Congresso de Inovação Com Tecnologias Open Source

Introdução ao Apache Lucene: inserindo busca eficiente ao seu projeto de software

Instrutora: Katia Cristina Lage dos Santos
Centro Federal de Educação Tecnológica de Minas Gerais

Tópicos do Mini-curso

CEFET-MG

1. O que é uma máquina de busca
2. Principais termos da recuperação de informação
3. Extração de metadados XML utilizando a API JDOM
4. Conhecendo e customizando a API Lucene
5. Como fica na minha aplicação?
6. Utilização do Componente de Busca em um Aplicativo Java Desktop
7. Outras perspectivas de utilização do componente de busca desenvolvido

Katia Cristina Lage dos Santos possui graduação e mestrado em Ciência da Computação pela Universidade Federal de Minas Gerais (UFMG). Atualmente é Professora Efetiva do Ensino Básico, Técnico e Tecnológico do Centro Federal de Educação Tecnológica de Minas Gerais (Cefet/MG), campus VIII, localizado em Varginha. Em termos acadêmicos, tem especial interesse nas seguintes áreas da Ciência da Computação: Engenharia de Software, Banco de Dados, Bibliotecas Digitais e Recuperação de Informação.

Contatos: katia.lage@gmail.com

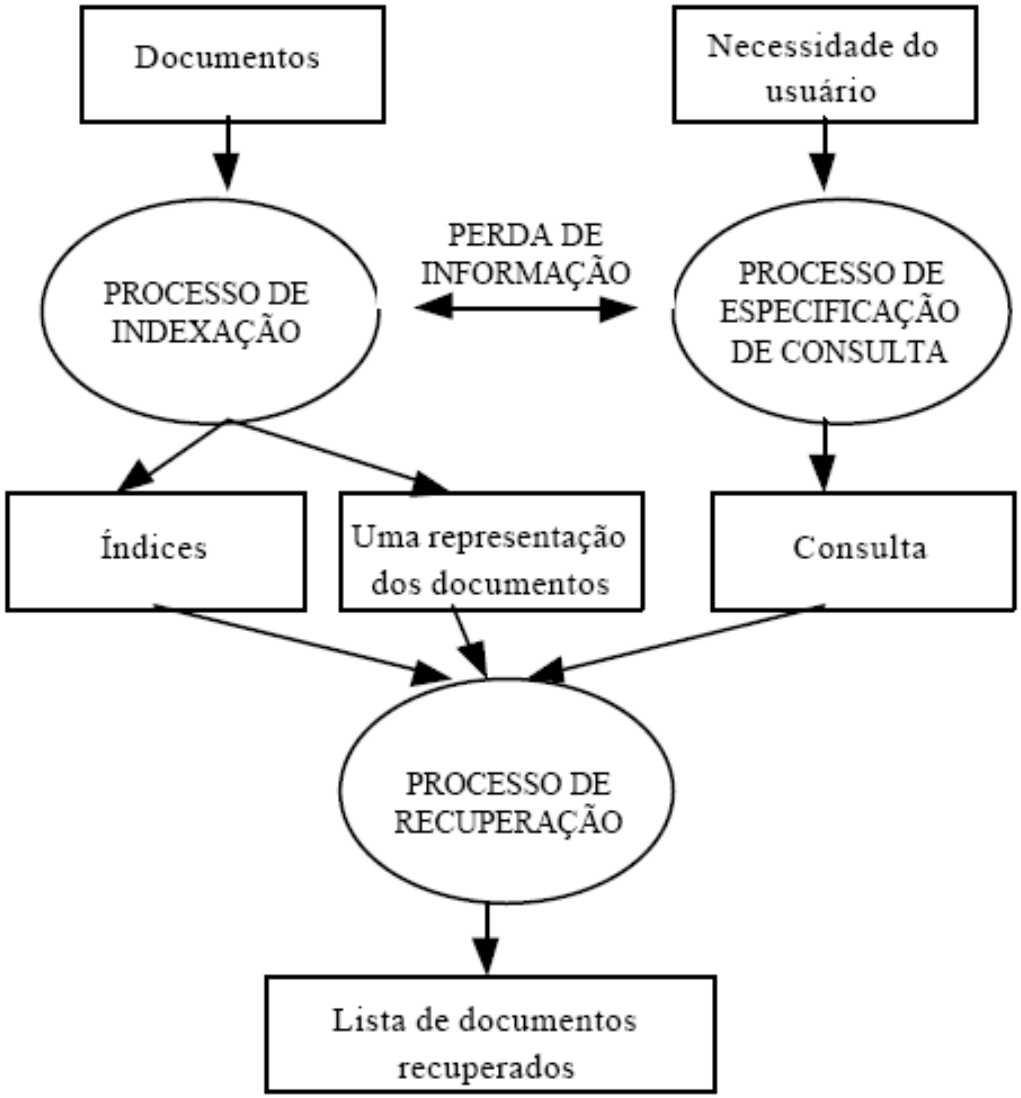
{twitter, facebook}katialage

Máquinas de Busca

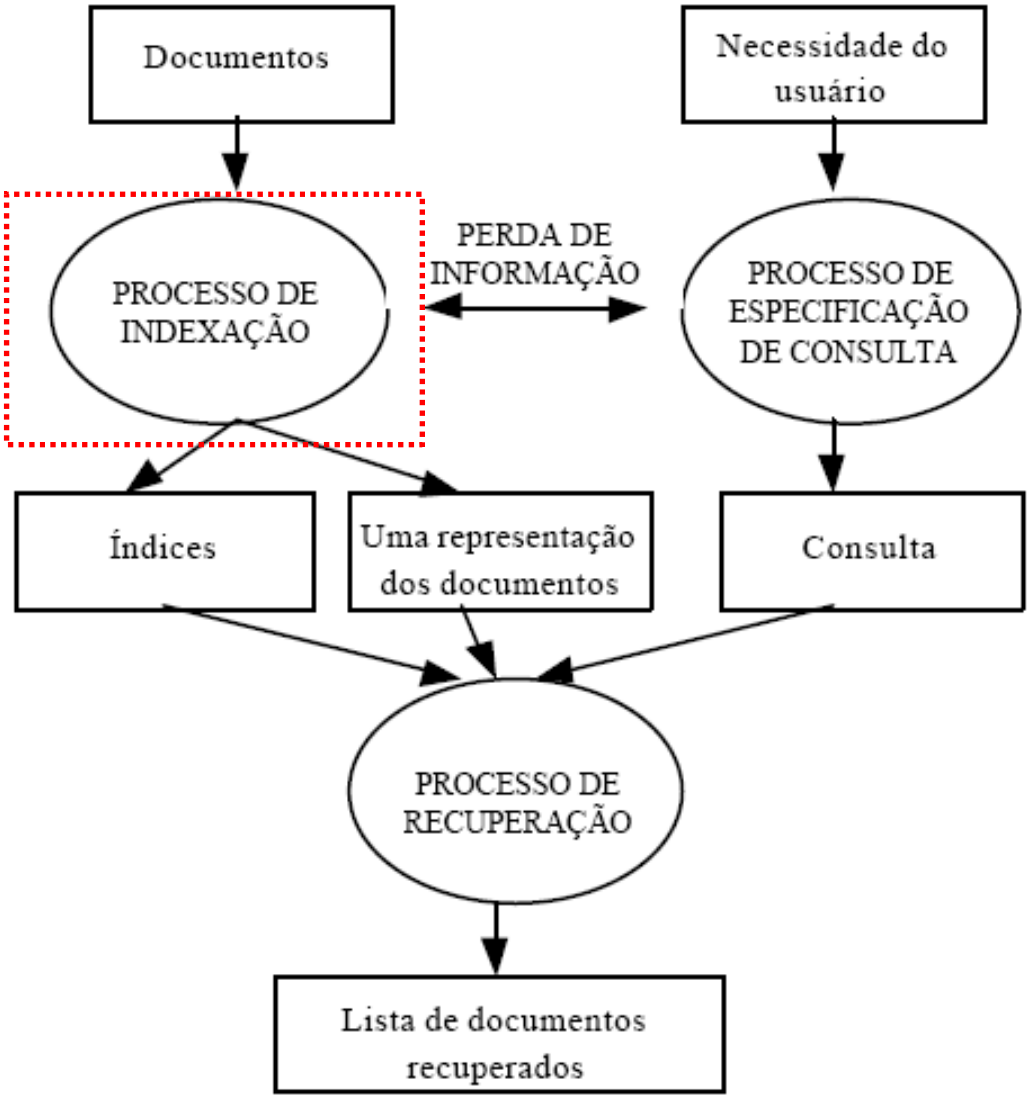
Máquinas de Busca

Máquina de Busca Textual

Máquinas de Busca



Indexação


Documentos
Necessidade do usuário
PROCESSO DE INDEXAÇÃO
PERDA DE INFORMAÇÃO
PROCESSO DE ESPECIFICAÇÃO DE CONSULTA
Índices
Uma representação dos documentos
Consulta
PROCESSO DE RECUPERAÇÃO
Lista de documentos recuperados

Lista Invertida

TEXTO 1

...Java é uma linguagem de programação...

