

ΜΥΕ003: Ανάκτηση Πληροφορίας

Διδάσκουσα: Ευαγγελία Πιτουρά
Περιγραφή Εργασίας

Περιεχόμενα Παρουσίασης

Σύντομη παρουσίαση

- Lucene
- Dataset/Collection
- Εργασία

Εργασία

Θέμα: Σχεδιασμός και υλοποίηση ενός συστήματος αναζήτησης άρθρων της wikipedia.

Βήμα 1: Συλλογή Wikipedia άρθρων. Αυτά τα άρθρα θα αποτελούν τη συλλογή σας (corpus).

Βήμα 2: Υλοποίηση μιας μηχανή αναζήτησης αυτών των άρθρων.

Συγκεκριμένα:

- Ο χρήστης θα θέτει ερωτήματα.
- Το σύστημα θα επιστρέφει τα συναφή με το ερώτημα άρθρα της συλλογής σας σε διάταξη με βάση τη συνάφεια τους με το ερώτημα.

Για την υλοποίηση, θα χρησιμοποιήσετε τη βιβλιοθήκη **Lucene**

Διαδικαστικά

Καταληκτικές Ημερομηνίες

- Τετάρτη 28 Απριλίου 2020, 5μμ Σύντομη περιγραφή σχεδιασμού και συλλογή δεδομένων
- Παρασκευή 29 Μαΐου 2020, 5μμ Παράδοση εργασίας
- Όταν ανοίξει το Πανεπιστήμιο, Εξέταση εργασίας

Οι καταληκτικές ημερομηνίες είναι αυστηρές, **δεν θα γίνουν δεκτές αργοπορημένες παραδόσεις ασκήσεων**

- Η εργασία μπορεί να γίνει σε ομάδες έως 2 ατόμων.
- Η εργασία μετράει σε ποσοστό 50% στο βαθμό σας στο μάθημα.

ΜΥΕ003: Ανάκτηση Πληροφορίας

Διδάσκουσα: Ευαγγελία Πιτουρά

Lucene

Εισαγωγή

- **Open source** search software
- **Lucene Core** provides **Java-based** indexing and search as well as spellchecking, hit highlighting and advanced analysis/tokenization capabilities
- Let you add search to your application, not a complete search system by itself -- **software library** not an application
- Written by Doug Cutting



Εισαγωγή

- Used by LinkedIn, Twitter, Netflix, Oracle, ...
 - and many more (see <http://wiki.apache.org/lucene-java/PoweredBy>)
- Ports/integrations to other languages
 - C/C++, C#, Ruby, Perl, PHP
 - **PyLucene**: a Python port of the Core project

Μπορείτε να την κατεβάσετε από

<http://lucene.apache.org/core/>

Some features (indexing)

Scalable, high-performance indexing

- over 150GB/hour on modern hardware
- small RAM requirements -- only 1MB heap
- incremental indexing as fast as batch indexing
- index size roughly 20-30% the size of text indexed

Some features (search)

Powerful, accurate and efficient search algorithms

- **ranked** searching -- best results returned first
- many powerful **query types**: phrase queries, wildcard queries, proximity queries, range queries and more
- **fielded** searching (e.g. title, author, contents)
- **sorting** by any field
- allows **simultaneous update and searching**
- flexible **faceting, highlighting, joins** and **result grouping**
- fast, memory-efficient and typo-tolerant **suggesters**
- **pluggable ranking models**, including the Vector Space Model and Okapi BM25

Στόχος της παρουσίασης:

Σύντομη εισαγωγή

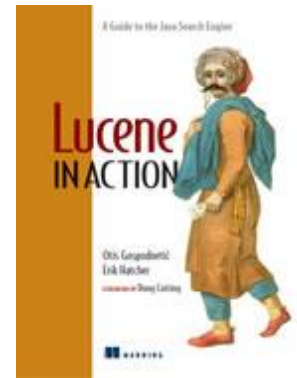
Περισσότερες πληροφορίες (προσοχή κάποια στοιχεία αναφέρονται σε παλιότερη έκδοση)

