

ΜΥΕ003: Ανάκτηση Πληροφορίας

Διδάσκουσα: Ευαγγελία Πιτουρά
Περιγραφή Εργασίας

Ένα σύστημα αναζήτησης πληροφορίας για εστιατόρια και κριτικές εστιατορίων.

Πληροφορίες για εστιατόρια και κριτικές τους από το σύστημα **Yelp** (<https://www.yelp.com/>).

Ως πρώτο βήμα, δημιουργείστε τη συλλογή σας κατεβάζοντας δεδομένα από το **Yelp Open Dataset** (<https://www.yelp.com/dataset>).

Διαλέξτε ένα υποσύνολο των διαθέσιμων δεδομένων τα οποία να αφορούν εστιατόρια και κριτικές αυτών των εστιατορίων.

Ελάχιστες απαιτήσεις:

- 10000 εστιατόρια.
- 100000 κριτικές αυτών των εστιατορίων

Βασικές Ημερομηνίες

Πέμπτη 29 Μαρτίου 2018, Ορισμός ομάδων και περιγραφή δεδομένων

Πέμπτη 26 Απριλίου 2018, Περιγραφή αρχικού σχεδιασμού

Τετάρτη 23 Μαΐου 2018, Παράδοση εργασίας

Πέμπτη 24 ή/και Παρασκευή 25 Μαΐου 2018, Εξέταση εργασίας

Για την υλοποίηση, θα χρησιμοποιήσετε το σύστημα Lucene
(<https://lucene.apache.org/>)

Περιεχόμενα Παρουσίασης

Σύντομη παρουσίαση

- Lucene
- Yelp dataset
- Εργασίας

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
- Lets 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, Python (**PyLucene**: a Python port of the Core project), PHP, ...

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

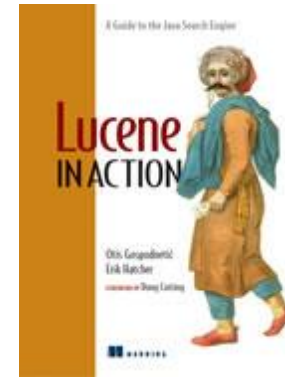
- **ranked** searching -- best results returned first
- many powerful **query types**: phrase queries, wildcard queries, proximity queries, range queries and more
- **fielded** searching (e.g. title, author, contents)
- **sorting** by any field
- allows **simultaneous update and searching**
- flexible **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

