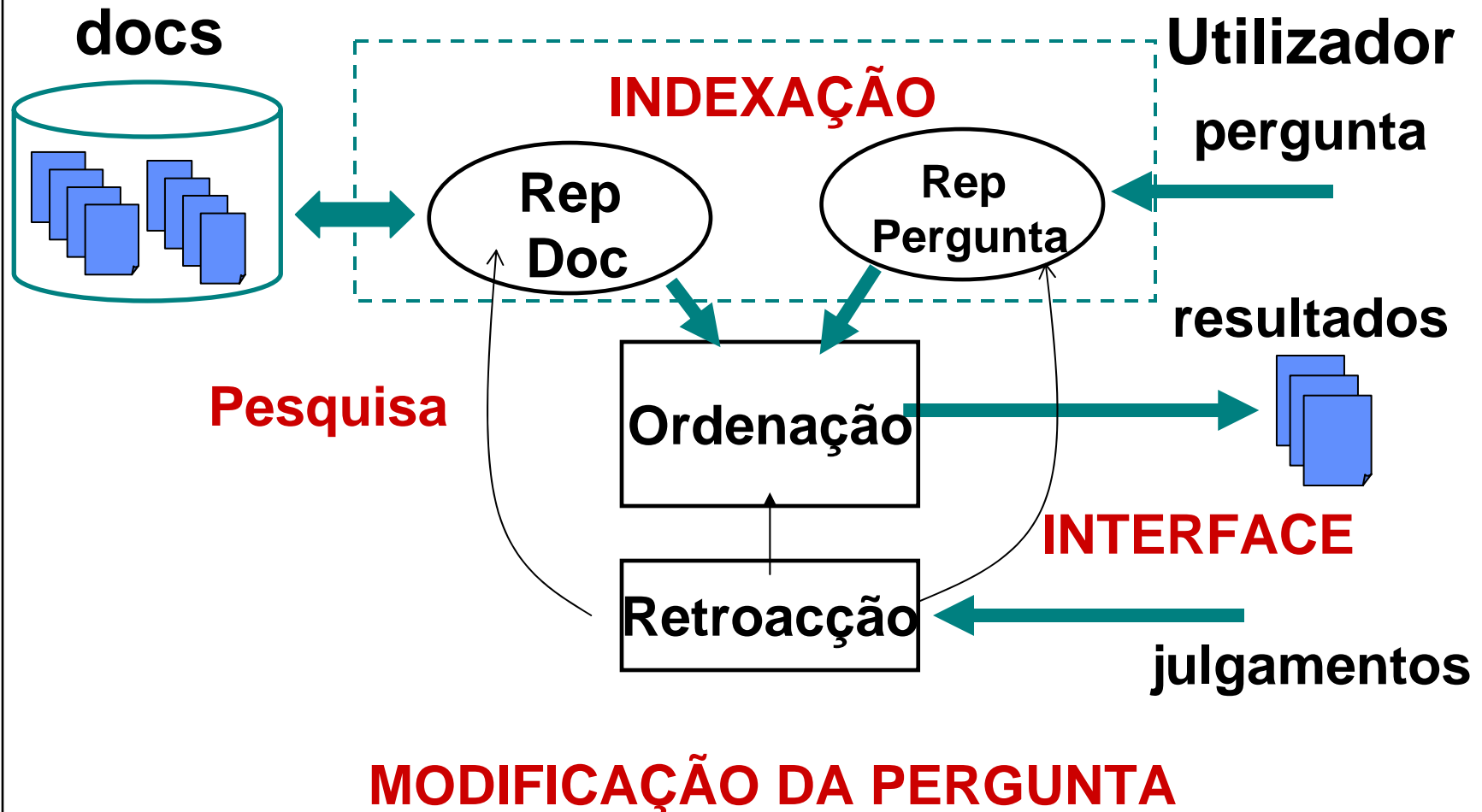


Recuperação de Informação – sistema “Lucene”

Recuperação de Informação – arquitectura geral



Sistemas de Recuperação de Informação “open source”

- Smart (Cornell)
 - modelo vectorial com várias opções para os pesos dos termos
 - escrito em C; bom desempenho na TREC
- MG (RMIT & Melbourne, Australia; Waikato, New Zealand)
 - modelo vectorial com ajustes para aumentar desempenho
 - escrito em C; bom desempenho na TREC; pesquisa texto e imagem
- Lucy (RMIT, Australia)
 - modelo booleano com ordem e interrogação com frases
 - escrito em C; bom desempenho na TREC; pesquisa documentos HTML
- Lemur (CMU/Univ. of Massachusetts)
 - sistema de teste de modelos de Recuperação de Informação
 - modelo vectorial, probabilístico, modelos linguísticos, ...
 - escrito em C++; bom desempenho na TREC

Sistema Lucene – aspectos gerais

Sistema de Recuperação de Informação de código fonte aberto:

- realiza indexação de colecções de texto, e
- responde a interrogações sobre colecções.

Lucene (“The Apache Software Foundation”)

[http://**lucene**.apache.org/](http://lucene.apache.org/)

Apenas trabalha sobre texto, mas existem extensões que permitem converter, em texto, diversos outros formatos, tais como: .doc, .ppt, .xls, .pdf, etc.