TEXTO 2

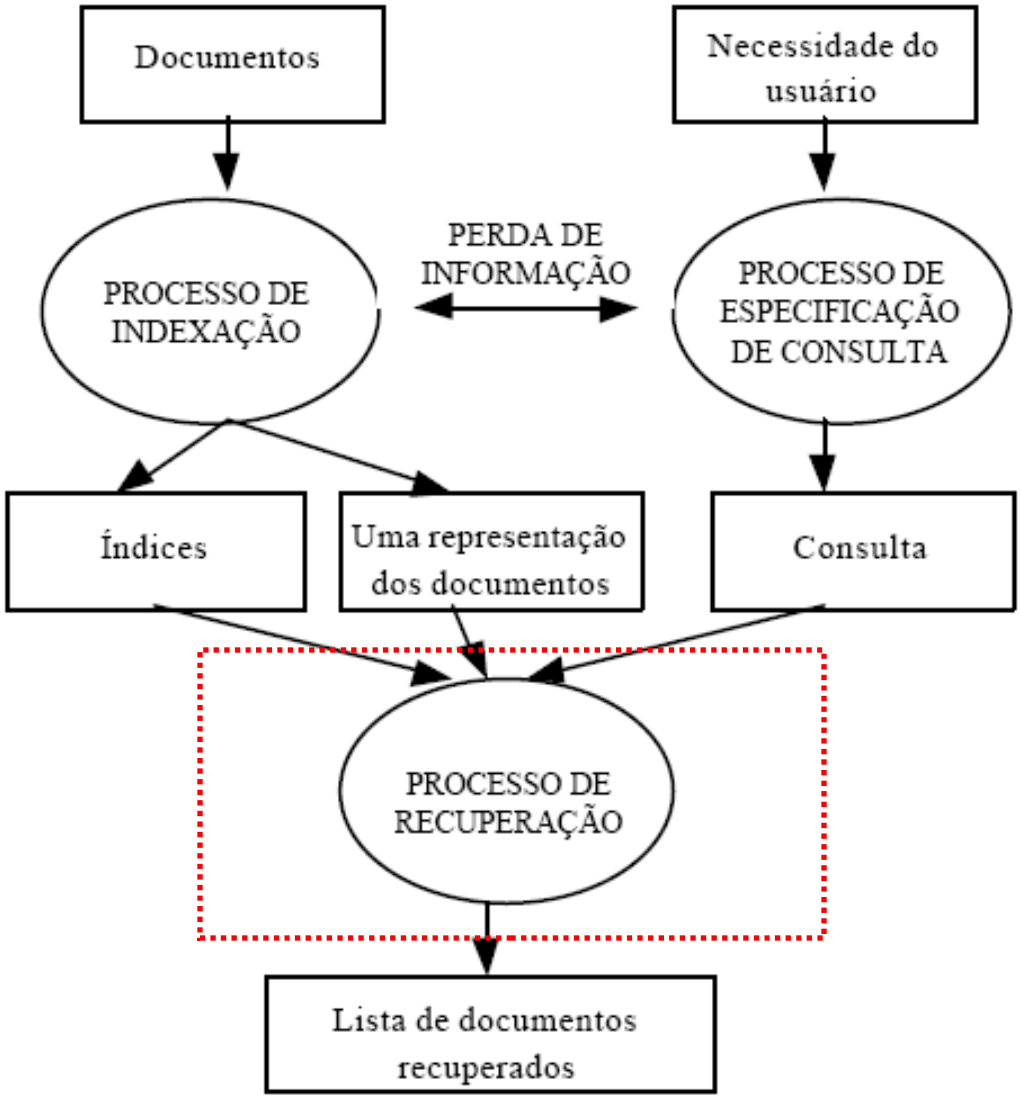
...na ilha de Java o café é uma das...

TEXTO 3

...Esse ano a exportação do café brasileiro...

Índice de palavras	Referência
A	TEXTO3
BRASILEIRO	TEXTO3
CAFÉ	TEXTO2, TEXTO3
ESTE	TEXTO3
EXPORTAÇÃO	TEXTO3
ILHA	TEXTO2
JAVA	TEXTO1, TEXTO2
LINGUAGEM	TEXTO1
PROGRAMAÇÃO	TEXTO1

Métodos de Recuperação



Métodos de Recuperação

- Modelo Booleano
  - As consultas são uma combinação dos termos através dos operadores AND, OR e NOT
  - PESQUISA POR: “café brasileiro”
  - Algoritmo:
    - Localizar ocorrência de “café”
    - Recuperar os resultados
    - Localizar ocorrência de “brasileiro”
    - Recuperar os resultados
    - “Combinar os dois resultados”

Métodos de Recuperação

TEXTO 1

...Java é uma linguagem de programação...

TEXTO 2

...na ilha de Java o café é uma das...

TEXTO 3

...Esse ano a exportação do café brasileiro...

Índice de palavras	Referência
A	TEXTO3
BRASILEIRO	TEXTO3
CAFÉ	TEXTO2, TEXTO3
ESTE	TEXTO3
EXPORTAÇÃO	TEXTO3
ILHA	TEXTO2
JAVA	TEXTO1, TEXTO2
LINGUAGEM	TEXTO1
PROGRAMAÇÃO	TEXTO1