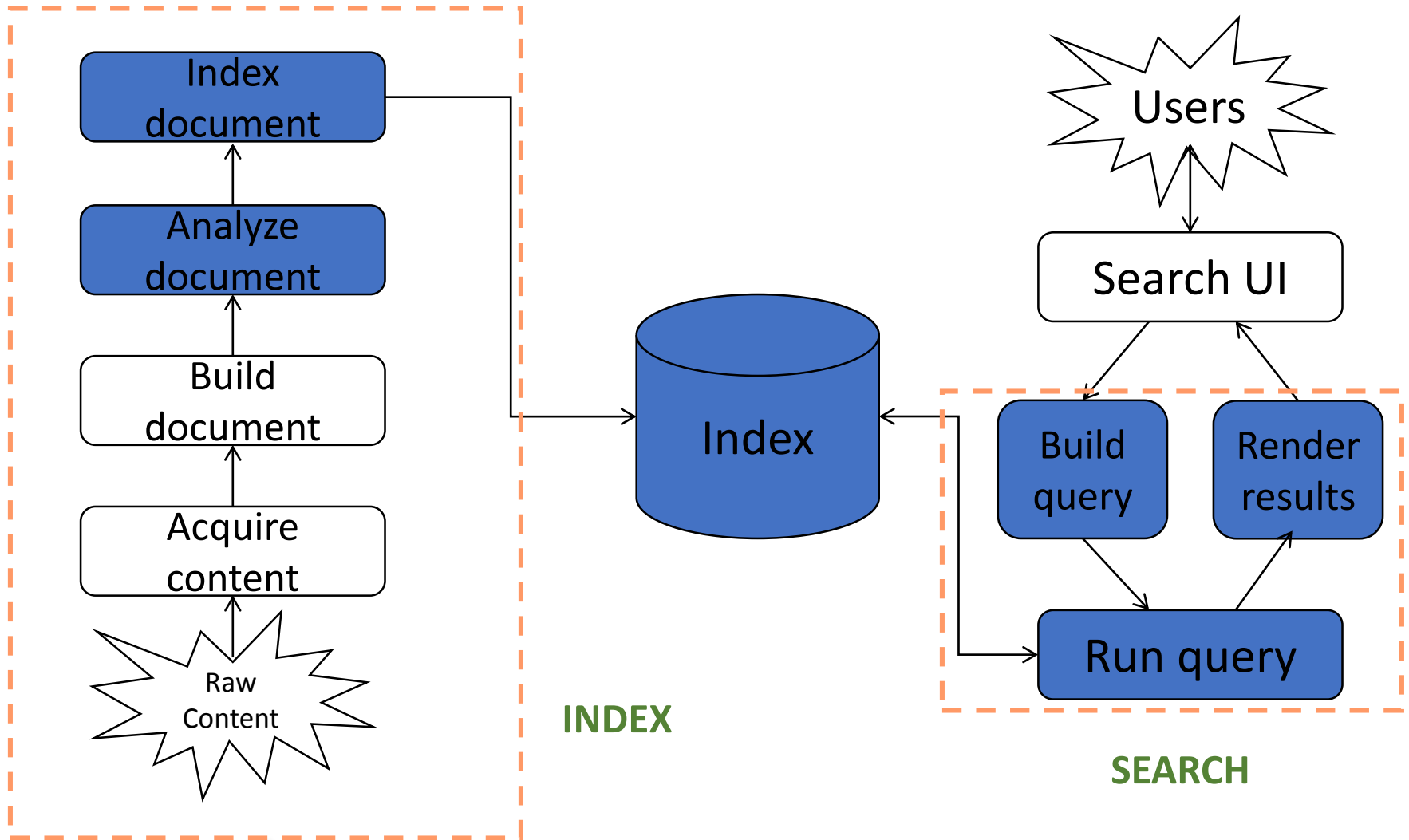
Περισσότερες πληροφορίες (προσοχή κάποια στοιχεία αναφέρονται σε παλιότερη έκδοση)

- <http://www.lucene-tutorial.com/>
- Lucene in Action



- **Lucene 7.2.1** demo API (recommended for more up-to-date code examples)
- offers simple example code to show the features of Lucene
 - https://lucene.apache.org/core/7_2_1/core/overview-summary.html#overview.description
 - https://lucene.apache.org/core/7_2_1/demo/overview-summary.html#overview_description
- Μπορείτε να χρησιμοποιήσετε παλαιότερη version αν θέλετε

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**.
- Consists of one or more **Fields**
 - A Field is a name-value pair.
example: title, body or metadata (creation time, etc)

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.