“Apache POI – Java API To Access Microsoft Format Files”

[http://**poi**.apache.org/](http://poi.apache.org/)

“Java libraries to read and write PDF files”

<http://schmidt.devlib.org/java/libraries-pdf.html/>

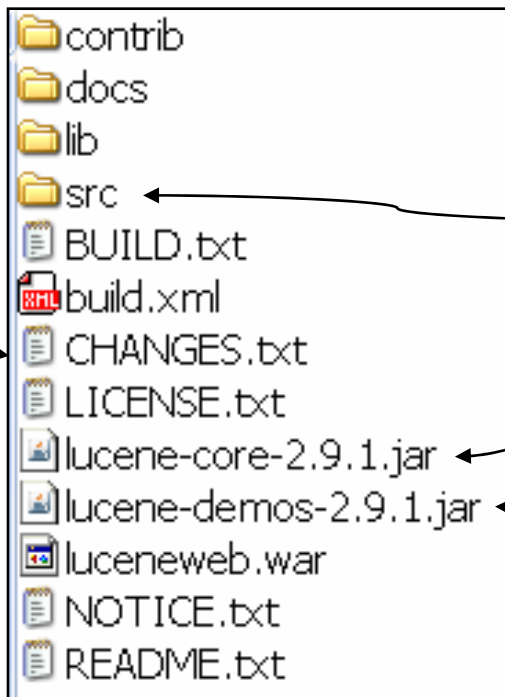
... instalar e usar



lucene-2.9.1.zip
26,357 KB

```
import org.apache.lucene.document.Document;  
import org.apache.lucene.document.Field;  
import org.apache.lucene.analysis.Analyzer;  
import org.apache.lucene.index.IndexWriter;  
...
```

MeuIndexador.java



o código fonte (Java)

a biblioteca a utilizar pelas aplicações

uma biblioteca com exemplos simples

O Lucene é API para Recuperação de Informação (RI)

O essencial na RI:

- indexar, e
- interrogar.

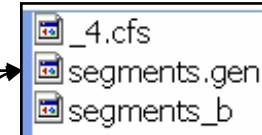
Core

org.apache.lucene	Top-level package.
org.apache.lucene.analysis	API and code to convert text into indexable/searchable tokens.
org.apache.lucene.analysis.standard	A grammar-based tokenizer constructed with JavaCC.
org.apache.lucene.document	The logical representation of a Document for indexing and searching.
org.apache.lucene.index	Code to maintain and access indices.
org.apache.lucene.queryParser	A simple query parser implemented with JavaCC.
org.apache.lucene.search	Code to search indices.
org.apache.lucene.search.function	Programmatic control over documents scores.
org.apache.lucene.search.payloads	The payloads package provides Query mechanisms for finding and using payloads.
org.apache.lucene.search.spans	The calculus of spans.
org.apache.lucene.store	Binary i/o API, used for all index data.
org.apache.lucene.util	Some utility classes.

Indexação – conceitos (no Lucene)

- Índice contém uma sequência de Documentos
 - o índice está representado em ficheiros
 - ... esses ficheiros têm os dados que indexam os documentos
- Documento (perspectiva do índice) é numa sequência de Campos
 - cada instância de `Document` contém uma lista de `Field`
- Campo tem um nome e uma sequência de Termos
 - cada instância de `Field` contém uma lista de `Term`
- ... um Termo é uma “string”
- 1 mesma “string” em 2 Campos são 2 Termos diferentes
- ... assim, cada Termo é representado por um par de “strings”
 - *<nomeDoCampo, textoDoTermo>*

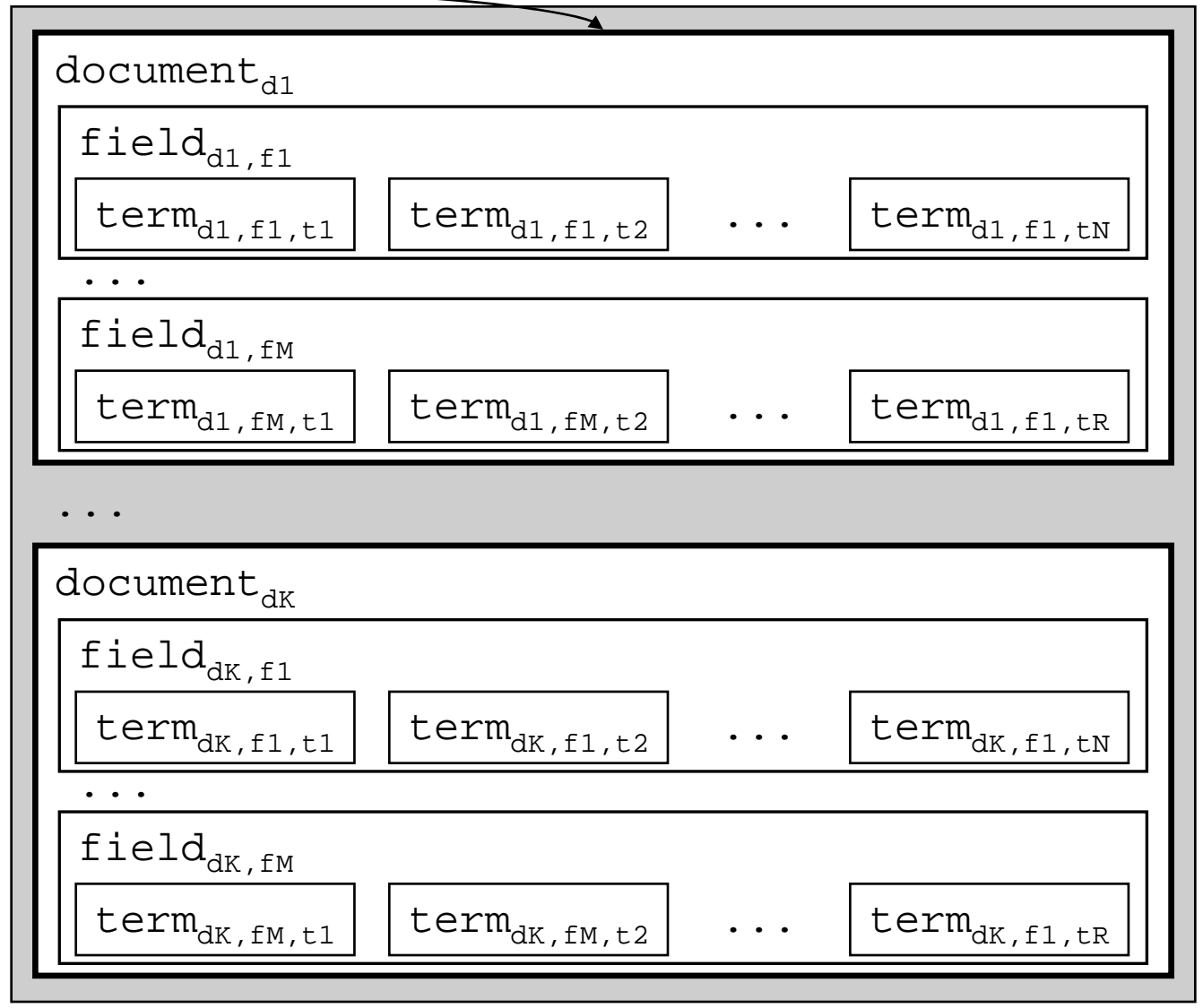
Exemplo de um índice
(que indexa vários documentos)



