

Lucene

Jianguo Lu

School of Computer Science

University of Windsor

A Comparison of Open Source Search Engines for 1.69M Pages

Search Engine	Storage ^(f)	Increm. Index	Results Excerpt	Results Template	Stop words	Filetype ^(e)	Stemming	Fuzzy Search	Sort ^(d)	Ranking	Search Type ^(c)	Indexer Lang. ^(b)	License ^(a)
Datapark	2	■	■	■	■	1,2,3	■	■	1,2	■	2	1	4
ht://Dig	1	■	■	■	■	1,2	■	■	1	■	2	1,2	4
Indri	1	■	■	■	■	1,2,3,4	■	■	1,2	■	1,2,3	2	3
IXE	1	■	■	■	■	1,2,3	□	■	1,2	■	1,2,3	2	8
Lucene	1	■	□	□	■	1,2,4	■	■	1	■	1,2,3	3	1
MG4J	1	■	■	■	■	1,2	■	□	1	■	1,2,3	3	6
mnoGoSearch	2	■	■	■	■	1,2	■	■	1	■	2	1	4
Namazu	1	■	■	■	□	1,2	□	□	1,2	■	1,2,3	1	4
Omega	1	■	□	■	■	1,2,4,5	■	□	1	■	1,2,3	2	4
OmniFind	1	■	■	■	■	1,2,3,4,5	■	■	1	■	1,2,3	3	5
OpenFTS	2	■	□	□	■	1,2	■	■	1	■	1,2	4	4
SWISH-E	1	■	□	□	■	1,2,3	■	■	1,2	■	1,2,3	1	4
SWISH++	1	■	□	□	■	1,2	■	□	1	■	1,2,3	2	4
Terrier	1	□	□	□	■	1,2,3,4,5	■	■	1	■	1,2,3	3	7
WebGlimpse	1	■	■ ^(g)	■ ^(g)	□	1,2	□	■	1 ^(e)	■	1,2,3	1	8,9
XMLSearch	1	■	□	□	■	3	□	■	3	□	1,2,3	2	8
Zettair	1	■	■	□	■	1,2	■	□	1	■	1,2,3	1	2

^(a) 1:Apache,2:BSD,3:CMU,4:GPL,5:IBM,6:LGPL,7:MPL,8:Comm,9:Free
^(b) 1:C, 2:C++, 3:Java, 4:Perl, 5:PHP, 6:Tcl
^(c) 1:phrase, 2:boolean, 3:wild card.
^(d) 1:ranking, 2:date, 3:none.
^(e) 1:HTML, 2:plain text, 3:XML, 4:PDF, 5:PS.
^(f) 1:file, 2:database.
^(g) Commercial version only.