- <http://www.lucene-tutorial.com/>

<https://www.manning.com/books/lucene-in-action-second-edition>

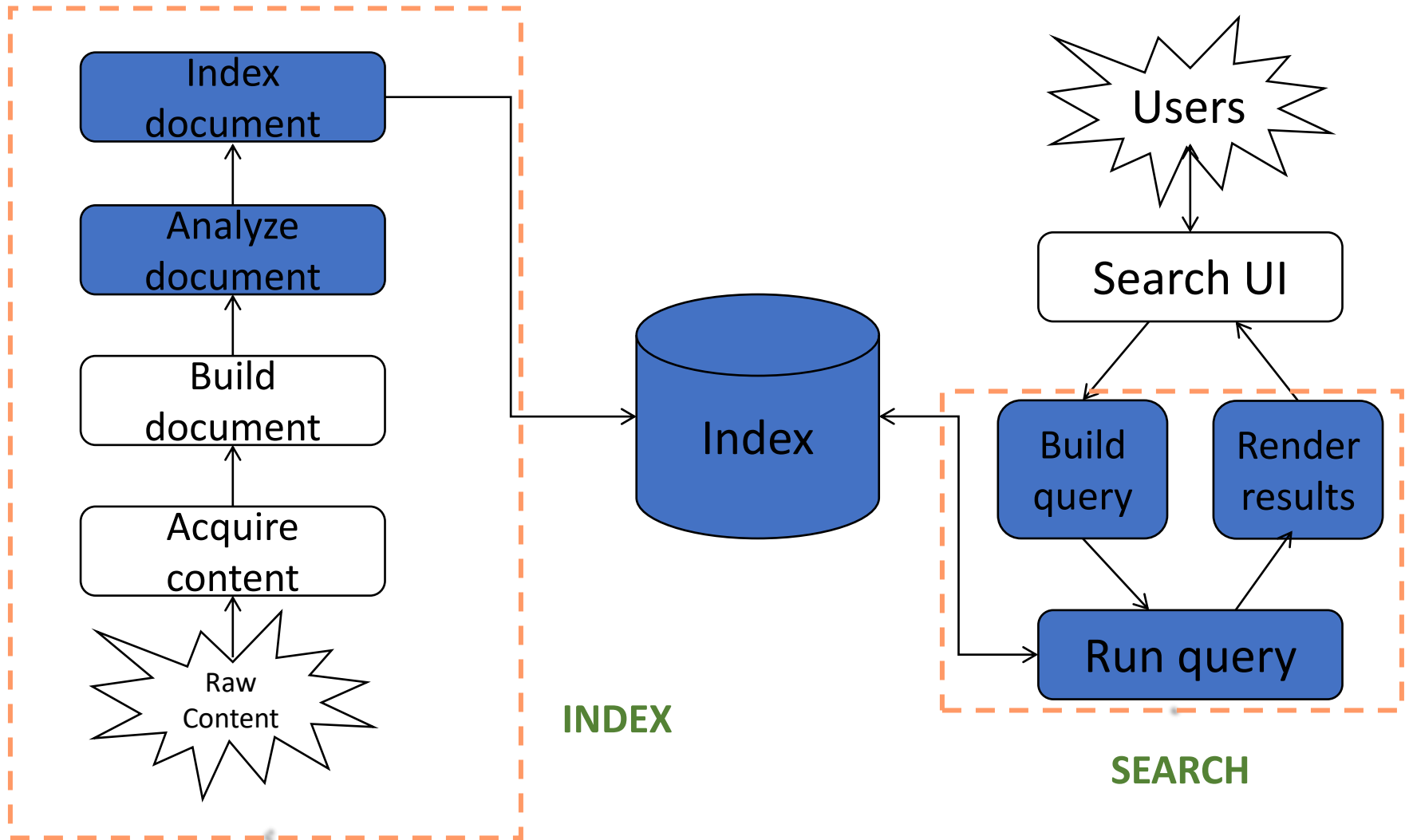
- **Lucene 8.5.0** demo API (recommended for more up-to-date code examples)
- offers simple example code to show the features of Lucene
 - http://lucene.apache.org/core/8_5_0/core/overview-summary.html#overview_description
 - http://lucene.apache.org/core/8_5_0/demo/overview-summary.html#overview_description