Βασικές έννοιες: Fields

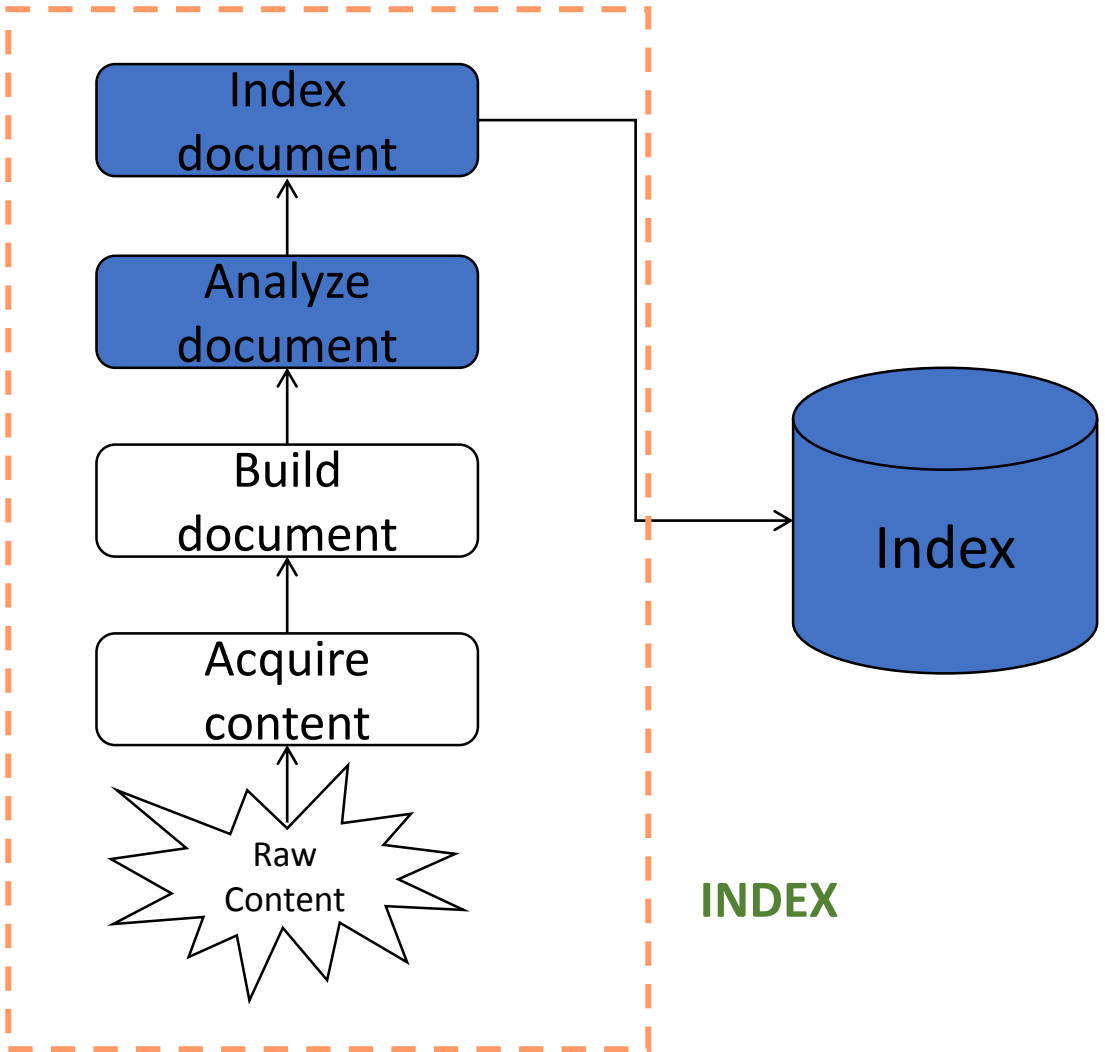
- You have to translate raw content into Fields
- Search a field using name:term, e.g., title:lucene

Lucene in a search system: **index**

Lucene in a search system: **index**

Steps

- 1. Acquire content
- 2. Build content
- 3. Analyze documents
- 4. Index documents