... índice, documento, campo e termo (no Lucene)

Documento
(na perspectiva
do índice)

Índice
(representado
em ficheiros)



... Campo (“Field”) – características

- No campo, cada termo pode ser
 - armazenado (ficará no `Document`)
 - não armazenado (posteriormente não se obtém em `Document`)
- No campo, cada termo pode ser
 - indexado, ou
 - não indexado
- ... cada termo indexado pode ser
 - transformado pelo processo de “análise” (“analyzed”), ou
 - usado literalmente (“not analyzed”)
- A maioria dos campos é “analyzed” mas,
 - é útil definir campos identificadores “not analyzed”
 - e.g., o caminho (“path”) o documento é um seu identificador
 - ... armazenar sem transformação (para não perder informação)

Exemplo – indexar e pesquisar informação em artigos

Considere-se um sistema para submeter artigos.

Cada artigo submetido será avaliado de modo cego, i.e. sem fornecer aos avaliadores qualquer informação sobre os autores.

Para isso, o processo de submissão solicita separadamente o título, a lista de autores, os temas e o conteúdo propriamente dito.

Depois, apenas o título e conteúdo são enviados aos avaliadores.

Pretende-se:

- indexar toda a informação sobre os artigos submetidos de modo a simplificar a pesquisa dos seus conteúdos.

... exemplo – criar cada documento

```
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.index.IndexWriter;
```

```
protected Document criarDocumento( String a_titulo, String a_lstAutores,
                                   String a_lstTemas, File a_ficheiro )
```

```
throws FileNotFoundException
```

```
{ Document documento = new Document();
```

```
→ documento.add(new Field("titulo", a_titulo, Field.Store.YES, Field.Index.ANALYZED));
```

```
→ documento.add(new Field("autor", a_lstAutores, Field.Store.YES, Field.Index.ANALYZED));
```

```
→ documento.add(new Field("tema", a_lstTemas, Field.Store.YES, Field.Index.ANALYZED));
```

```
    // Sobre o ficheiro
```

```
    ○String l_localizacao = a_ficheiro.getAbsolutePath();
```

```
→ documento.add(new Field("localizacao", l_localizacao,
                           Field.Store.YES, Field.Index.NOT_ANALYZED));
```

```
    ○String l_ultimaDataDeEscrita = DateTools.timeToString(a_ficheiro.lastModified(),
                                                           DateTools.Resolution.MINUTE);
```

```
→ documento.add(new Field("ultimaDataDeEscrita", l_ultimaDataDeEscrita,
                           Field.Store.YES, Field.Index.NOT_ANALYZED));
```

```
    // O conteúdo do documento
```

```
    ●FileReader l_leitorDoConteudo = new FileReader( a_ficheiro );
```

```
→ documento.add(new Field("conteudo", l_leitorDoConteudo));
```

```
    return documento; }
```

Criar documento (índice) separando

- título, autores, temas e
- conteúdo

... cada um no seu "Field"

"localização" é
identificador único,
"not analyzed"

"ultimaDataEscrita" é
informação adicional

"analyzed field"

... exemplo – indexar cada documento

... (continuação da folha anterior)

```
private void indexarDocumento( Document documento ) throws Exception
{
    Analyzer analisador = new StandardAnalyzer();
    //Analyzer analisador = new MeuAnalisador( "_asMinhasStopWords.txt" );

    String directorioComOsMeusIndices = "_osMeusIndices";
    File dir = new File( directorioComOsMeusIndices );
    boolean dirEmpty = ( dir.isDirectory() && ( dir.list().length == 0 ) );
    boolean dirExists = dir.exists();
    boolean dirReadable = dir.canRead();
    boolean criarNovo = dirEmpty || ( ! dirExists ) || ( ! dirReadable );

    IndexWriter escritorDoIndice = new IndexWriter( directorioComOsMeusIndices,
                                                    analisador,
                                                    criarNovo );

    escritorDoIndice.addDocument( documento );
    escritorDoIndice.optimize();
    escritorDoIndice.close();
}
```

definição
de “outro”
analisador;
depois
veremos
como se
constrói

construtor
do índice

... exemplo – indexar um “artigo”