Μπορείτε να χρησιμοποιήσετε παλαιότερη version αν θέλετε



April 2020 update: Lucene 8.5.0

Βασικές έννοιες



Βασικές έννοιες: document

- The **unit** of search and index.
- **Indexing** involves adding Documents to an **IndexWriter**.
- **Searching** involves retrieving Documents from an index via an **IndexSearcher**.
- A document consists of one or more **Fields**
 - A Field is a name-value pair.
example: title, body or metadata (creation time, etc)

Βασικές έννοιες: Fields

- You have to translate raw content into Fields
- Search a field using <field-name:term>,
 - e.g., title:lucene

Βασικές έννοιες: index

- Indexing in Lucene
 1. Create documents comprising of one or more Fields
 2. Add these Documents to an IndexWriter.

Βασικές έννοιες: search

Searching requires an index to have already been built.

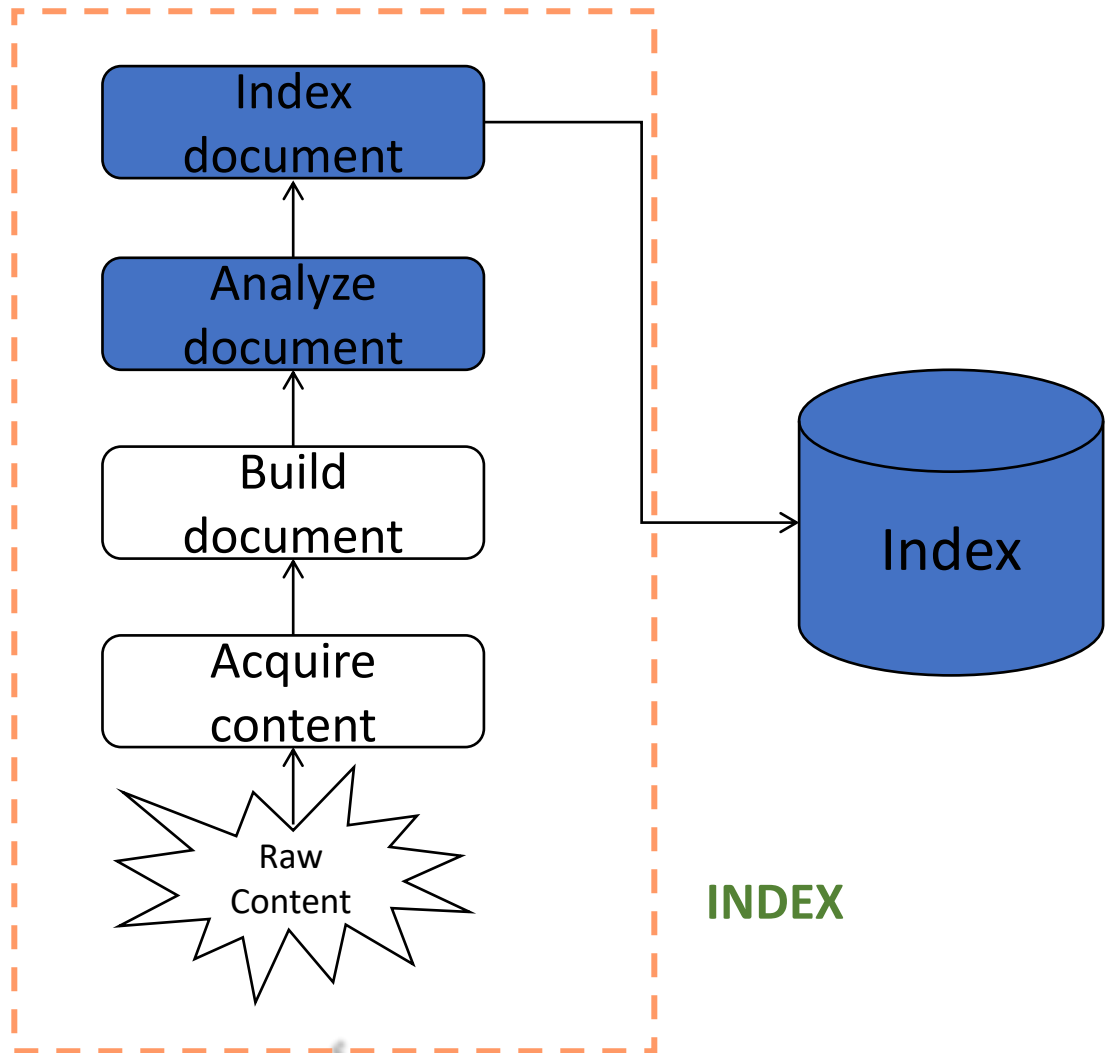
- It involves
 1. Create a Query (usually via a `QueryParser`) and
 2. Handle this Query to an `IndexSearcher`, which returns a list of `Hits`.
- The `Lucene query language` allows the user to specify
 - which field(s) to search on,
 - which fields to give more weight to (boosting),
 - the ability to perform boolean queries (AND, OR, NOT) and
 - other functionality.