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/>

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/>

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();
```

```
// SERACH: 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();
```

Documents

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

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”

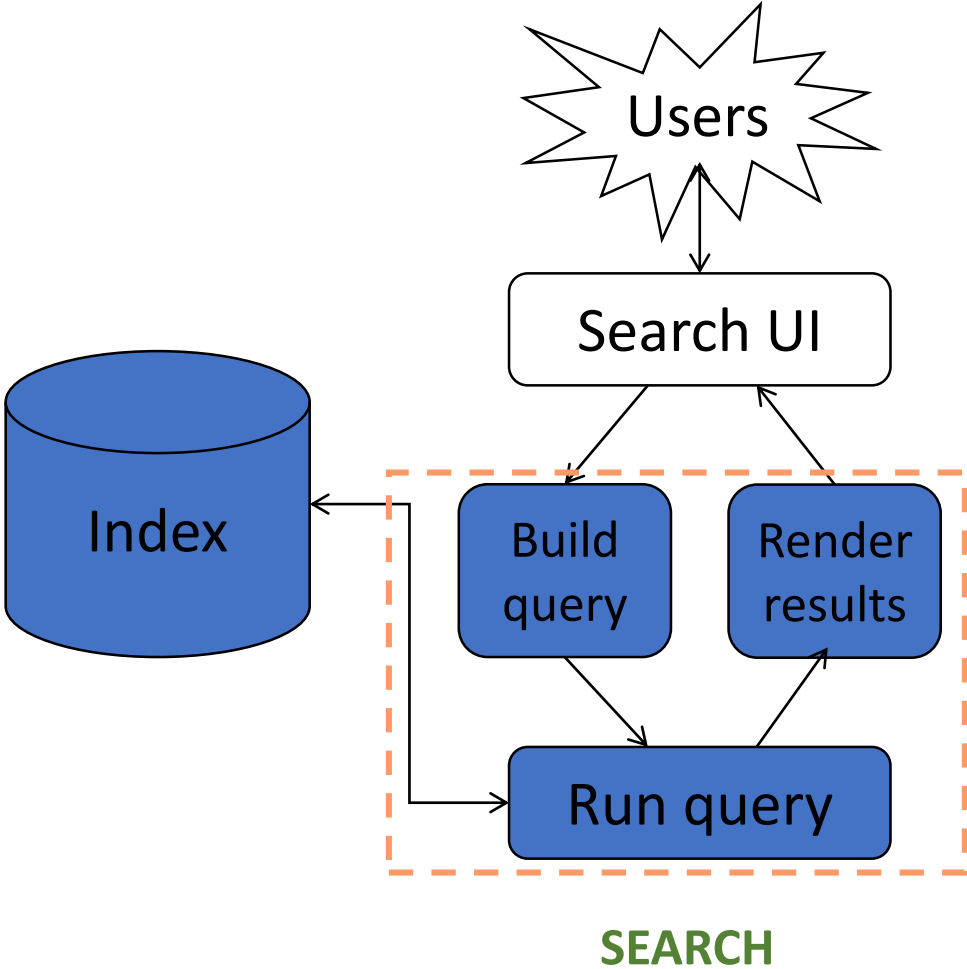
- `WhitespaceAnalyzer`
 - `[The] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dog]`
- `SimpleAnalyzer`
 - `[the] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dog]`
- `StopAnalyzer`
 - `[quick] [brown] [fox] [jumped] [over] [lazy] [dog]`
- `StandardAnalyzer`
 - `[quick] [brown] [fox] [jumped]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

Yelp dataset και απαιτήσεις εργασίας

Yelp dataset

Download from

<https://www.yelp.com/dataset>

(JSON, SQL)

Documentation

<https://www.yelp.com/dataset/documentation/json>

Yelp dataset: Businesses I

```
{ // string, 22 character unique string business id  
  "business_id": "tnhfDv5ll8EaGSXZGiuQGg",  
  // string, the business's name  
  "name": "Garaje",  
  // string, the neighborhood's name  
  "neighborhood": "SoMa",  
  // string, the full address of the business  
  "address": "475 3rd St",  
  // string, the city "city": "San Francisco",  
  // string, 2 character state code, if applicable  
  "state": "CA",  
  // string, the postal code  
  "postal code": "94107",  
  // float, latitude  
  "latitude": 37.7817529521,  
  // float, longitude "longitude": -122.39612197,
```



location

Yelp dataset: Businesses II

// float, star rating, rounded to half-stars

"stars": 4.5,

// interger, number of reviews

"review_count": 1198,

// integer, 0 or 1 for closed or open, respectively

"is_open": 1,

// object, business attributes to values. note: some attribute values might be objects

"attributes": { "RestaurantsTakeOut": true, "BusinessParking": { "garage": false, "street": true, "validated": false, "lot": false, "valet": false }, },

// an array of strings of business categories

"categories": ["Mexican", "Burgers", "Gastropubs"],

// an object of key day to value hours, hours are using a 24hr clock

"hours": { "Monday": "10:00-21:00", "Tuesday": "10:00-21:00", "Friday": "10:00-21:00", "Wednesday": "10:00-21:00", "Thursday": "10:00-21:00", "Sunday": "11:00-18:00", "Saturday": "10:00-21:00" } }

Yelp dataset: Reviews