Indexar

≡

criar estrutura do documento na perspectiva do índice

+

representar fisicamente essa estrutura e juntá-la às restantes

```
public void indexarArtigo( String a_titulo, String a_listaDeAutores,
                          String a_listaDeTemas, String a_nomeDoFicheiro )
{
    File ficheiro = new File( a_nomeDoFicheiro );
    if( ( ! ficheiro.exists() ) || ficheiro.isDirectory() )
    { throw new Error( "Erro MeuIndexador - leitura do ficheiro" ); }
    try
    {
        Document documento = criarDocumento( a_titulo, a_listaDeAutores,
                                              a_listaDeTemas, ficheiro );
        indexarDocumento( documento );
    }
    catch( FileNotFoundException e )
    { System.out.println( "Não existe: " + a_nomeDoFicheiro ); }
    catch( Exception e )
    { System.out.println( "Erro: indexarArtigo" ); }
}
```

... exemplo – indexar diversos “artigos”

Uma forma de invocar os métodos (atrás desenvolvidos)

... (continuação da folha anterior)

```
public static void main( String[] args )
{
    System.out.println( "O Meu Indexador!" );

    MeuIndexador meuIndexador = new MeuIndexador();

    meuIndexador.indexarArtigo( "O rato e o rei",
                                "Desconhecido & Popular",
                                "destrava línguas",
                                "oArtigo_A.txt" );

    meuIndexador.indexarArtigo( "A vida",
                                "Gato & Rato & Rei",
                                "crónica",
                                "oArtigo_B.txt" );

    meuIndexador.indexarArtigo( "Gato branco, gato preto",
                                "Kusturika",
                                "filme estrangeiro",
                                "oArtigo_C.txt" );
}
```

Uma colecção de “artigos”!

O rato roeu a rolha do
garrafão do rei da Rússia;
o rei ficou zangado!

oArtigo_A.tx

O gato e o rato vivem no
palácio do rei.

oArtigo_B.tx

"Gato branco, gato preto"
é um filme (de Kusturika)
com imagens surrealistas.

oArtigo_C.tx

Os índices sobre a colecção

_4.cfs
segments.gen
segments_b

... exemplo – obter os termos indexados

```
import org.apache.lucene.index.IndexReader;
import org.apache.lucene.index.Term;
import org.apache.lucene.index.TermEnum;

private IndexReader obterLeitorDoIndice( String a_directorioComOsIndices )
{
    IndexReader leitorDoIndice = null;
    try{ leitorDoIndice = IndexReader.open( a_directorioComOsIndices ); }
    catch( Exception e ) { System.out.println( "Erro MeuInfo (a)" ); }
    return leitorDoIndice; }
}
```

```
public void obterTermos( String a_directorioComOsIndices )
{
    IndexReader leitorDoIndice = obterLeitorDoIndice( a_directorioComOsIndices );
    try{
        TermEnum listaDeTermos = leitorDoIndice.terms();
        while( listaDeTermos.next() )
        {
            Term termo = listaDeTermos.term();
            String lstr_termo = termo.text();
            String lstr_campo = termo.field();
            int numeroDeDocumentosComTermo = listaDeTermos.docFreq();
            System.out.println( "< " + lstr_termo + ", campo=\"" + lstr_campo + "\"" +
                                ", df=" + numeroDeDocumentosComTermo + " >" ); } }
    catch( Exception e ){} }
}
```

... executar obterTermos

```
< desconhecido, campo="autor", df=1 >
< gato, campo="autor", df=1 >
< kusturika, campo="autor", df=1 >
< popular, campo="autor", df=1 >
< rato, campo="autor", df=1 >
< rei, campo="autor", df=1 >
< a, campo="conteudo", df=1 >
< branco, campo="conteudo", df=1 >
< com, campo="conteudo", df=1 >
< da, campo="conteudo", df=1 >
< de, campo="conteudo", df=1 >
< do, campo="conteudo", df=2 >
< e, campo="conteudo", df=1 >
< ficou, campo="conteudo", df=1 >
< filme, campo="conteudo", df=1 >
< garrafão, campo="conteudo", df=1 >
< gato, campo="conteudo", df=2 >
< imagens, campo="conteudo", df=1 >
< kusturika, campo="conteudo", df=1 >
< no, campo="conteudo", df=1 >
< o, campo="conteudo", df=2 >
< palácio, campo="conteudo", df=1 >
< preto, campo="conteudo", df=1 >
< rato, campo="conteudo", df=2 >
< rei, campo="conteudo", df=2 >
< roeu, campo="conteudo", df=1 >
```

```
< rolha, campo="conteudo", df=1 >
< rússia, campo="conteudo", df=1 >
< surrealistas, campo="conteudo", df=1 >
< um, campo="conteudo", df=1 >
< vivem, campo="conteudo", df=1 >
< zangado, campo="conteudo", df=1 >
< é, campo="conteudo", df=1 >
< D:\ptrigo\oArtigo_A.txt, campo="localizacao", df=1 >
< D:\ptrigo\oArtigo_B.txt, campo="localizacao", df=1 >
< D:\ptrigo\oArtigo_C.txt, campo="localizacao", df=1 >
< crônica, campo="tema", df=1 >
< destrava, campo="tema", df=1 >
< estrangeiro, campo="tema", df=1 >
< filme, campo="tema", df=1 >
< línguas, campo="tema", df=1 >
< a, campo="titulo", df=1 >
< branco, campo="titulo", df=1 >
< e, campo="titulo", df=1 >
< gato, campo="titulo", df=1 >
< o, campo="titulo", df=1 >
< preto, campo="titulo", df=1 >
< rato, campo="titulo", df=1 >
< rei, campo="titulo", df=1 >
< vida, campo="titulo", df=1 >
< 200711041817, campo="ultimaDataDeEscrita", df=2 >
< 200711041818, campo="ultimaDataDeEscrita", df=1 >
```