Lucene in a search system: index

Lucene in a search system: **index**

Steps

1. Acquire content
2. Build document
3. **Analyze** document
4. **Index** documents



Step 1: Acquire and build content



Not supported by core Lucid

Collection depending on type may require:

- Crawler or spiders (web)
- Specific APIs provided by the application (e.g., Twitter, FourSquare, imdb)
- Scrapping
- Complex software if scattered at various location, etc

Complex documents (e.g., XML, JSON, relational databases, pptx etc)

Solr high performance search server built using Lucene Core, with XML/HTTP and JSON/Python/Ruby APIs, hit highlighting, faceted search, caching, replication, and a web admin interface.

<https://lucene.apache.org/solr/>

Competitor: **Elasticsearch**

Tika the Apache Tika™ toolkit detects and extracts metadata and text from over a thousand different file types (such as PPT, XLS, and PDF)

For example latest release automating image captioning

<http://tika.apache.org/>

Step 2: Build Documents

Create documents by adding fields

Fields may be

- indexed or not
 - Indexed fields may or may not be analyzed (i.e., tokenized with an `Analyzer`)
 - *Non-analyzed fields view the entire value as a single token* (useful for URLs, paths, dates, social security numbers, ...)
- stored or not
 - Useful for fields that you'd like to display to users
- Optionally store term vectors and other options such as positional indexes

Step 2: Build Documents

Create documents by adding fields

Step 1 – Create a method to get a Lucene document from a text file.

Step 2 – **Create various fields** which are key value pairs containing keys as names and values as contents to be indexed.

Step 3 – Set field to be **analyzed or not, stored or not**

Step 4 – Add the newly-created fields to the document object and return it to the caller method.

Step 2: Build Documents

```
private Document getDocument(File file) throws IOException {
    Document document = new Document();

    //index file contents
    Field contentField = new Field(LuceneConstants.CONTENTES,
    new FileReader(file))
    //index file name
    Field fileNameField = new Field(LuceneConstants.FILE_NAME, file.getName(), Field.Store.YES,Field.Index.NOT_ANALYZED);

    //index file path
    Field filePathField = new Field(LuceneConstants.FILE_PATH, file.getCanonicalPath(), Field.Store.YES,Field.Index.NOT_ANALYZED);

    document.add(contentField);
    document.add(fileNameField);
    document.add(filePathField);

    return document;
}
```

Step 3:analyze and index

Create an IndexWriter and add documents to it with addDocument();

Core indexing classes

- Analyzer

- Extracts tokens from a text stream

- IndexWriter

- create a new index, open an existing index, and
- add, remove, or update documents in an index

