

# Information Search and Retrieval

LAB6 – Luke and  
tuning the Apache Lucene



# The goal of this lab

---

- To better understand Lucene and IR systems:
  - we will use Luke to explore index and our own implementations of custom parts of Lucene
  - we will design our own analyzer and see how it works
  - Lucene benchmarking (evaluation)
  - we will design our own scoring function and see how it works
  - how does spelling work in Lucene?

# What do we need for this lab?

---

- Theoretical background (lectures)
- To be able to run Lucene demo
  - Lab 4 material on how to use Apache Lucene
  - Create an index on lucene 3.0.3 java doc API

# Obtaining Luke

---

- We will use Luke 1.0.1 f luke
- Download Luke bundle from here:
  - <http://code.google.com/p/luke/>
  - or
  - <http://www.inf.unibz.it/~lbaltrunas/lukeall-1.0.1.jar>
- Make an index using Lucene 3.0.3
  - You did it in Lab 4.
- Run Luke
  - `java -jar lukeall-1.0.1.jar`
  - Open the previously created index

# Luke Demo

The screenshot shows the Luke - Lucene Index Toolbox application. The title bar reads "Luke - Lucene Index Toolbox, v 1.0.1 (2010-03-05)". The menu bar includes "File", "Tools", "Settings", and "Help". The toolbar contains icons for "Overview", "Documents", "Search", "Files", and "Plugins".

The main interface is divided into several sections:

- Analyzer Tool:** A section for analyzing analyzers, by Mark Harwood (<mailto:mharwood@apache.org>).
- Hadoop Plugin:** Shows available analyzers found on the current classpath and a compatibility dropdown set to "LUCENE\_CURRENT".
- Scripting Luke:** A dropdown menu showing "org.apache.lucene.analysis.WhitespaceAnalyzer".
- Custom Similarity:** A section for text to be analyzed.
- Vocabulary Analysis Tool:** A section for analyzing text. It contains a text area with the following text: "Welcome to the analysis viewer. This tool is used to demonstrate how different analyzers process text into tokens. You can edit this text to try different input such as numbers like 23231.23 or characters (mharwood@apache.org). Once happy, select an Analyzer from the list of analyzers found on the current classpath and then hit the Analyze button. The tokens produced are shown below and when you select them the right panel shows their attributes, and the corresponding span in the original text is highlighted." Below the text area is an "Analyze" button.
- Zipf distributions:** A section for analyzing text.

Below the "Analyze" button, there are two panels:

- Tokens created by t:** A list of tokens: "Welcome", "to", "the", "analysis", "viewer.", "This", "tool", "is", "used", "to", "demonstrate", "how", "different", "analyzers", "process", "text", "into".
- Token attributes:** A table showing the attributes for the selected token "Welcome".

Class	Implementation	Value
OffsetAttribute	OffsetAttributeImpl	0,7
TermAttribute	TermAttributeImpl	Welcome

The status bar at the bottom shows "Index name: /Users/mumas/index".

# Investigate an Index

---

- ❑ Delete a document
- ❑ What is the next term in the dictionary for the filed:contents that comes after "apache"?
- ❑ Which documents contain these terms?

# Analyzers

---

- ❑ Check how different analyzers work
- ❑ Implement your own simple analyzer and add it to the luke
  - remove common html tags from the indexed words
  - `java -classpath demo-analyzer.jar:lukeall-1.0.1.jar org.getopt.luke.Luke`
  - (on windows use ; instead of : )
- ❑ Implement n-gram analyzer

```
public class NGramAnalyzer extends Analyzer{  
  
    @Override  
    public TokenStream tokenStream(String arg0, Reader arg1) {  
        NGramTokenizer tokenizer = new NGramTokenizer(arg1, 3, 4);  
        return new NGramTokenFilter(tokenizer);  
    }  
  
}
```

# Benchmarking

---

- ❑ Get lucene **source** distribution (lucene-3.0.3-src.jar)
  - go to lucene-3.0.3-src/contrib/benchmark
  - run
    - ❑ ant run-task
    - ❑ ant run-task -Dtask.alg=conf/analyzer.alg
- ❑ Check available benchmarks in
  - lucene-3.0.1/contrib/benchmark/conf
- ❑ If you would need to run your own benchmarks
  - [http://lucene.apache.org/java/3\\_0\\_3/api/contrib-benchmark/org/apache/lucene/benchmark/byTask/package-summary.html](http://lucene.apache.org/java/3_0_3/api/contrib-benchmark/org/apache/lucene/benchmark/byTask/package-summary.html)

# Scoring (recap.)

---

- ❑ make sure you read [http://lucene.apache.org/java/3\\_0\\_3/scoring.html](http://lucene.apache.org/java/3_0_3/scoring.html)
  - In Lucene, the objects we are scoring are Documents
  - Lucene scoring works on **Fields** and then **combines** the results to return Documents
  - Lucene allows influencing search results by "boosting" in more than one level
  - Lucene combines Boolean model (BM) with Vector Space Model (VSM) of Information Retrieval
    - ❑ docs approved by BM are scored with VSM
    - ❑ [http://lucene.apache.org/java/3\\_0\\_3/api/core/org/apache/lucene/search/Similarity.html](http://lucene.apache.org/java/3_0_3/api/core/org/apache/lucene/search/Similarity.html)

# Changing default scoring

---

- ❑ You can change default scoring by rewriting
  - `org.apache.lucene.search.Similarity`
  - and setting it to `org.apache.lucene.search.Searcher.setSimilarity(Similarity similarity)`
- ❑ Implement your own similarity that does not take into account idf value
- ❑ Load it in Luke
  - search for a documents containing common term
  - check the explanations for the search

# Spelling

---

- ❑ Modify searching demo in such a way, that:
  - if 0 records returned because of a misspelled word
  - you would suggest rerun the search with the list of 5 words from the dictionary with the closest spelling
- ❑ You will need
  - <http://wiki.apache.org/lucene-java/SpellChecker>
  - lucene-3.0.3/contrib/spellchecker/lucene-spellchecker-3.0.3.jar
  - example of use:
    - ❑ lucene-3.0.3-src/contrib/spellchecker/src/test/org/apache/lucene/search/spell/TestPlainTextDictionary.java