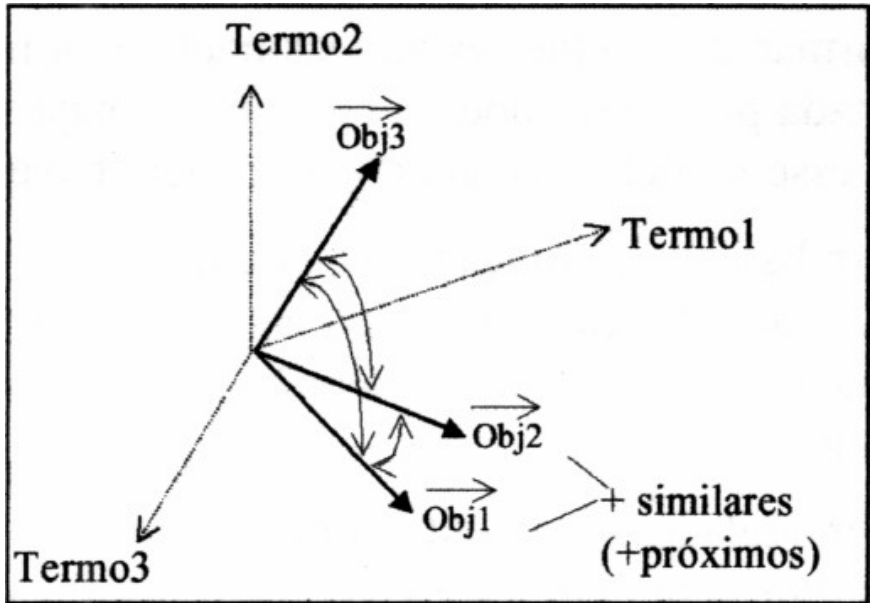
Métodos de Recuperação

- Modelo Vetorial

$freq\ (k,S)$ → TF

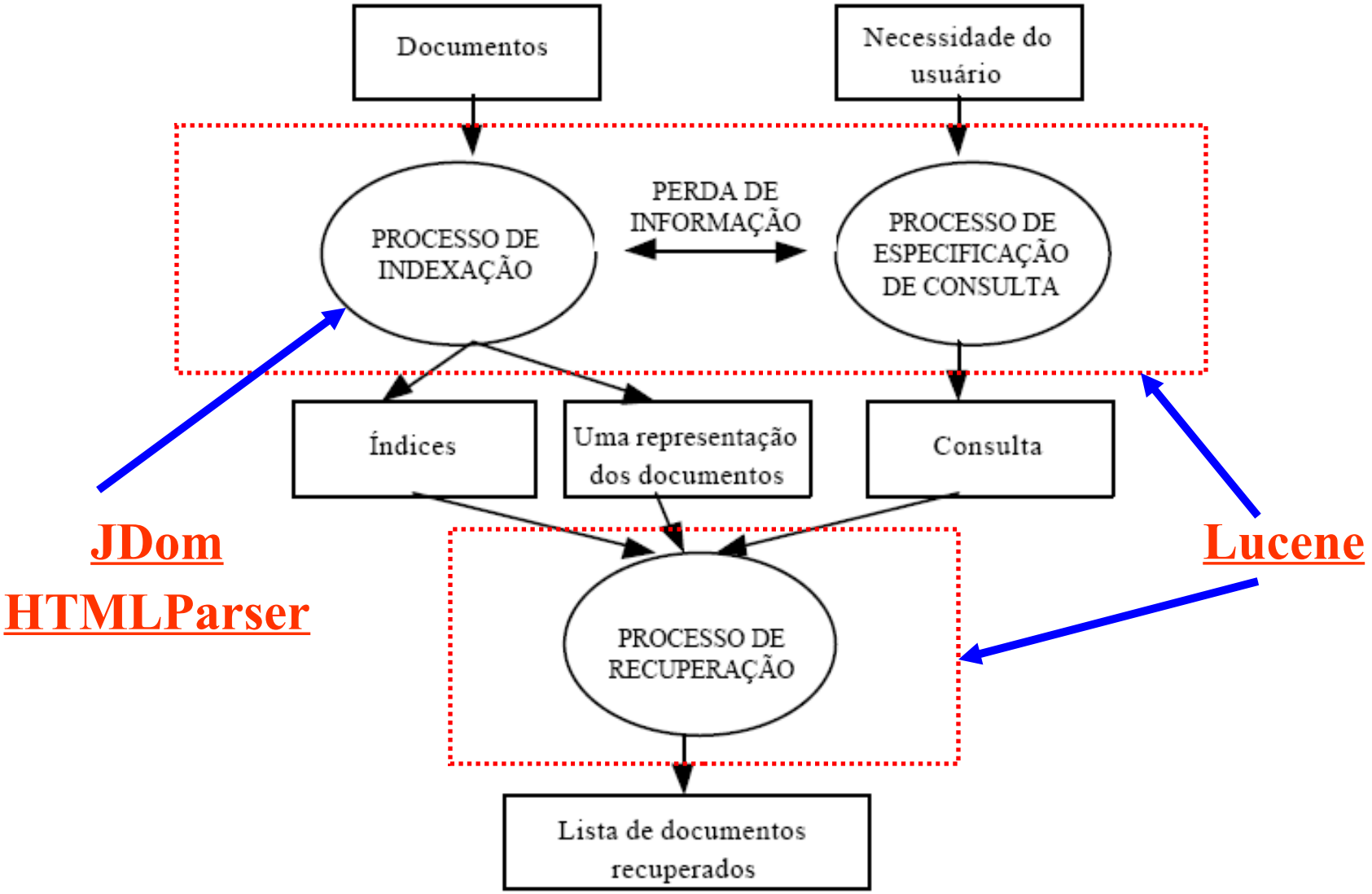
$\log\ (N / n_k)$ → IDF

$$freq\ (k,S) \times \log\ (N / n_k)$$

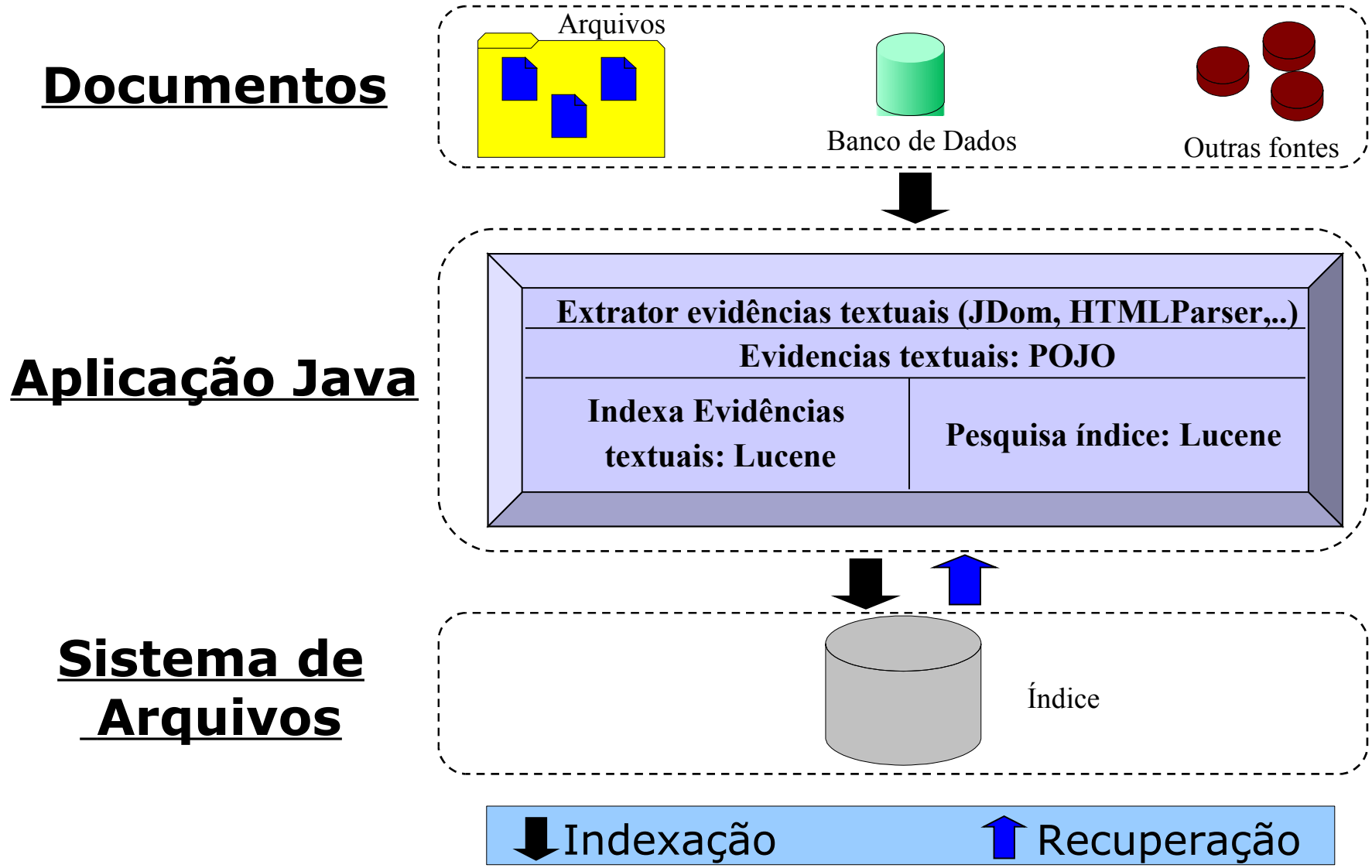


$$\text{similaridade (Q,D)} = \frac{\sum_{k=1}^{n} w_{qk} \cdot w_{dk}}{\sqrt{\sum_{k=1}^{n} (w_{qk})^2 \cdot \sum_{k=1}^{n} (w_{dk})^2}}$$

Máquina de Busca + APIs Java



Como fica na minha solução?



