entretanto, o modelo usado como padrão é o Okapi BM25, implementado na classe BM25Similarity, utilizando parâmetros k1 = 1.2 e b = 0.75.

```
1. FSP950117-034 5.2097573
5.2097573 = sum of:
  1.307858 = weight(title:cris in 10044) [BM25Similarity], result of:
    1.307858 = score(freq=1.0), product of:
      2.9199939 = idf, computed as log(1 + (N - n + 0.5) / (n + 0.5)) from:
        5604 = n, number of documents containing term
        103913 = N, total number of documents with field
      0.44789752 = tf, computed as freq / (freq + k1 * (1 - b + b * dl / avgdl)) from:
        1.0 = freq, occurrences of term within document
        1.2 = k1, term saturation parameter
        0.75 = b, length normalization parameter
        216.0 = dl, length of field (approximate)
        208.43752 = avgdl, average length of field
  3.9018993 = weight(title:energetic in 10044) [BM25Similarity], result of:
    3.9018993 = score(freq=3.0), product of:
      5.5051293 = idf, computed as log(1 + (N - n + 0.5) / (n + 0.5)) from:
        422 = n, number of documents containing term
        103913 = N, total number of documents with field
      0.7087752 = tf, computed as freq / (freq + k1 * (1 - b + b * dl / avgdl)) from:
        3.0 = freq, occurrences of term within document
        1.2 = k1, term saturation parameter
        0.75 = b, length normalization parameter
        216.0 = dl, length of field (approximate)
        208.43752 = avgdl, average length of field
```

Exemplo da pesquisa: “Crises energéticas”, utilizando a função explain(Query, doc) da classe IndexSearcher, que nos possibilita entender exatamente como foi computado o score da query naquele documento.

Tokenização do texto

O primeiro passo para a realização das consultas é a tokenização do texto. No Lucene, fazemos isso pela classe Analyzer. Os analisadores são os responsáveis por processar os dados de texto e convertê-los em tokens armazenados no índice. O Lucene acompanha vários analisadores integrados, que diferem na forma como tokenizam o texto e aplicam os filtros.

É possível também criar analisadores customizados usando os blocos de construção básicos fornecidos pelo Lucene. Como o idioma do texto é Português, usamos o PortugueseAnalyzer, que já é implementado com um arquivo default de stopwords em português, mas um dos construtores desse analisador permite que seja passado um conjunto personalizado de stopwords.

```
27        PortugueseAnalyzer analyzer = new PortugueseAnalyzer();
28        //StandardAnalyzer analyzer = new StandardAnalyzer();
29
```

Indexação do Texto

Os tokens gerados são incluídos como termos no índice do Lucene, e o IndexWriter é uma classe que cria ou mantém um índice. Em seu construtor devem ser passados um analisador (o que é usado para tokenizar os dados) e um diretório (que representa o local onde os arquivos de índice serão armazenados).

```
39        IndexWriter w = new IndexWriter(new RAMDirectory(), new IndexWriterConfig(analyzer));
```

Adição de documentos ao índice

Os arquivos para teste estão em formato SGML. Criamos uma Classe SGMLtoObject que analisa todos os arquivos SGML de um caminho dado e converte para um ArrayList da classe Doc, que possui campos id e text, representando cada um dos documentos encontrados nos arquivos SGML.

Depois disso, os documentos são adicionados ao índice pelo método addDoc.

```
37          ArrayList<Doc> documentos =
38                  SGMLtoObject.toDoc("C:\\dev\\colecao_teste");
39          // TEM QUE MODIFICAR ISSO AQUI PRO PATH DA SUA COLECAO
40
41          for (Doc d1 : documentos) {
42              addDoc(w, d1.getText(), d1.getId());
43          }
```

Análise da Consulta

QueryParser é a classe utilizada para analisar cadeias/expressões de consultas inseridas pelo usuário em um objeto de consulta do Lucene (Query),que pode ser passado para o método de procura do IndexSearcher. Ele converte internamente uma cadeia de consulta inserida em uma das subclasses de consulta concretas. É possível construir consultas booleanas textualmente, usando os operadores AND, OR e NOT.

```
52          Query q = new QueryParser("title", analyzer).parse(querystr);
```

Consulta

O IndexSearcher permite procurar índices armazenados em um diretório, representado por um IndexReader. O método search retorna uma ordenação dos documentos (classe TopDocs) que correspondem a uma consulta conforme sua classificação pelas pontuações calculadas. O ScoreDoc é um ponteiro simples para um documento contido nos resultados de procura, englobando a posição do documento no índice e a pontuação calculada pelo Lucene.

```
55          int hitsPerPage = 10;
56          IndexReader reader = DirectoryReader.open(index);
57          IndexSearcher searcher = new IndexSearcher(reader);
58          TopDocs docs = searcher.search(q, hitsPerPage);
59          ScoreDoc[] hits = docs.scoreDocs;
```

Experimento: Consulta na Coleção de Teste

Não conseguimos que retornassem muitos resultados refinando a consulta com operações booleanas, então colocamos todas as consultas como o título da consulta. Os resultados das consultas (100 hits) estão no arquivo resultados.txt. e pode ser simulado usando o código contido neste repositório.