- Directory

- Abstract class that represents the location of an index


```
Analyzer analyzer = new StandardAnalyzer();
```

```
// INDEX: Store the index in memory: (για την εργασία θα το αποθηκεύστε στο δίσκο – θα δημιουργηθεί μια φορά στην αρχή)
```

```
Directory directory = new RAMDirectory();
```

```
// To store an index on disk, use this instead:
```

```
// Directory directory = FSDirectory.open("/tmp/testindex");
```

```
IndexWriterConfig config = new IndexWriterConfig(analyzer);
```

```
IndexWriter iwriter = new IndexWriter(directory, config);
```

```
Document doc = new Document();
```

```
String text = "This is the text to be indexed.";
```

```
doc.add(new Field("fieldname", text, TextField.TYPE_STORED));
```

```
iwriter.addDocument(doc);
```

```
iwriter.close();
```

```
// SEARCH: Now search the index:
```

```
DirectoryReader ireader = DirectoryReader.open(directory);
```

```
IndexSearcher isearcher = new IndexSearcher(ireader);
```

```
// Parse a simple query that searches for "text":
```

```
QueryParser parser = new QueryParser("fieldname", analyzer);
```

```
Query query = parser.parse("text");
```

```
ScoreDoc[] hits = isearcher.search(query, null, 1000).scoreDocs;
```

```
// Iterate through the results:
```

```
for (int i = 0; i < hits.length; i++) {
```

```
    Document hitDoc = isearcher.doc(hits[i].doc);
```

```
}
```

```
ireader.close();
```

```
directory.close();
```

Using Field options

Index	Store	TermVector	Example usage
NOT_ANALYZED	YES	NO	Identifiers, telephone/SSNs, URLs, dates, ...
ANALYZED	YES	WITH_POSITIONS_OFFSETS	Title, abstract
ANALYZED	NO	WITH_POSITIONS_OFFSETS	Body
NO	YES	NO	Document type, DB keys (if not used for searching)
NOT_ANALYZED	NO	NO	Hidden keywords

Analyzers

Tokenizes the input text

- Common Analyzers
 - WhitespaceAnalyzer
Splits tokens on whitespace
 - SimpleAnalyzer
Splits tokens on non-letters, and then lowercases
 - StopAnalyzer
Same as SimpleAnalyzer, but also removes stop words
 - StandardAnalyzer
Most sophisticated analyzer that knows about certain token types, lowercases, removes stop words, ...

Analysis examples

“The quick brown fox jumped over the lazy dog”

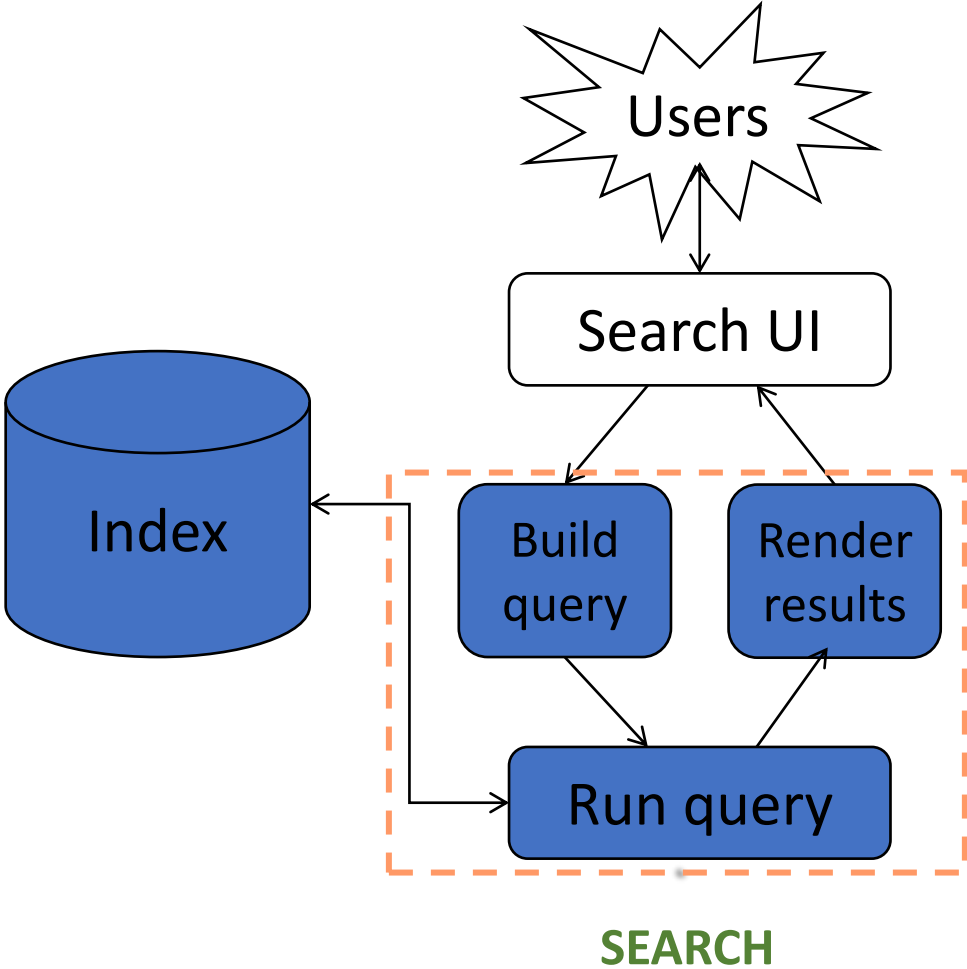
- `WhitespaceAnalyzer`
 - `[The] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dog]`
- `SimpleAnalyzer`
 - `[the] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dog]`
- `StopAnalyzer`
 - `[quick] [brown] [fox] [jumped] [over] [lazy] [dog]`
- `StandardAnalyzer`
 - `[quick] [brown] [fox] [jumped] [over] [lazy] [dog]`