Prática 1

- Criar um projeto no Eclipse
- Criar camadas (aplicação, controle, entidade e fronteira)
- Criar pasta lib para colocar os .jar utilizados

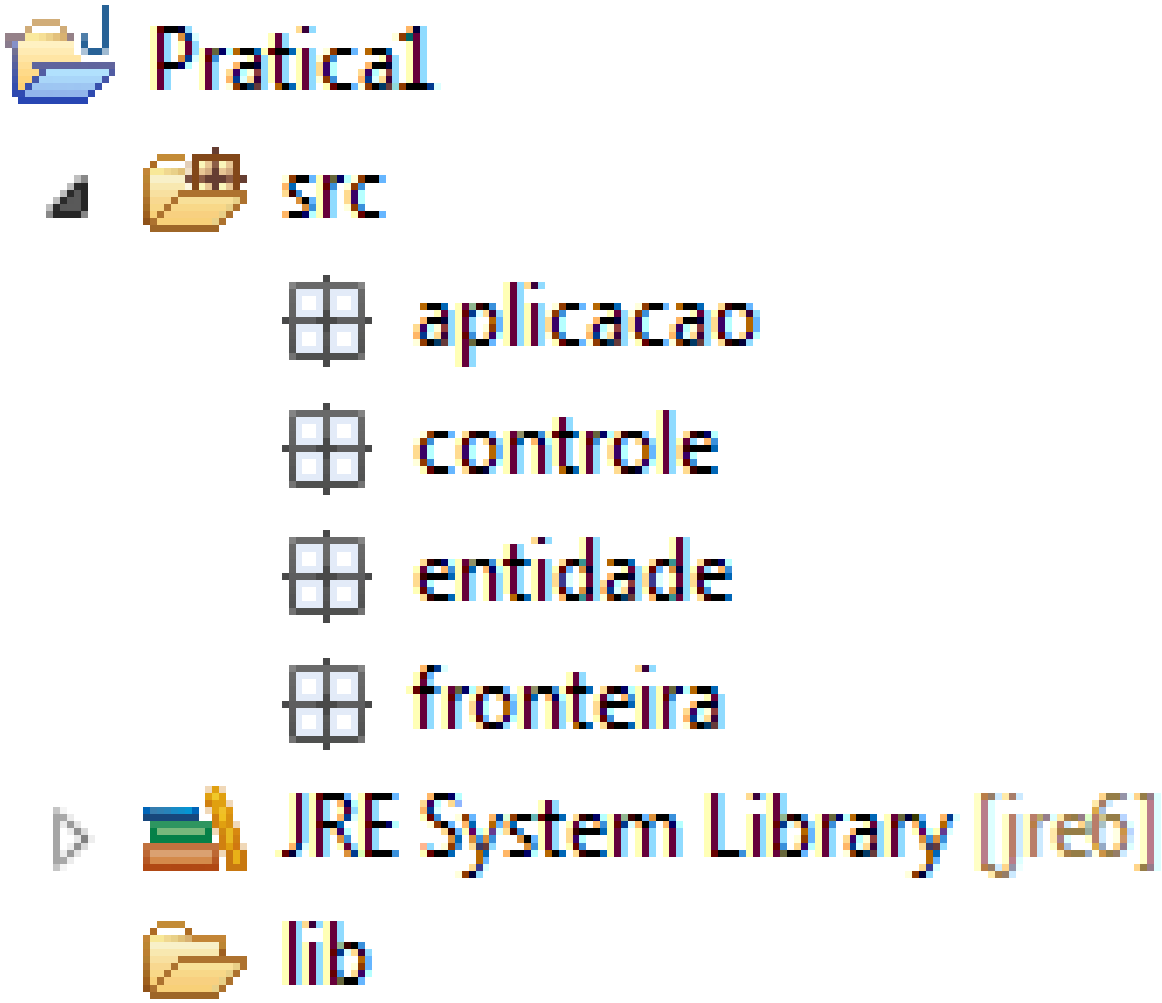
Prática 1

- Criar um projeto no Eclipse
- Criar camadas (aplicação, controle, entidade e fronteira)
- Criar pasta lib para colocar os .jar utilizados

Prática 1

- Criar um projeto no Eclipse
- Criar camadas (aplicação, controle, entidade e fronteira)
- Criar pasta lib para colocar os .jar utilizados

Prática 1


Pratica1
src
aplicacao
controle
entidade
fronteira
JRE System Library [jre6]
lib

JDom

jdom.org

- Biblioteca *opensource* Java;
- Realiza o *parser* de documentos XML;

```
<metadados-customizados>
    <tipo>livro</tipo>
    <titulo>Java, como programar</titulo>
    <autores> Deitel & Deitel</autores>
    <palavras-chave>Java, programação, informática</palavras-chave>
</metadados-customizados>
```

Prática 2

- Copiar amostras XML da coleção de teste
- Criar interface (entidade/intf) e a classe concreta da entidade (entidade/xml)que representa os dados do negócio a serem indexados
- Baixar a API JDom
- Incluir como lib do projeto Java
- Criar interface (controle/extrairEvidencias/intf) e concreta do Controlador que lerá os dados dos arquivos XML a serem indexados com Jdom (controle/extrairEvidencias/xml)

Prática 2

```
<custom-metadata>
    <CNPJ-Remetente>78349506543298</CNPJ-Remetente>
    <uf-Remetente>MG</uf-Remetente>
    <Endereco-Remetente>Avenida Presidente Antonio Carlos, 1200.
Pampulha. Belo Horizonte/MG</Endereco-Remetente>
    <Razao-Social-Remetente>Embalagens Gerais Ltda</Razao-Social-
Remetente>
    <codOperacao>03984</codOperacao>
    <nomeProduto>Embalagem para marmitex</nomeProduto>
    <dhEmissao>2010-10-25</dhEmissao>
    <CNPJ-Destinatario>8690498439094</CNPJ-Destinatario>
    <uf-Destinatario>SP</uf-Destinatario>
    <Endereco-Destinatario>Avenida São João, 721. Centro. São
Paulo/SP</Endereco-Destinatario>
    <Razao-Social-Destinatario>Restaurante Comer Bem</Razao-Social-
Destinatario>
</custom-metadata>
```

Prática 2

- Copiar amostras XML da coleção de teste
- Criar interface (entidade/intf) e a classe concreta da entidade (entidade/xml)que representa os dados do negócio a serem indexados
- Baixar a API JDom
- Incluir como lib do projeto Java
- Criar interface (controle/extrairEvidencias/intf) e concreta do Controlador que lerá os dados dos arquivos XML a serem indexados com Jdom (controle/extrairEvidencias/xml)

Prática 2

EvidenciasTextuais.java

```
package entidade.intf;

public interface EvidenciasTextuais {

}
```

Prática 2

EvidenciasNFe.java

```
package entidade.xml;

import entidade.intf.EvidenciasTextuais;

public class EvidenciasNFe implements EvidenciasTextuais{

    private String cnpjRemetente;
    private String ufRemetente;
    private String endRemetente;
    private String razaoSocialRemetente;
    private String codOperacao;
    private String nomeProduto;
    private String dhEmissao;
    private String cnpjDestinatario;
    private String ufDestinatario;
    private String endDestinatario;
    private String razaoSocialDestinatario;
```

Prática 2

- Copiar amostras XML da coleção de teste
- Criar interface (entidade/intf) e a classe concreta da entidade (entidade/xml)que representa os dados do negócio a serem indexados
- Baixar a API JDom
- Incluir como lib do projeto Java
- Criar interface (controle/extrairEvidencias/intf) e concreta do Controlador que lerá os dados dos arquivos XML a serem indexados com Jdom (controle/extrairEvidencias/xml)

Prática 2

- Copiar amostras XML da coleção de teste
- Criar interface (entidade/intf) e a classe concreta da entidade (entidade/xml)que representa os dados do negócio a serem indexados
- Baixar a API JDom
- Incluir como lib do projeto Java
- Criar interface (controle/extrairEvidencias/intf) e concreta do Controlador que lerá os dados dos arquivos XML a serem indexados com Jdom (controle/extrairEvidencias/xml)

Prática 2

- Copiar amostras XML da coleção de teste
- Criar interface (entidade/intf) e a classe concreta da entidade (entidade/xml)que representa os dados do negócio a serem indexados
- Baixar a API JDom
- Incluir como lib do projeto Java
- Criar interface (controle/extrairEvidencias/intf) e concreta do Controlador que lerá os dados dos arquivos XML a serem indexados com Jdom (controle/extrairEvidencias/xml)

Prática 2

ParserArquivosDados.java

```
package controle.extrairEvidencias.intf;

import java.io.File;

public interface ParserArquivosDados {

    public void extrairEvidencias(File diretorioBaseEvidencias);

    public EvidenciasTextuais getEvidencia();

}
```

Prática 2

*ParserArquivosXML.java

```
package controle.extrairEvidencias.xml;

import java.io.File;

public class ParserArquivosXML implements ParserArquivosDados{

    private EvidenciasNFe evidencias;

    public ParserArquivosXML() {
        evidencias = new EvidenciasNFe();
    }

    public void extrairEvidencias(File caminhoArquivoXML){   }

    @Override
    public EvidenciasTextuais getEvidencia() {
        return evidencias;
    }
```