Como construir “outro” analisador?

```
public class MeuAnalisador extends Analyzer
{
    private Set<String> _conjunto_stopWords = null;

    public MeuAnalisador( String a_nomeFicheiro_stopWords )
    {
        super();
        File ficheiro_stopWords = new File( a_nomeFicheiro_stopWords );
        if( ficheiro_stopWords.exists() && ( ! ficheiro_stopWords.isDirectory() ) )
        {
            try
            {
                _conjunto_stopWords = WordlistLoader.getWordSet( ficheiro_stopWords );
                System.out.println( "stopWords carregadas com sucesso" );
            }
            catch( Exception e )
            {
                System.out.println( "erro a carregar ficheiro com stopWords" );
            }
        }
    }

    // Método abstracto em "Analyzer"
    public TokenStream tokenStream( String nomeDoCampo, Reader leitorIO )
    {
        TokenStream resultado = null;
        resultado = new StandardTokenizer( leitorIO );
        resultado = new StandardFilter( resultado );
        resultado = new LowerCaseFilter( resultado );
        resultado = new StopFilter( resultado, _conjunto_stopWords );
        return resultado;
    }
}
```

Um analisador
que aceita um
ficheiro com
“Stop Words”

o “meu
analisador”;
comparar com
o código do
“standard”

org.apache.lucene.analysis.standard.StandardAnalyzer

... uma lista de “Stop Words” e os termos gerados

a
com
como
da
de
do
e
é
no
nos
na
nas
o
os
um
uma

```
< desconhecido, campo="autor", df=1 >
< gato, campo="autor", df=1 >
< kusturika, campo="autor", df=1 >
< popular, campo="autor", df=1 >
< rato, campo="autor", df=1 >
< rei, campo="autor", df=1 >
< branco, campo="conteudo", df=1 >
< ficou, campo="conteudo", df=1 >
< filme, campo="conteudo", df=1 >
< garrafão, campo="conteudo", df=1 >
< gato, campo="conteudo", df=2 >
< imagens, campo="conteudo", df=1 >
< kusturika, campo="conteudo", df=1 >
< palácio, campo="conteudo", df=1 >
< preto, campo="conteudo", df=1 >
< rato, campo="conteudo", df=2 >
< rei, campo="conteudo", df=2 >
< roeu, campo="conteudo", df=1 >
< rolha, campo="conteudo", df=1 >
< rússia, campo="conteudo", df=1 >
< surrealistas, campo="conteudo", df=1 >
< vivem, campo="conteudo", df=1 >
< zangado, campo="conteudo", df=1 >
```

```
< D:\ptrigo\oArtigo_A.txt, campo="localizacao", df=1 >
< D:\ptrigo\oArtigo_B.txt, campo="localizacao", df=1 >
< D:\ptrigo\oArtigo_C.txt, campo="localizacao", df=1 >
< crónica, campo="tema", df=1 >
< destrava, campo="tema", df=1 >
< estrangeiro, campo="tema", df=1 >
< filme, campo="tema", df=1 >
< línguas, campo="tema", df=1 >
< branco, campo="titulo", df=1 >
< gato, campo="titulo", df=1 >
< preto, campo="titulo", df=1 >
< rato, campo="titulo", df=1 >
< rei, campo="titulo", df=1 >
< vida, campo="titulo", df=1 >
< 200711041817, campo="ultimaDataDeEscrita", df=2 >
< 200711041818, campo="ultimaDataDeEscrita", df=1 >
```

... exemplo – interrogar

```
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.Hits;

public void interrogar()
{
    IndexReader leitorDoIndice = null;
    try { leitorDoIndice = IndexReader.open( _kDirectorioComOsIndices ); }
    catch( Exception e ) { System.out.println( "Erro MeuInterrogador (a)" ); }

    IndexSearcher pesquisador = new IndexSearcher( leitorDoIndice );
    Analyzer analisador = new StandardAnalyzer(); //Analisador usado na Interrogação
    try
    {
        String lstr_campo = obterCampoParaInterrogacao(); //Interagir com Utilizador
        QueryParser analisadorInterrogacao = new QueryParser( lstr_campo, analisador );


        String lstr_interrogacao = obterInterrogacao(); //Interagir com Utilizador
        if( lstr_interrogacao.isEmpty() ) return;

        Query documentoInterrogacao = analisadorInterrogacao.parse( lstr_interrogacao );
        Hits resposta = pesquisador.search( documentoInterrogacao );
        apresentarResposta( resposta );
    }
    catch( Exception e ) { System.out.println( "Erro MeuInterrogador (b)" ); } }
```

```
public interface I_Config
{
    final public static String
        _kDirectorioComOsIndices = "_osMeusIndices";
}
```

... exemplo – apresentar o resultado da interrogação

```
private void apresentarResposta( Hits resposta ) throws Exception
{
    Iterator<Hit> iterador = resposta.iterator();
    int ordem = 0;
    while( iterador.hasNext() )
    {
        Document documento = iterador.next().getDocument();
        apresentarDetalhe( documento, ordem + 1 );
        ordem++;
    }
}
```



```
private void apresentarDetalhe( Document documento, int ordem )
{
    String lstr_localizacao = documento.get( "localizacao" );
    String lstr_titulo = documento.get( "titulo" );
    String lstr_autor = documento.get( "autor" );
    String lstr_tema = documento.get( "tema" );
    String lstr_ultimaDataDeEscrita = documento.get( "ultimaDataDeEscrita" );
    Date lobj_ultimaDataDeEscrita = null;
    try
    {
        lobj_ultimaDataDeEscrita = DateTools.stringToDate( lstr_ultimaDataDeEscrita );
    }
    catch( Exception e ) { System.out.println( "Erro apresentarDetalhe" ); }

    System.out.print("[ " + ordem + " ] ");
    System.out.println( lstr_titulo + "; " + lstr_autor + "; " + lstr_tema );
    System.out.println( lstr_localizacao + " [ " + lobj_ultimaDataDeEscrita + " ]" );
}
```

... exemplo de interrogação (interacção com o utilizador)

Interacção com o utilizador;

- neste caso indica querer pesquisar, no campo “conteudo” o termo “rei”

O Meu Interrogador!