```
{ // string, 22 character unique review id
  "review_id": "zdSx_SD6obEhz9VrW9uAWA",
  // string, 22 character unique user id, maps to the user in user.json
  "user_id": "Ha3iJu77CxlRfm-vQRs_8g",
  // string, 22 character business id, maps to business in business.json
  "business_id": "tnhfDv5ll8EaGSXZGiuQGg",
  // integer, star rating
  "stars": 4,
  // string, date formatted YYYY-MM-DD
  "date": "2016-03-09",
  // string, the review itself
  "text": "Great place to hang out after work: the prices are decent, and the
  ambience is fun. It's a bit loud, but very lively. The staff is friendly, and the food
  is good. They have a good selection of drinks.",
  // integer, number of useful, funny, cool votes received
  "useful": 0,
  "funny": 0,
  "cool": 0 }
```

Yelp dataset: other

User data including the user's friend mapping and all the metadata associated with the user.

Checkins on a business.

Tips written by a user on a business. Tips are shorter than reviews and tend to convey quick suggestions.

Photos

Εργασία

Λειτουργικότητα 1: Αναζήτηση εστιατορίων

Το σύστημα σας θα πρέπει να υποστηρίζει αναζήτηση εστιατορίων (δηλαδή, το αποτέλεσμα θα είναι ένα ή περισσότερα εστιατόρια) *τουλάχιστον* με βάση:

- Τη γεωγραφική θέση του εστιατορίου,
- Το *πλήρες κείμενο* των κριτικών του εστιατορίου (για παράδειγμα εστιατόρια των οποίων οι κριτικές περιλαμβάνουν τη λέξη «sesame»),
- Συνδυασμό των παραπάνω.

Ο βασικός τρόπος διάταξης θα πρέπει να είναι με βάση το κείμενο. Επιπρόσθετα, θα παρέχετε η δυνατότητα αναδιάταξης με βάση τουλάχιστον (1) τον αριθμό των κριτικών που έχει λάβει κάθε εστιατόριο και (2) τον αριθμό αστεριών του.

Εργασία

Λειτουργικότητα 2: Αναζήτηση κριτικής

Το σύστημα σας θα πρέπει να υποστηρίζει αναζήτηση κριτικών εστιατορίων (δηλαδή, το αποτέλεσμα θα είναι μια ή περισσότερες κριτικές εστιατορίων) τουλάχιστον με βάση

- Το όνομα του εστιατορίου,
- Λέξεις κλειδιά,
- Συνδυασμό των παραπάνω.

Ο βασικός τρόπος διάταξης θα πρέπει να είναι με βάση το κείμενο. Επιπρόσθετα, θα παρέχετε η δυνατότητα αναδιάταξης με βάση τουλάχιστον (1) το πόσο σημαντική είναι η κριτική (πχ, useful count), και (2) το χρόνο: η πιο πρόσφατη κριτική να εμφανίζεται πρώτη.

Εργασία

Λειτουργικότητα 3: Παρουσίαση αντιπροσωπευτικών αποτελεσμάτων

Επεκτείνετε την αναζήτηση εστιατορίων και κριτικών εστιατορίων ώστε να παρουσιάζεται ως απάντηση στο χρήστη ένα αντιπροσωπευτικό υποσύνολο των αποτελεσμάτων.

Για παράδειγμα στην περίπτωση των εστιατορίων, μερικές ιδέες είναι να παρουσιάζονται εστιατόρια σε: (1) σε διαφορετικές τοποθεσίες, (2) με διαφορετικές κουζίνες, (3) με διαφορετικό αριθμό αστεριών (4) οποιοδήποτε συνδυασμός των παραπάνω.

Για παράδειγμα στην περίπτωση των κριτικών εστιατορίων, μερικές ιδέες είναι (1) ένα μείγμα θετικών και αρνητικών κριτικών, (2) κριτικών που να αναφέρονται σε διαφορετικά θέματα, (3) πρόσφατων και λιγότερων πρόσφατων κριτικών, (4) οποιοδήποτε συνδυασμός των παραπάνω.

Μπορείτε επίσης να χρησιμοποιείτε και τα δεδομένα για τους χρήστες ώστε να παρέχετε κριτικές από «διαφορετικούς» χρήστες.

Ερωτήσεις;