Prática 2

ParserArquivosXML.java

```
public void extrairEvidencias(File caminhoArquivoXML)
{
    SAXBuilder builder = new SAXBuilder();
    Document documentoXML = null;
    Element raiz = null;
    try {
        documentoXML = builder.build(caminhoArquivoXML);
        raiz = documentoXML.getRootElement();
        evidencias.setCnpjRemetente(raiz.getChild("CNPJ-Remetente").
                getValue());
    } catch (JDOMException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Lucene

- Biblioteca *opensource*;
- Implementada em várias linguagens, como C, Java, C#
- Viabiliza a indexação e pesquisa textual;
- Bem documentada
  - API da Lucene
  - Libs/lucene/lucene-2.9.3/lucene-2.9.3/docs/api/core/index.html

INDEXAÇÃO

```
try {
SimpleFSDirectory dirDocsColecao =
                    new SimpleFSDirectory(diretorioIndice);
IndexWriter writer = new IndexWriter(dirDocsColecao,
analyzer, novo, IndexWriter.MaxFieldLength.UNLIMITED);
// cria um documento
Document document  = new Document();
document.add(new Field(CAMPO_INF_NUM_IMG,
evidenciasTextuais.getNumeroImagensPaginaWeb(),
Field.Store.YES, Field.Index.NO));
writer.addDocument(document);
writer.close();                                    (...)
```

INDEXAÇÃO

```
try {
SimpleFSDirectory dirDocsColecao =
                    new SimpleFSDirectory(diretorioIndice);
IndexWriter writer = new IndexWriter(caminhoIndice,
analyzer, novo, IndexWriter.MaxFieldLength.UNLIMITED);
// cria um documento
Document document  = new Document();
document.add(new Field(CAMPO_INF_NUM_IMG,
evidenciasTextuais.getNumeroImagensPaginaWeb(),
Field.Store.YES, Field.Index.NO));
writer.addDocument(document);
writer.close();                                    (...)
```

org.apache.lucene.store

Class Directory

java.lang.Object
  └ org.apache.lucene.store.Directory

Direct Known Subclasses:

FileSwitchDirectory, FSDirectory, RAMDirectory

```
public abstract class Directory
extends Object
```

A Directory is a flat list of files. Files may be written once, when they are created. Once a file is created it may only be opened for read, or deleted. Random access is permitted both when reading and writing.

Java's i/o APIs not used directly, but rather all i/o is through this API. This permits things such as:

- implementation of RAM-based indices;
- implementation indices stored in a database, via JDBC;
- implementation of an index as a single file;

Directory locking is implemented by an instance of LockFactory, and can be changed for each Directory instance using setLockFactory(org.apache.lucene.store.LockFactory).

---

org.apache.lucene.store

Class FSDirectory

java.lang.Object
  └ org.apache.lucene.store.Directory
    └ org.apache.lucene.store.FSDirectory

Direct Known Subclasses:

MMapDirectory, NIOFSDirectory, SimpleFSDirectory

```
public class FSDirectory
extends Directory
```

Base class for Directory implementations that store index files in the file system. There are currently three core subclasses:

- `SimpleFSDirectory` is a straightforward implementation using java.io.RandomAccessFile. However, it has poor concurrent performance (multiple threads will bottleneck) as it synchronizes when multiple threads read from the same file.
- `NIOFSDirectory` uses java.nio's FileChannel's positional io when reading to avoid synchronization when reading from the same file. Unfortunately, due to a Windows-only Sun JRE bug this is a poor choice for Windows, but on all other platforms this is the preferred choice.
- `MMapDirectory` uses memory-mapped IO when reading. This is a good choice if you have plenty of virtual memory relative to your index size, eg if you are running on a 64 bit JRE, or you are running on a 32 bit JRE but your index sizes are small enough to fit into the virtual memory space. Java has currently the limitation of not being able to unmap files from user code. The files are unmapped, when GC releases the byte buffers. Due to this bug in Sun's JRE, MMapDirectory's

CEFET-MG

INDEXAÇÃO

```
try {
SimpleFSDirectory dirDocsColecao =
                    new SimpleFSDirectory(diretorioIndice);
IndexWriter writer = new IndexWriter(caminhoIndice,
analyzer, novo, IndexWriter.MaxFieldLength.UNLIMITED);
// cria um documento
Document document  = new Document();
document.add(new Field(CAMPO_INF_NUM_IMG,
evidenciasTextuais.getNumeroImagensPaginaWeb(),
Field.Store.YES, Field.Index.NO));
writer.addDocument(document);
writer.close();                                    (...)
```

CEFET-MG

org.apache.lucene.analysis

Class Analyzer

java.lang.Object
  └ org.apache.lucene.analysis.Analyzer

Direct Known Subclasses:

KeywordAnalyzer, PerFieldAnalyzerWrapper, SimpleAnalyzer, StandardAnalyzer, StopAnalyzer, WhitespaceAnalyzer

---

```
public abstract class Analyzer
extends Object
```

An Analyzer builds TokenStreams, which analyze text. It thus represents a policy for extracting index terms from text.

Typical implementations first build a Tokenizer, which breaks the stream of characters from the Reader into raw Tokens. One or more TokenFilters may then be applied to the output of the Tokenizer.

org.apache.lucene.analysis.standard

Class StandardAnalyzer

```
java.lang.Object
  └ org.apache.lucene.analysis.Analyzer
      └ org.apache.lucene.analysis.standard.StandardAnalyzer
```

```
public class StandardAnalyzer
extends Analyzer
```

Filters `StandardTokenizer` with `StandardFilter`, `LowerCaseFilter` and `StopFilter`, using a list of English stop words.

You must specify the required `Version` compatibility when creating StandardAnalyzer:

- As of 2.9, StopFilter preserves position increments
- As of 2.4, Tokens incorrectly identified as acronyms are corrected (see LUCENE-1608

Version:

$Id: StandardAnalyzer.java 829134 2009-10-23 17:18:53Z mikemccand $

CEFET-MG

StandardAnalyzer

```
public StandardAnalyzer(Version matchVersion,
                        File stopwords)
                 throws IOException
```

Builds an analyzer with the stop words from the given file.

Parameters:

`matchVersion` - Lucene version to match See above

`stopwords` - File to read stop words from

Throws:

`IOException`

See Also:

`WordlistLoader.getWordSet(File)`

StandardAnalyzer

```
public StandardAnalyzer(Reader stopwords)
                 throws IOException
```

Deprecated. *Use `StandardAnalyzer(Version, Reader)` instead*

Builds an analyzer with the stop words from the given reader.

Throws:

`IOException`

getWordSet

```
public static HashSet getWordSet(File wordfile)
                          throws IOException
```

Loads a text file and adds every line as an entry to a HashSet (omitting leading and trailing whitespace). Every line of the file should contain only one word. The words need to be in lowercase if you make use of an Analyzer which uses LowerCaseFilter (like StandardAnalyzer).

Parameters:

`wordfile` - File containing the wordlist

Returns:

A HashSet with the file's words

Throws:

`IOException`

INDEXAÇÃO

```
try {
SimpleFSDirectory dirDocsColecao =
                    new SimpleFSDirectory(diretorioIndice);
IndexWriter writer = new IndexWriter(caminhoIndice,
analyzer, novo, IndexWriter.MaxFieldLength.UNLIMITED);
// cria um documento
Document document   = new Document();
document.add(new Field(CAMPO_INF_NUM_IMG,
evidenciasTextuais.getNumeroImagensPaginaWeb(),
Field.Store.YES, Field.Index.NO));
writer.addDocument(document);
writer.close();                                        (...)
```

INDEXAÇÃO

```
try {
SimpleFSDirectory dirDocsColecao =
                    new SimpleFSDirectory(diretorioIndice);
IndexWriter writer = new IndexWriter(caminhoIndice,
analyzer, novo, IndexWriter.MaxFieldLength.UNLIMITED);
// cria um documento
Document document  = new Document();
document.add(new Field(CAMPO_INF_NUM_IMG,
evidenciasTextuais.getNumeroImagensPaginaWeb(),
Field.Store.YES, Field.Index.NO));
writer.addDocument(document);
writer.close();                                          (...)
```

CEFET-MG

Field Summary	
static Field.Index	ANALYZED Index the tokens produced by running the field's value through an Analyzer.
static Field.Index	ANALYZED_NO_NORMS Expert: Index the tokens produced by running the field's value through an Analyzer, and also separately disable the storing of norms.
static Field.Index	NO Do not index the field value.
static Field.Index	NO_NORMS **Deprecated.** *This has been renamed to* NOT_ANALYZED_NO_NORMS
static Field.Index	NOT_ANALYZED Index the field's value without using an Analyzer, so it can be searched.
static Field.Index	NOT_ANALYZED_NO_NORMS Expert: Index the field's value without an Analyzer, and also disable the storing of norms.
static Field.Index	TOKENIZED **Deprecated.** *this has been renamed to* ANALYZED
static Field.Index	UN_TOKENIZED **Deprecated.** *This has been renamed to* NOT_ANALYZED

RECUPERAÇÃO