Campo = { titulo, autor, tema, conteudo } [conteudo]

? > conteudo

%conteudo%

? > rei

%rei%

[1] "O rato e o rei"; "Desconhecido & Popular"; "destrava línguas"

D:\ptrigo\Lucene\myAppl\meuRI\oArtigo_A.txt [Sun Nov 04 18:17:00 GMT 2007]

[2] "A vida"; "Gato & Rato & Rei"; "crónica"

D:\ptrigo\Lucene\myAppl\meuRI\oArtigo_B.txt [Sun Nov 04 18:17:00 GMT 2007]

Número de documentos encontrados: 2

Solução encontrada pelo sistema

... outra interrogação (na forma “campo : termo”)

Interacção com o utilizador;

- neste caso indica querer pesquisar, no campo “tema” o termo “destrava”

O Meu Interrogador!

Campo = { titulo, autor, tema, conteudo } [conteudo]

? >

%conteudo%

Implementou-se que por omissão
se considera o campo “conteudo”

? > tema:destrava

%tema:destrava%

[1] "O rato e o rei"; "Desconhecido & Popular"; "destrava línguas"

D:\ptrigo\Lucene\myAppl\meuRI\oArtigo_A.txt [Sun Nov 04 18:17:00 GMT 2007]

Número de documentos encontrados: 1

Solução encontrada pelo sistema

Sintaxe básica da interrogação (Lucene)

[campo:]termo1 OPERADOR [campo:]termo2 ...

OPERADOR ≡ AND, OR, - (representa NOT)

... interrogação avançada (Lucene)

campo com termo que é sequência de palavras (e.g., um título)

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

termo com meta-carácter

`te?t*`

? \equiv 0 ou 1 carácter

* \equiv 0 ou + caracteres

aumentar explicitamente (boost) a relevância de um termo

`jakarta^4 apache`

`"jakarta apache"^4 "Apache Lucene"`

default: boost = 1

termos com proximidade superior a determinado valor

`home~0.8`

`term~p`, $0 \leq p \leq 1$

default: `p` = 0.5

"jakarta" e "apache" distando 10 palavras um do outro no documento

`"jakarta apache"~10`

termos contidos no domínio (ordenação lexicográfica)

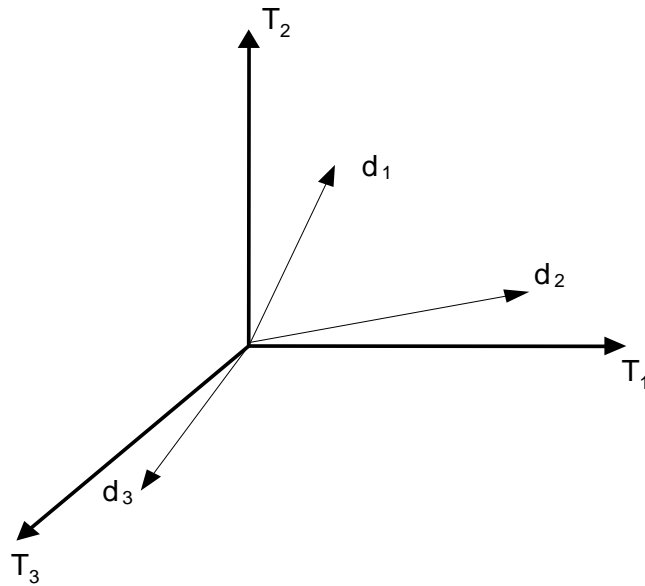
`nascido:[18880101 TO 19990101]`

`titulo:{Aida TO Carmen}`

cf. documentação Lucene em: "Apache Lucene - Query Parser Syntax"

Recordar conceitos do modelo vectorial

Modelo vectorial → ... no exemplo temos $n=3$ e $t=3$



	T_1	T_2	...	T_t
d_1	ω_{11}	ω_{21}	...	ω_{t1}
d_2	ω_{12}	ω_{22}	...	ω_{t2}
\vdots	\vdots	\vdots		\vdots
\vdots	\vdots	\vdots		\vdots
d_n	ω_{1n}	ω_{2n}	...	ω_{tn}

Recordar o peso “tf-idf”:

- $\omega_{t,d} = \text{tf}_{t,d} \times \log_2 (N / \text{df}_t)$, onde
- $\text{tf}_{t,d} \equiv$ número de ocorrências do termo t no documento d , e
- $\text{df}_t \equiv$ número de documentos que têm o termo t na colecção.

... como obter o valor de df_t e $tf_{t,d}$ (termo t e documento d)?

df_t \equiv número de documentos que têm o termo t na colecção

```
...  
TermEnum listaDeTermos = leitorDoIndice.terms();  
while( listaDeTermos.next() )  
{ Term termo = listaDeTermos.term();  
  int numeroDeDocumentosComTermo = listaDeTermos.docFreq();  
...  
}
```

extracto de código
já apresentado
anteriormente

$tf_{t,d}$ \equiv número de ocorrências do termo t no documento d

```
import org.apache.lucene.index.TermFreqVector;
```

veremos
em seguida

TermFreqVector

contém cada termo e respectiva frequência para cada campo de um documento

... dos conceitos à API

Formulação abstracta (como “tuplo”)

$\text{Collection} \equiv \langle \text{Document}_1, \dots, \text{Document}_n \rangle$

$\text{Document}_d \equiv \langle \text{Field}_{d,1}, \dots, \text{Field}_{d,m} \rangle$

$\text{Field}_{d,i} \equiv \langle \text{name}_i, \text{TermFreqVector}_{d,i} \rangle$

TermFreqVector $_{d,i} \equiv \langle \langle \text{term}_{1,i}, \text{termCount}_i \rangle, \dots, \langle \text{term}_{k,i}, \text{termCount}_i \rangle \rangle$

→ Nota: a frequência é relativa ao campo (o documento é uma lista de campos)

Formulação concreta (como “API” Java)

```
Interface TermFreqVector {  
    public String getField();  
    public String[] getTerms();  
    public int[] getTermFrequencies();  
    ...  
}
```

... obtido a partir
de um índice

```
Class IndexReader {  
    abstract TermFreqVector getTermFreqVector(int docNumber, String field);  
    ...  
}
```

Como criar índices que contenham os $tf_{t,d}$?