More analysis examples

- “XY&Z Corporation – xyz@example.com”
- WhitespaceAnalyzer
 - [XY&Z] [Corporation] [-] [xyz@example.com]
- SimpleAnalyzer
 - [xy] [z] [corporation] [xyz] [example] [com]
- StopAnalyzer
 - [xy] [z] [corporation] [xyz] [example] [com]
- StandardAnalyzer
 - [xy&z] [corporation] [xyz@example.com]

Lucene in a search system: **search**

Lucene in a search system: search



Search User Interface (UI)

No default search UI, but many useful modules

General instructions

- Simple (do not present a lot of options in the first page)
 - `search box` better than 2-step process
- Result presentation is very important
 - highlight matches
 - make sort order clear, etc

Core searching classes

■ **QueryParser**

- Parses a textual representation of a query into a Query instance
- Constructed with an analyzer used to interpret query text in the same way as the documents are interpreted

■ **Query**

- Contains the results from the QueryParser which is passed to the searcher
- Abstract query class
- Concrete subclasses represent specific types of queries, e.g., matching terms in fields, boolean queries, phrase queries, ...

■ **IndexSearcher**

- Central class that exposes several search methods on an index
- Returns **TopDocs** with max n hits

```
Analyzer analyzer = new StandardAnalyzer();
```

//INDEX: Store the index in memory: (για την εργασία θα το αποθηκεύστε στο δίσκο – θα δημιουργηθεί μια φορά στην αρχή)

```
Directory directory = new RAMDirectory();
```

// To store an index on disk, use this instead:

```
// Directory directory = FSDirectory.open("/tmp/testindex");
```

```
IndexWriterConfig config = new IndexWriterConfig(analyzer);
```

```
IndexWriter iwriter = new IndexWriter(directory, config);
```

```
Document doc = new Document();
```

```
String text = "This is the text to be indexed.";
```

```
doc.add(new Field("fieldname", text, TextField.TYPE_STORED));
```

```
iwriter.addDocument(doc);
```

```
iwriter.close();
```

// QUERY: Now search the index:

```
DirectoryReader ireader = DirectoryReader.open(directory);
```

```
IndexSearcher isearcher = new IndexSearcher(ireader);
```

// Parse a simple query that searches for "text":

```
QueryParser parser = new QueryParser("fieldname", analyzer);
```

```
Query query = parser.parse("text");
```

```
ScoreDoc[] hits = isearcher.search(query, null, 1000).scoreDocs;
```

// Iterate through the results:

```
for (int i = 0; i < hits.length; i++) {
```

```
    Document hitDoc = isearcher.doc(hits[i].doc);
```

```
}
```

```
ireader.close();
```

```
directory.close();
```