```
try {
    SimpleFSDirectory dirDocsColecao =
       new SimpleFSDirectory(indicePaginasWebColecao);
    boolean somenteLeitura = true;
    IndexSearcher isearcher =
      new IndexSearcher(dirDocsColecao, somenteLeitura);
    QueryParser parser =
     new QueryParser(Version.LUCENE_29,"nome",analyzer);
    Query query =
        parser.parse("nome"+":\""+strConsulta+"\"");


                                                  (...)
```

RECUPERAÇÃO

```
(...)
    TopDocs topDocs = isearcher.search(query, 5);
    ScoreDoc[] hits = topDocs.scoreDocs;

    List<Document> listaDocumentosRecuperados =
        new ArrayList<Document>();

    for (int i = 0; i < hits.length; i++) {
        int docId = hits[i].doc;
        Document d = isearcher.doc(docId);
        listaDocumentosRecuperados.add(d);
    }
}// fecha bloco try

                                                    (...)
```

RECUPERAÇÃO

```
try {
    SimpleFSDirectory dirDocsColecao =
        new SimpleFSDirectory(indicePaginasWebColecao);
    boolean somenteLeitura = true;
    IndexSearcher isearcher =
      new IndexSearcher(dirDocsColecao, somenteLeitura);
    QueryParser parser =
     new QueryParser(Version.LUCENE_29,"nome",analyzer);
    Query query =
        parser.parse("nome"+":\""+strConsulta+"\"");


                                                (...)
```

RECUPERAÇÃO

```
try {
    SimpleFSDirectory dirDocsColecao =
       new SimpleFSDirectory(indicePaginasWebColecao);
    boolean somenteLeitura = true;
    IndexSearcher isearcher =
      new IndexSearcher(dirDocsColecao, somenteLeitura);
    QueryParser parser =
     new QueryParser(Version.LUCENE_29,"nome",analyzer);
    Query query =
        parser.parse("nome"+":\""+strConsulta+"\"");


                                                  (...)
```

Lucene

RECUPERAÇÃO

```
try {
    SimpleFSDirectory dirDocsColecao =
       new SimpleFSDirectory(indicePaginasWebColecao);
    boolean somenteLeitura = true;
    IndexSearcher isearcher =
      new IndexSearcher(dirDocsColecao, somenteLeitura);
    QueryParser parser =
     new QueryParser(Version.LUCENE_29,"nome",analyzer);
    Query query =
        parser.parse("nome"+":\""+strConsulta+"\"");


                                                  (...)
```

RECUPERAÇÃO

```
try {
    SimpleFSDirectory dirDocsColecao =
       new SimpleFSDirectory(indicePaginasWebColecao);
    boolean somenteLeitura = true;
    IndexSearcher isearcher =
      new IndexSearcher(dirDocsColecao, somenteLeitura);
    QueryParser parser =
     new QueryParser(Version.LUCENE_29,"nome",analyzer);
    Query query =
        parser.parse("nome"+":\""+strConsulta+"\"");

                                                    (...)
```

CEFET-MG

org.apache.lucene.queryParser

Class QueryParser

java.lang.Object
  └ org.apache.lucene.queryParser.QueryParser

All Implemented Interfaces:

QueryParserConstants

Direct Known Subclasses:

MultiFieldQueryParser

```
public class QueryParser
extends Object
implements QueryParserConstants
```

This class is generated by JavaCC. The most important method is `parse(String)`. The syntax for query strings is as follows: A Query is a series of clauses. A clause may be prefixed by:

- a plus (+) or a minus (-) sign, indicating that the clause is required or prohibited respectively; or
- a term followed by a colon, indicating the field to be searched. This enables one to construct queries which search multiple fields.

A clause may be either:

- a term, indicating all the documents that contain this term; or
- a nested query, enclosed in parentheses. Note that this may be used with a +/- prefix to require any of a set of terms.

Thus, in BNF, the query grammar is:

```
Query  ::= ( Clause )*
Clause ::= ["+", "-"] [<TERM> ":"] ( <TERM> | "(" Query ")" )
```

Examples of appropriately formatted queries can be found in the query syntax documentation.

```
title:"The Right Way" AND text:go
```

or

```
title:"Do it right" AND right
```

Since text is the default field, the field indicator is not required.

Note: The field is only valid for the term that it directly precedes, so the query

```
title:Do it right
```

Will only find "Do" in the title field. It will find "it" and "right" in the default field (in this case the text field).

Wildcard Searches

Lucene supports single and multiple character wildcard searches within single terms (not within phrase queries).

To perform a single character wildcard search use the "?" symbol.

To perform a multiple character wildcard search use the "*" symbol.

The single character wildcard search looks for terms that match that with the single character replaced. For example, to search for "text" or "test" you can use the search:

```
te?t
```

Multiple character wildcard searches looks for 0 or more characters. For example, to search for test, tests or tester, you can use the search:

```
test*
```

You can also use the wildcard searches in the middle of a term.

```
te*t
```

Note: You cannot use a * or ? symbol as the first character of a search.

org.apache.lucene.search

Class WildcardQuery

java.lang.Object
  └ org.apache.lucene.search.Query
    └ org.apache.lucene.search.MultiTermQuery
      └ org.apache.lucene.search.WildcardQuery

All Implemented Interfaces:

Serializable, Cloneable

```
public class WildcardQuery
extends MultiTermQuery
```

Implements the wildcard search query. Supported wildcards are *, which matches any character sequence (including the empty one), and ?, which matches any single character. Note this query can be slow, as it needs to iterate over many terms. In order to prevent extremely slow WildcardQueries, a Wildcard term should not start with one of the wildcards * or ?.

This query uses the `MultiTermQuery.CONSTANT_SCORE_AUTO_REWRITE_DEFAULT` rewrite method.

See Also:

`WildcardTermEnum`, Serialized Form

RECUPERAÇÃO

```
try {
    SimpleFSDirectory dirDocsColecao =
       new SimpleFSDirectory(indicePaginasWebColecao);
    boolean somenteLeitura = true;
    IndexSearcher isearcher =
      new IndexSearcher(dirDocsColecao, somenteLeitura);
    QueryParser parser =
     new QueryParser(Version.LUCENE_29,"nome",analyzer);
    Query query =
        parser.parse("nome"+":\""+strConsulta+"\"");

                                                    (...)
```

RECUPERAÇÃO

```
(...)
    TopDocs topDocs = isearcher.search(query, 5);
    ScoreDoc[] hits = topDocs.scoreDocs;

    List<Document> listaDocumentosRecuperados =
        new ArrayList<Document>();

    for (int i = 0; i < hits.length; i++) {
        int docId = hits[i].doc;
        Document d = isearcher.doc(docId);
        listaDocumentosRecuperados.add(d);
    }
}// fecha bloco try

                                                    (...)
```

CEFET-MG

search

```
public TopDocs search(Query query,
                      int n)
               throws IOException
```

Finds the top `n` hits for `query`.

Throws:

`BooleanQuery.TooManyClauses`

`IOException`

search

```
public void search(Query query,
                   Collector results)
            throws IOException
```

Lower-level search API.

`Collector.collect(int)` is called for every matching document.

Applications should only use this if they need *all* of the matching documents. The high-level search API (`search(Query, int)`) is usually more efficient, as it skips non-high-scoring hits.

Note: The `score` passed to this method is a raw score. In other words, the score will not necessarily be a float whose value is between 0 and 1.

Throws:

`BooleanQuery.TooManyClauses`

`IOException`

CEFET-MG

Class Collector

java.lang.Object
└ org.apache.lucene.search.Collector

Direct Known Subclasses:
HitCollectorWrapper, PositiveScoresOnlyCollector, TimeLimitingCollector, TopDocsCollector

---