Definir um campo (`Field`) que armazene o vector de frequências (`TermFreqVector`)

```
new Field(  
    "nomeDoCampo",  
    "Este é o texto que será indexado!"  
    Field.Store.YES,  
    Field.Index.ANALYZED,  
    Field.TermVector.YES);
```

```
Field.TermVector.YES  
Field.TermVector.NO - default  
Field.TermVector.WITH_POSITIONS - Token Position  
Field.TermVector.WITH_OFFSETS - Character offsets  
Field.TermVector.WITH_POSITIONS_OFFSETS
```

outras
opções

... diferença entre “Store”, “Index” e “TermVector”

Ver mais detalhe em: `org.apache.lucene.document.Field`

- `Store.YES` \equiv as-is value stored in the Lucene index
 - in a non-inverted manner.
- `Index.ANALYZED` \equiv field analyzed with specified Analyzer
 - the tokens emitted are indexed in an inverted manner.
- `Index.NOT_ANALYZED` \equiv index field value without any Analyzer
 - the field's value is indexed in an inverted manner.
- `TermVector.YES` \equiv store the term vector of each document
- `TermVector.WITH_POSITION` \equiv YES + term position

... alterar `criarDocumento` para lidar com $tf_{t,d}$

```
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.index.IndexWriter;

protected Document criarDocumento( String a_titulo, String a_lstAutores,
                                   String a_lstTemas, File a_ficheiro )
throws FileNotFoundException
{ Document documento = new Document();
  documento.add(new Field("titulo", a_titulo, Field.Store.YES, Field.Index.ANALYZED));
  documento.add(new Field("autor", a_lstAutores, Field.Store.YES, Field.Index.ANALYZED));
  documento.add(new Field("tema", a_lstTemas, Field.Store.YES, Field.Index.ANALYZED));

  // Sobre o ficheiro
  String l_localizacao = a_ficheiro.getAbsolutePath();
  documento.add(new Field("localizacao", l_localizacao,
                          Field.Store.YES, Field.Index.NOT_ANALYZED));
  String l_ultimaDataDeEscrita = DateTools.timeToString(a_ficheiro.lastModified(),
                                                         DateTools.Resolution.MINUTE);
  documento.add(new Field("ultimaDataDeEscrita", l_ultimaDataDeEscrita,
                          Field.Store.YES, Field.Index.NOT_ANALYZED));

  // O conteúdo do documento
  FileReader l_leitorDoConteudo = new FileReader( a_ficheiro );
  documento.add(new Field("conteudo", l_leitorDoConteudo, Field.TermVector.WITH_POSITIONS));
  return documento; }
```

Única
alteração!

... obter os valores de $tf_{t,d}$

```
import org.apache.lucene.index.IndexReader;
import org.apache.lucene.index.TermFreqVector;

private IndexReader obterVectorDeFrequenciaDeCadaTermo( String a_directorioComOsIndices )
{
    IndexReader leitorDoIndice = obterLeitorDoIndice( a_directorioComOsIndices );
    int numeroDeDocumentos = leitorDoIndice.numDocs();
    for( int iLoop = 0; iLoop < numeroDeDocumentos; iLoop++ )
    {
        try { TermFreqVector vector_tf =
            leitorDoIndice.getTermFreqVector( iLoop, "conteudo" );
            Document documento = leitorDoIndice.document( iLoop );
            apresentarVectorTermoFrequencia( documento, vector_termoFrequencia );
        } catch( Exception e ) { System.out.println( "Erro" ); } } }
```

```
private void apresentarTermoFrequencia( Document documento, TermFreqVector vector_tf )
{
    String[] listaDeTermos = vector_tf.getTerms();
    int[] listaDeFrequencias = vector_tf.getTermFrequencies();
    int numeroDeTermos = listaDeTermos.length;

    for( int iLoop = 0; iLoop < numeroDeTermos; iLoop++ )
    {
        String termo = listaDeTermos[ iLoop ];
        int frequencia = listaDeFrequencias[ iLoop ];
        int[] listaDePosicoesDoTermo =
            ((TermPositionVector) vector_tf ).getTermPositions( iLoop );
        ...
    }
}
```

... uma lista de termos e frequências

Um formato para apresentar a informação

```
documento-d  
[ <termo- $t_d$ ,  $tf_{t,d}$ , |posição $_{1,d}$ , ..., posição $_{N,d}$ |>,  
  ...  
  <termo- $t_d$ ,  $tf_{t,d}$ , |posição $_{1,d}$ , ..., posição $_{N,d}$ |> ]  
...
```

a lista de posições do termo, t , no documento d

Recordar:

a colecção de “artigos”!
e as “Stop Words”

a
com
como
da
de
do
e
é
no
nos
na
nas
o
os
um
uma

...