QueryParser syntax examples

Query expression	Document matches if...
java	Contains the term <i>java</i> in the default field
java junit java OR junit	Contains the term <i>java</i> or <i>junit</i> or both in the default field (<i>the default operator can be changed to AND</i>)
+java +junit java AND junit	Contains both <i>java</i> and <i>junit</i> in the default field
title:ant	Contains the term <i>ant</i> in the title field
title:extreme - subject:sports	Contains <i>extreme</i> in the title and not <i>sports</i> in subject
(agile OR extreme) AND java	Boolean expression matches
title:"junit in action"	Phrase matches in title
title:"junit action"~5	Proximity matches (within 5) in title
java*	Wildcard matches
java~	Fuzzy matches
lastmodified:[1/1/09 TO 12/31/09]	Range matches

Scoring

- Scoring function uses basic *tf-idf* scoring with
 - Programmable boost values for certain fields in documents
 - Length normalization
 - Boosts for documents containing more of the query terms
- IndexSearcher provides a method that explains the scoring of a document

Summary

To use Lucene

1. Create [Documents](#) by adding [Fields](#);
2. Create an [IndexWriter](#) and add documents to it with [addDocument\(\)](#);
3. Call [QueryParser.parse\(\)](#) to build a query from a string; and
4. Create an [IndexSearcher](#) and pass the query to its [search\(\)](#) method.

Summary: Lucene API packages

- *org.apache.lucene.analysis* defines *an abstract Analyzer API* for converting text from a Reader into a TokenStream, an enumeration of token Attributes.
- *org.apache.lucene.document* provides a simple Document class. A **Document** is simply a set of named **Fields**, whose values may be strings or instances of Reader.
- *org.apache.lucene.index* provides two primary classes: **IndexWriter**, which creates and adds documents to indices; and **IndexReader**, which accesses the data in the index.
- *org.apache.lucene.store* defines an abstract class for storing persistent data, the **Directory**, which is a collection of named files written by an **IndexOutput** and read by an **IndexInput**. Multiple implementations are provided, including **FSDirectory**, which uses a file system directory to store files, and **RAMDirectory** which implements files as memory-resident data structures.

Summary: Lucene API packages

- *org.apache.lucene.search* provides
 - data structures to represent queries (ie **TermQuery** for individual words, **PhraseQuery** for phrases, and **BooleanQuery** for boolean combinations of queries) and
 - the **IndexSearcher** which turns queries into **TopDocs**.
 - A number of **QueryParsers** are provided for producing query structures from strings or xml.
- *org.apache.lucene.codecs* provides an abstraction over the encoding and decoding of the inverted index structure, as well as different implementations that can be chosen depending upon application needs.
- *org.apache.lucene.util* contains a few handy data structures and util classes, ie **FixedBitSet** and **PriorityQueue**.

ΜΥΕ003: Ανάκτηση Πληροφορίας

Διδάσκουσα: Ευαγγελία Πιτουρά

Λίγα περισσότερα για την εργασία

Wikipedia data

Έχετε δύο επιλογές

- Είτε να κατεβάσετε μόνοι σας τα άρθρα
- Είτε να χρησιμοποιείτε έτοιμες συλλογές

Wikipedia data

Εδώ υπάρχουν κάποιες οδηγίες για να πάρετε άρθρα

<https://stackabuse.com/implementing-word2vec-with-gensim-library-in-python/>

Scraping με χρήση BeautifulSoup

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

Wikipedia data

Αυτό είναι το interface της Wikipedia (υπάρχει όλο το dump, αλλά πολύ μεγάλο)

<https://en.wikipedia.org/wiki/Special:Export>

Wikipedia data

Συλλογές

Για παράδειγμα από το Kaggle (η λίστα είναι μόνο ενδεικτική)

<https://www.kaggle.com/jrobischon/wikipedia-movie-plots>

<https://www.kaggle.com/kenshoresearch/kensho-derived-wikimedia-data>

<https://www.kaggle.com/jacksoncrow/extended-wikipedia-multimodal-dataset>

Εργασία

Ανάλυση κειμένου και κατασκευή ευρετηρίου. Η Lucene παρέχει τη δυνατότητα για stemming, απαλοιφή stop words, επέκταση συνωνύμων, κλπ.