```
public abstract class Collector
extends Object
```

Expert: Collectors are primarily meant to be used to gather raw results from a search, and implement sorting or custom result filtering, collation, etc.

As of 2.9, this class replaces the deprecated HitCollector, and offers an API for efficient collection of hits across sequential IndexReaders. IndexSearcher advances the collector through each of the sub readers, in an arbitrary order. This results in a higher performance means of collection.

Lucene's core collectors are derived from Collector. Likely your application can use one of these classes, or subclass TopDocsCollector, instead of implementing Collector directly:

- TopDocsCollector is an abstract base class that assumes you will retrieve the top N docs, according to some criteria, after collection is done.
- TopScoreDocCollector is a concrete subclass TopDocsCollector and sorts according to score + docID. This is used internally by the IndexSearcher search methods that do not take an explicit Sort. It is likely the most frequently used collector.
- TopFieldCollector subclasses TopDocsCollector and sorts according to a specified Sort object (sort by field). This is used internally by the IndexSearcher search methods that take an explicit Sort.
- TimeLimitingCollector, which wraps any other Collector and aborts the search if it's taken too much time, will subclass Collector in 3.0 (presently it subclasses the deprecated HitCollector).
- PositiveScoresOnlyCollector wraps any other Collector and prevents collection of hits whose score is <= 0.0

Collector decouples the score from the collected doc: the score computation is skipped entirely if it's not needed. Collectors that do need the score should implement the setScorer(org.apache.lucene.search.Scorer) method, to hold onto the passed Scorer instance, and call Scorer.score() within the collect method to compute the current hit's score. If your collector may request the score for a single hit multiple times, you should use ScoreCachingWrappingScorer.

RECUPERAÇÃO

```
(...)
    TopDocs topDocs = isearcher.search(query, 5);
    ScoreDoc[] hits = topDocs.scoreDocs;

    List<Document> listaDocumentosRecuperados =
        new ArrayList<Document>();

    for (int i = 0; i < hits.length; i++) {
        int docId = hits[i].doc;
        Document d = isearcher.doc(docId);
        listaDocumentosRecuperados.add(d);
    }
}// fecha bloco try

                                                (...)
```

Apache Lucene - Scoring

PDF

- Introduction
- Scoring
  - Fields and Documents
  - Score Boosting
  - Understanding the Scoring Formula
  - The Big Picture
  - Query Classes
  - Changing Similarity
- Changing your Scoring -- Expert Level
- Appendix
  - Algorithm

Introduction

Lucene scoring is the heart of why we all love Lucene. It is blazingly fast and it hides almost all of the complexity from the user. In a nutshell, it works. At least, that is, until it doesn't work, or doesn't work as one would expect it to work. Then we are left digging into Lucene internals or asking for help on java-user@lucene.apache.org to figure out why a document with five of our query terms scores lower than a different document with only one of the query terms.

While this document won't answer your specific scoring issues, it will, hopefully, point you to the places that can help you figure out the what and why of Lucene scoring.

Lucene scoring uses a combination of the Vector Space Model (VSM) of Information Retrieval and the Boolean model to determine how relevant a given Document is to a User's query. In general, the idea behind the VSM is the more times a query term appears in a document relative to the number of times the term appears in all the documents in the collection, the more relevant that document is to the query. It uses the Boolean model to first narrow down the documents that need to be scored based on the use of boolean logic in the Query specification. Lucene also adds some capabilities and refinements onto this model to support boolean and fuzzy searching, but it essentially remains a VSM based system at the heart. For some valuable references on VSM and IR in general refer to the Lucene Wiki IR references.

The rest of this document will cover Scoring basics and how to change your Similarity. Next it will cover ways you can customize the Lucene internals in Changing your Scoring -- Expert Level which gives details on implementing your own Query class and related functionality. Finally, we will finish up with some reference material in the Appendix.

Prática 3

- Baixar a API Lucene
- Incluir como lib do projeto Java
- Criar uma interface (controle/indexar/intf) e uma classe concreta referente ao Controlador (controle/indexar) para indexar os dados de negócio escolhido
- Criar uma interface (controle/recuperar/intf) e uma classe concreta referente ao Controlador (controle/recuperar/xml) para recuperar os dados com base no que foi indexado anteriormente

Prática 3

- Baixar a API Lucene
- Incluir como lib do projeto Java
- Criar uma interface (controle/indexar/intf) e uma classe concreta referente ao Controlador (controle/indexar) para indexar os dados de negócio escolhido
- Criar uma interface (controle/recuperar/intf) e uma classe concreta referente ao Controlador (controle/recuperar/xml) para recuperar os dados com base no que foi indexado anteriormente

Prática 3

- Baixar a API Lucene
- Incluir como lib do projeto Java
- Criar uma interface (controle/indexar/intf) e uma classe concreta referente ao Controlador (controle/indexar/xml) para indexar os dados de negócio escolhido
- Criar uma interface (controle/recuperar/intf) e uma classe concreta referente ao Controlador (controle/recuperar/xml) para recuperar os dados com base no que foi indexado anteriormente

Prática 3

IndexadorLucene.java

```
package controle.indexar.intf;

import org.apache.lucene.analysis.Analyzer;


public interface IndexadorLucene {

    public abstract void indexarDocumento(String caminhoIndiceInvertido,
                                          EvidenciasTextuais evidenciasTextuais,
                                          Analyzer analyzer,
                                          boolean boolCriaNovoIndice);

}
```

Prática 3

IndexadorLuceneXML.java

```
public class IndexadorLuceneXML implements IndexadorLucene{

    @Override
    public void indexarDocumento(String caminhoIndiceInvertido,
            EvidenciasTextuais evidenciasTextuais, Analyzer analyzer,
            boolean boolCriaNovoIndice) {
        IndexWriter writer = null;
        Document document = null;
        SimpleFSDirectory dirDocsColecao = null;

        try {
            dirDocsColecao = new SimpleFSDirectory(new File(caminhoIndiceInvertido));
            writer = new IndexWriter(dirDocsColecao, analyzer, boolCriaNovoIndice,  IndexWriter.M

            // cria um documento
            document  = new Document();

            // armazena o caminho completo da página HTML
            document.add(new Field("cnpj-remetente", ((EvidenciasNFe)evidenciasTextuais).getCnpjR
                    Field.Store.YES, Field.Index.ANALYZED, Field.TermVector.NO));

            // adiciona documento no índice
            writer.addDocument(document);
```

Prática 3

- Baixar a API Lucene
- Incluir como lib do projeto Java
- Criar uma interface (controle/indexar/intf) e uma classe concreta referente ao Controlador (controle/indexar/xml) para indexar os dados de negócio escolhido
- Criar uma interface (controle/recuperar/intf) e uma classe concreta referente ao Controlador (controle/recuperar/xml) para recuperar os dados com base no que foi indexado anteriormente

Prática 3

RecuperadorLucene.java

```
package controle.recuperar.intf;

import java.util.List;

import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.document.Document;

public interface RecuperadorLucene {

    public List<Document> recuperaTermoIndiceInvertido(
        String caminhoIndiceInvertido,
        String strConsulta,
        Analyzer analyzer);

}
```

Prática 3

RecuperadorLuceneXML.java

```
public List<Document> recuperaTermoIndiceInvertido(
        String caminhoIndiceInvertido, String strConsulta, Analyzer analyzer) {

    List<Document>  listaDocumentosRecuperados = null;
    try {
        SimpleFSDirectory dirDocsColecao = new SimpleFSDirectory(new File(caminhoIndiceInvert
        boolean somenteLeitura = true;
        IndexSearcher isearcher = new IndexSearcher(dirDocsColecao, somenteLeitura);
        QueryParser parser = new QueryParser(Version.LUCENE_29, "textoCompleto", analyzer);
        Query query = parser.parse("textoCompleto" + ":\""+strConsulta+"\"");

        TopDocs topDocs = isearcher.search(query, 5);
        ScoreDoc[] hits = topDocs.scoreDocs;

        listaDocumentosRecuperados = new ArrayList<Document>();

        for (int i = 0; i < hits.length; i++) {
            int docId = hits[i].doc;
            Document d = isearcher.doc(docId);
            listaDocumentosRecuperados.add(d);
        }