"Gato branco, gato preto"
é um filme (de Kusturika)
com imagens surrealistas.

oArtigo_C.tx

```
D:\ptrigo\oArtigo_A.txt  
[ <ficou, 1, |7|>,  
  <garraão, 1, |3|>,  
  <rato, 1, |0|>,  
  <rei, 2, |4,6|>,  
  <roeu, 1, |1|>,  
  <rolha, 1, |2|>,  
  <rússia, 1, |5|>,  
  <zangado, 1, |8|> ]
```

```
D:\ptrigo\oArtigo_B.txt  
[ <gato, 1, |0|>,  
  <palácio, 1, |3|>,  
  <rato, 1, |1|>,  
  <rei, 1, |4|>,  
  <vivem, 1, |2|> ]
```

```
D:\ptrigo\oArtigo_C.txt  
[ <branco, 1, |1|>,  
  <filme, 1, |4|>,  
  <gato, 2, |0,2|>,  
  <imagens, 1, |6|>,  
  <kusturika, 1, |5|>,  
  <preto, 1, |3|>,  
  <surrealistas, 1, |7|> ]
```

Exemplo útil – obter os termos mais frequentes

```
import org.cnlp.apachecon.search.TermFreqComparator;
...
private static TermFreqComparator comparator = new TermFreqComparator();
...

protected Collection getTopTerms( TermFreqVector vector_tf, int numTopTerms )
{
    String[] terms = vector_tf.getTerms();
    int [] freqs = vector_tf.getTermFrequencies();
    List result = new ArrayList( terms.length );

    for (int i = 0; i < terms.length; i++)
    {
        result.add( new TermFreq( terms[i], freqs[i] ) );
    }
    Collections.sort( result, comparator );
    if ( numTopTerms < result.size() )
    {
        result = result.subList( 0, numTopTerms );
    }
    return result;
}
```


Outros aspectos – interrogação e posição do termo

```
import org.apache.lucene.search.spans;
```

Cada Span é um intervalo de posições de termos num documento.

Span \equiv <document, startPosition, endPosition>

```
Interface Spans {  
    public int doc(); //document number of current match  
    public int start(); //startPosition of current match  
    public boolean next();  
    public int end(); //endPosition of currentmatch  
    ...
```

```
Class SpanQuery {  
    abstract Spans getSpans(IndexReader reader);  
    ...
```

... obtido a partir de
interrogação "span"

... exemplo da interrogação “span”

Nesta frase alguns dos termos estão junto de outros!

first

near

allNear

```
SpanTermQuery frase = new SpanTermQuery(new Term("x", "frase"));
SpanTermQuery termos = new SpanTermQuery(new Term("x", "termos"));
SpanTermQuery junto = new SpanTermQuery(new Term("x", "junto"));

SpanFirstQuery first = new SpanFirstQuery(frase, 2);
Spans spans = first.getSpans(indexReader);
//fazer algo com os spans

SpanQuery [] clausulas = { termos, junto };
SpanNearQuery near = new SpanNearQuery(clausulas, 2, true);
spans = first.getSpans(indexReader);
//fazer algo com os spans

clausulas = new SpanQuery[]{ first, near };
SpanNearQuery allNear = new SpanNearQuery(clausulas, 3, false);
spans = allNear.getSpans(indexReader);
//do something with the spans
```

manter
relação de
ordem?

true

false

As interrogações Span*

- `SpanTermQuery` matches all spans containing a particular Term
- `SpanNearQuery` matches spans which occur near one another
 - within a maximum number of intervening unmatched positions,
 - can implement
 - ◊ phrase search (when built from `SpanTermQueries`), and
 - ◊ inter-phrase proximity (when built from other `SpanNearQueries`).
- `SpanOrQuery` merges spans from other `SpanQueries`
- `SpanNotQuery` removes spans matching one `SpanQuery` which
 - overlap another; can implement within-paragraph search
- `SpanFirstQuery` matches end position less than specified
 - can be used to constrain matches to the first part of the document

Como remover um documento de um índice?

- If you know the document number of a document (e.g. when iterating over `Hits`) that you want to delete you may use:
 - `IndexReader.deleteDocument(docNum)`
 - ... it not appear in `TermDocs` nor `TermPositions` enumerations.
- To delete all documents that contain a specific term you may use:
 - `IndexReader.deleteDocuments(term)`
 - ... because a variable number of documents can be affected by this method call it returns the number of deleted documents.
- Useful when a document field hold a unique ID string for document.
 - to delete document, just construct a term with the appropriate field, and
 - the unique ID string as its text and passes it to this method.
- ... Lucene 1.9 – class `IndexModifier` allows deleting documents.

A implementação (Lucene) do modelo vectorial

Ver mais detalhe em: `org.apache.lucene.search.Similarity`

$$\text{score}(q,d) = \text{coord}(q,d) \cdot \text{queryNorm}(q) \cdot \sum_{t \in q} (\text{tf}(t \text{ in } d) \cdot \text{idf}(t)^2 \cdot \text{t.getBoost()} \cdot \text{norm}(t,d))$$

“boost”
de termos

$$\text{tf}(t \text{ in } d) = \text{frequency}^{1/2}$$

$$\text{idf}(t) = 1 + \log \left(\frac{\text{numDocs}}{\text{docFreq}+1} \right)$$

sobre o
documento (d)

$$\text{norm}(t,d) = \text{doc.getBoost}() \cdot \text{lengthNorm}(\text{field}) \cdot \prod_{\text{field } f \text{ in } d \text{ named as } t} f.\text{getBoost}()$$

$\text{coord}(q,d) \equiv$ score factor based on how many of the query, q, terms are found in the specified document d.

sobre a
interrogação (q)

$$\text{queryNorm}(q) = \text{queryNorm}(\text{sumOfSquaredWeights}) = \frac{1}{\text{sumOfSquaredWeights}^{1/2}}$$

$$\text{sumOfSquaredWeights} = \sum_{t \in q} (\text{q.getBoost}()^2 \cdot \text{idf}(t) \cdot \text{t.getBoost}()^2)$$

“boost”
de termos