Επίσης, κάποιες λειτουργίες, όπως η διόρθωση τυπογραφικών λαθών, ή η επέκταση ακρωνύμων, μπορούν να γίνουν εναλλακτικά κατά τη διάρκεια της αναζήτησης (τροποποιώντας το ερώτημα).

Επιλέξτε το είδος της ανάλυσης που θεωρείτε κατάλληλο και εξηγήστε την επιλογή σας.

Εργασία

Αναζήτηση. Η *ελάχιστη απαίτηση* από το σύστημα σας είναι να υποστηρίζει αναζήτηση άρθρων με λέξεις κλειδιά.

Επιπρόσθετα, μια *άριστη εργασία*, θα πρέπει

(1) Να υποστηρίζει και άλλα είδη ερωτήσεων, για παράδειγμα αναζήτηση πεδίου, δηλαδή, την εμφάνιση όρων σε συγκεκριμένα πεδία (πχ. στον τίτλο).

(2) Να διατηρεί πληροφορία για την ιστορία των αναζητήσεων. Χρησιμοποιείστε αυτήν την πληροφορία για να προτείνετε εναλλακτικά ερωτήματα.

(3) Να χρησιμοποιεί embedding για να βελτιώσει την αναζήτηση.

Εργασία

Παρουσίαση Αποτελεσμάτων. Η *ελάχιστη απαίτηση* από το σύστημα σας είναι να παρουσιάζει τα αποτελέσματα σε διάταξη με βάση τη συνάφεια τους με το ερώτημα.

Επιπρόσθετα, μια *άριστη εργασία*, θα πρέπει

- (1) Να παρουσιάζει τα αποτελέσματα ανά 10, με δυνατότητα στο χρήστη να προχωρήσει στα επόμενα.
- (2) Οι λέξεις κλειδιά να παρουσιάζονται τονισμένες στο αποτέλεσμα.
- (3) Δυνατότητα διαφορετικής διάταξης των αποτελεσμάτων

Εργασία: φάση 1

Τρόπος παράδοσης: Η εργασία θα υποβληθεί στο ecourse του μαθήματος. Μια υποβολή ανά ομάδα.

Στόχος αυτής της φάσης είναι να σας βοηθήσει να κατανοήσετε τα βήματα που πρέπει να ακολουθήσετε.

Περιγράψτε συνοπτικά το σχεδιασμό του συστήματός σας. Ο σχεδιασμός μπορεί να αλλάξει κατά την υλοποίηση.

Η περιγραφή θα πρέπει να είναι από 600-1000 λέξεις.

Εργασία: φάση 1

Συμπεριλάβετε πληροφορίες για τα παρακάτω:

Συλλογή εγγράφων (corpus). Τον τρόπο που θα συλλέξετε τα άρθρα. Θα πρέπει να έχετε ήδη συλλέξει τουλάχιστον 10 άρθρα. Πείτε ποια.

Ανάλυση κειμένου και κατασκευή ευρετηρίου. την προεπεξεργασία των άρθρων για τη δημιουργία του εγγράφου, ποια θα είναι η μονάδα εγγράφου και τα αντίστοιχα πεδία (fields), τα ευρετήρια που σκοπεύετε να δημιουργήσετε (σε ποια πεδία, και τι είδους), κλπ, ώστε να υποστηρίζονται οι διάφοροι τρόποι αναζήτησης

Αναζήτηση. Αν σκοπεύετε να υλοποιήσετε κάτι πέρα της ελάχιστης απαίτησης, εξηγήστε τι και πως.

Παρουσίαση Αποτελεσμάτων. Αν σκοπεύετε να υλοποιήσετε κάτι πέρα της ελάχιστης απαίτησης, εξηγήστε τι και πως.

Στις απαντήσεις στα παραπάνω όπου είναι δυνατόν αναφερθείτε και στα αντίστοιχα τμήματα της Lucene, όπως Build/Analyze/Index Document IndexSearcher/QueryParser, TopDocs, ScoreDocs

Ερωτήσεις;