    } catch (CorruptIndexException e) {
        e.printStackTrace();
```

Prática 4

- Baixar a API FormsLayout
- Incluir como lib do projeto Java
- Copiar classes da visão – interface gráfica
- Criar enum TipoDocumento em controle/constantes
- Fazer correções nas classes da visão em função do nome das outras classes
- Criar classe principal com método main (aplicacao)

Prática 4

- Baixar a API FormsLayout
- Incluir como lib do projeto Java
- Copiar classes da visão – interface gráfica
- Criar enum TipoDocumento em controle/constantes
- Fazer correções nas classes da visão em função do nome das outras classes
- Criar classe principal com método main (aplicacao)

Prática 4

- Baixar a API FormsLayout
- Incluir como lib do projeto Java
- Copiar classes da visão – interface gráfica
- Criar enum TipoDocumento em controle/constantes
- Fazer correções nas classes da visão em função do nome das outras classes
- Criar classe principal com método main (aplicacao)

Prática 4

- Baixar a API FormsLayout
- Incluir como lib do projeto Java
- Copiar classes da visão – interface gráfica
- Criar enum TipoDocumento em controle/constantes
- Fazer correções nas classes da visão em função do nome das outras classes
- Criar classe principal com método main (aplicacao)

Prática 4

TipoDocumento.java

```
package controle.constantes;

public enum TipoDocumento {

    XMLNFe("XML da nota fiscal eletrônica",
           "colecaoTeste/xml/amostras",
           "colecaoTeste/xml/indice");

    private String descricao, caminhoAmostras, caminhoIndice;

    TipoDocumento(String descricao, String caminhoAmostras, String caminhoIndice) {
        this.descricao = descricao;
    }

    public String getDescricao()
    {
        return descricao;
    }

    public String getCaminhoAmostras()
    {
        return caminhoAmostras;
    }
```

Prática 4

- Baixar a API FormsLayout
- Incluir como lib do projeto Java
- Copiar classes da visão – interface gráfica
- Criar enum TipoDocumento em controle/constantes
- Fazer correções nas classes da visão em função do nome das outras classes
- Criar classe principal com método main (aplicacao)

Prática 4

- Baixar a API FormsLayout
- Incluir como lib do projeto Java
- Copiar classes da visão – interface gráfica
- Criar enum TipoDocumento em controle/constantes
- Fazer correções nas classes da visão em função do nome das outras classes
- Criar classe principal com método main (aplicacao)

Prática 4

MaqBusca.java

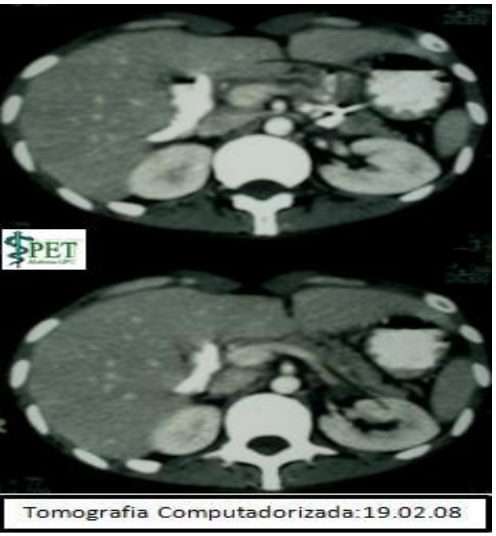
```
public static void main(String[] args) {
    ParserArquivosDados parserDados = new ParserArquivosXML();
    IndexadorLucene indexador = new IndexadorLuceneXML();
    File diretorioColecao;
    File[] ponteiroArquivosAmostras;
    boolean boolCriaNovoIndice = true;
    // lista arquivos do diretório
    diretorioColecao = new File(TipoDocumento.XMLNFe.getCaminhoAmostras());
    ponteiroArquivosAmostras = diretorioColecao.listFiles();

    // para cada página no diretório
    for (int i = 0; i < ponteiroArquivosAmostras.length; i++)
    {
        // extrai evidências textuais
        parserDados.extrairEvidencias(ponteiroArquivosAmostras[i]);
        // gera índice invertido
        indexador.indexarDocumento(TipoDocumento.XMLNFe.getCaminhoIndice(),
                parserDados.getEvidencia(), EvidenciasNFe.analisaStopWords,
                boolCriaNovoIndice);
        // depois de passar pelo primeiro documento a ser indexado, o índice é incremental
        boolCriaNovoIndice = false;
    }
    JanelaPrincipal.getInstance();
}
```

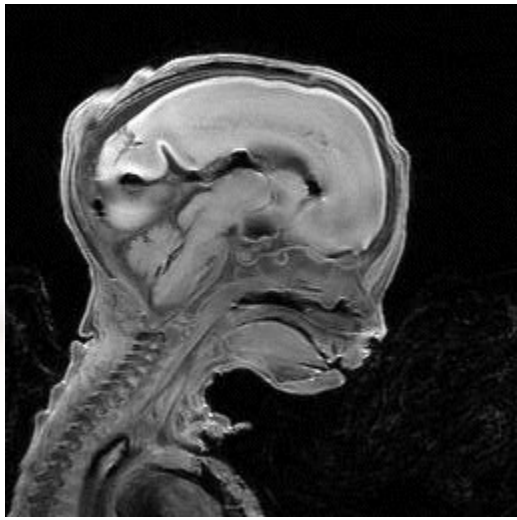
Considerações Finais

- Conhecimento do negócio:
  - O que posso empregar de conhecimento na implementação da minha máquina de busca para que o usuário fique mais satisfeito?
- Utilizando Java tenho várias APIs para lidar com os documentos, indexá-los e recuperá-los de maneira mais elegante e menos onerosa para o desenvolvedor.

Áreas de Aplicação

Medicina





Áreas de Aplicação

Internet

Áreas de Aplicação

Bibliotecas Virtuais

Melhorias

- Criar fábricas para parser, indexador e recuperador de dados
- Criar componente de busca genérico para alguns tipos de arquivo e gerar jar único
- Implementar extrator de características a partir de arquivos HTML ou PDF
- Implementar um método de recuperação mais eficiente

Melhorias

Changing Similarity

Chances are DefaultSimilarity is sufficient for all your searching needs. However, in some applications it may be necessary to customize your Similarity implementation. For instance, some applications do not need to distinguish between shorter and longer documents (see a "fair" similarity).

To change Similarity, one must do so for both indexing and searching, and the changes must happen before either of these actions take place. Although in theory there is nothing stopping you from changing mid-stream, it just isn't well-defined what is going to happen.

To make this change, implement your own Similarity (likely you'll want to simply subclass DefaultSimilarity) and then use the new class by calling IndexWriter.setSimilarity before indexing and Searcher.setSimilarity before searching.

If you are interested in use cases for changing your similarity, see the Lucene users's mailing list at Overriding Similarity. In summary, here are a few use cases:

1. SweetSpotSimilarity — SweetSpotSimilarity gives small increases as the frequency increases a small amount and then greater increases when you hit the "sweet spot", i.e. where you think the frequency of terms is more significant.

2. Overriding tf — In some applications, it doesn't matter what the score of a document is as long as a matching term occurs. In these cases people have overridden Similarity to return 1 from the tf() method.

3. Changing Length Normalization — By overriding lengthNorm, it is possible to discount how the length of a field contributes to a score. In DefaultSimilarity, lengthNorm = 1 / (numTerms in field)^0.5, but if one changes this to be 1 / (numTerms in field), all fields will be treated "fairly".

In general, Chris Hostetter sums it up best in saying (from the Lucene users's mailing list):

> [One would override the Similarity in] ... any situation where you know more about your data then just that it's "text" is a situation where it *might* make sense to to override your Similarity method.

Melhorias

Changing Scoring — Expert Level

Changing scoring is an expert level task, so tread carefully and be prepared to share your code if you want help.

With the warning out of the way, it is possible to change a lot more than just the Similarity when it comes to scoring in Lucene. Lucene's scoring is a complex mechanism that is grounded by three main classes:

1. Query — The abstract object representation of the user's information need.
2. Weight — The internal interface representation of the user's Query, so that Query objects may be reused.
3. Scorer — An abstract class containing common functionality for scoring. Provides both scoring and explanation capabilities.

Details on each of these classes, and their children, can be found in the subsections below.

Referências

- Site do Eclipse
  http://eclipse.org

- Site da API Lucene:
  http://lucene.apache.org/java/docs/

- Site da API JDom:
  http://www.jdom.org/

- Site da API HTMLParser:
  http://htmlparser.sourceforge.net/samples.html

Referências

- Site da API PDFBox:
  http://www.pdfbox.org/

- Outras soluções que convertem diferentes tipos de conteúdo em documentos da Lucene:
  ../Libs/lucene/lucene-2.9.3/lucene-2.9.3/docs/contributions.html#Lucene Document Converters

- Artigo: O Futuro das Máquinas de Busca
  Disponível em:
  http://googlediscovery.com/2007/11/16/o-futuro-das-maquinas-de-busca/

- XML Marker:
  http://symbolclick.com/