■ Available
 □ Not Available

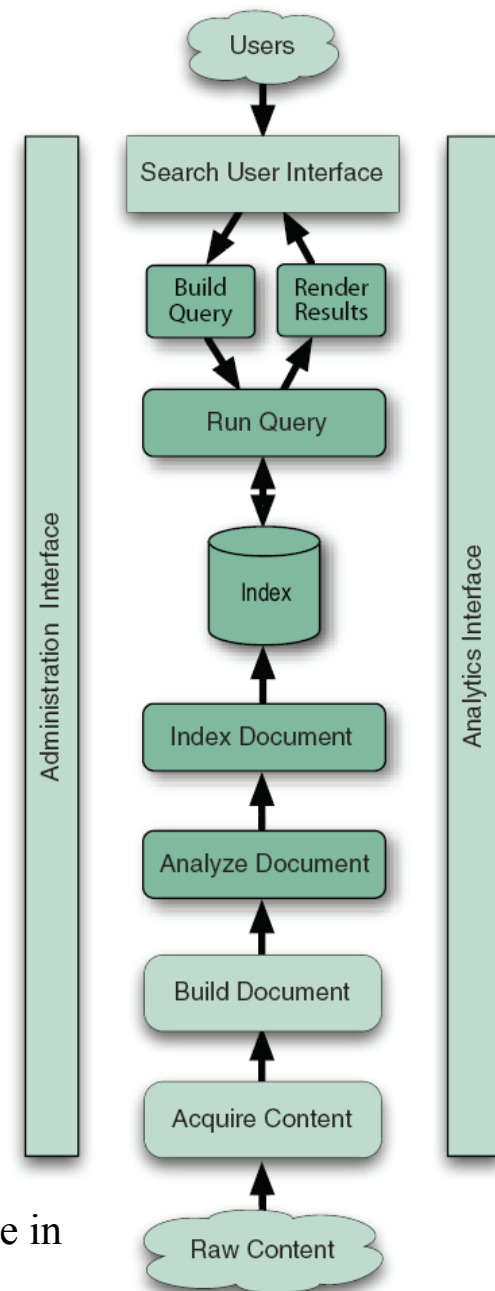
Lucene

- Developed by Doug Cutting initially
 - Java-based. Created in 1999, Donated to Apache in 2001
- Features
 - No crawler, No document parsing, No “PageRank”
- Websites Powered by Lucene
 - IBM Omnifind Y! Edition, Technorati
 - Wikipedia, Internet Archive, LinkedIn, monster.com
- Add documents to an index via IndexWriter
 - A document is a collection of fields
 - Flexible text analysis – tokenizers, filters
- Search for documents via IndexSearcher
 - Hits = search(Query,Filter,Sort,topN)
- Ranking based on $tf * idf$ similarity with normalization

What is lucene

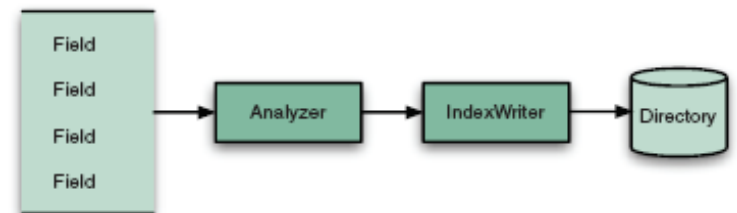
- Lucene is
 - an API
 - an Information Retrieval Library
- Lucene is not an application ready for use
 - Not a web server
 - Not a search engine
- It can be used to
 - Index files
 - Search the index
- It is open source, written in Java
- Two stages: index and search

Picture From Lucene in Action



Indexing

- Just the same as the index at the end of a book
- Without an index, you have to search for a keyword by scanning all the pages of a book
- Process of indexing
 - Acquire content
 - Say, semantic web DBPedia
 - Build document
 - Transform to text file from other formats such as pdf, ms word
 - Lucene does not support this kind of filter
 - There are tools to do this
 - Analyze document
 - Tokenize the document
 - Stemming
 - Stop words
 - Lucene provides a string of analyzers
 - User can also customize the analyzer
 - Index document
- Key classes in Lucene Indexing
 - Document, Analyzer, IndexWriter



Code snippets

```
Directory dir = FSDirectory.open(new File(indexDir));
```

```
writer = new IndexWriter(  
    dir, new StandardAnalyzer(Version.LUCENE_30),  
    true,  
    IndexWriter.MaxFieldLength.UNLIMITED);
```

Index is written
in this directory

When doc is added,
Use StandardAnalyzer

... ..

```
Document doc = new Document();  
doc.add(new Field("contents", new FileReader(f)));  
doc.add(new Field("filename", f.getName(),  
    Field.Store.YES, Field.Index.NOT_ANALYZED));  
doc.add(new Field("fullpath", f.getCanonicalPath(),  
    Field.Store.YES, Field.Index.NOT_ANALYZED));
```

Create a document
instance from a file

```
writer.addDocument(doc);
```

Add the doc to writer

Document and Field

```
doc.add(new Field("fullpath", f.getCanonicalPath(),  
                Field.Store.YES, Field.Index.NOT_ANALYZED));
```

- **Construct a Field:**

- First two parameters are field name and value
- Third parameter: whether to store the value
 - If NO, content is discarded after it indexed. Storing the value is useful if you need the value later, like you want to display it in the search result.
- Fourth parameter: whether and how the field should indexed.

```
doc.add(new Field("contents", new FileReader(f)));
```

- Create a tokenized and indexed field that is not stored

Lucene analyzers

- **StandardAnalyzer**
 - A sophisticated general-purpose analyzer.
- **WhitespaceAnalyzer**
 - A very simple analyzer that just separates tokens using white space.
- **StopAnalyzer**
 - Removes common English words that are not usually useful for indexing.
- **SnowballAnalyzer**
 - A stemming that works on word roots.
- **Analyzers for languages other than English**

Examples of analyzers

```
%java lia.analysis.AnalyzerDemo "No Fluff, Just Stuff"
```

Analyzing "No Fluff, Just Stuff"

org.apache.lucene.analysis.WhitespaceAnalyzer:

[No] [Fluff,] [Just] [Stuff]

org.apache.lucene.analysis.SimpleAnalyzer:

[no] [fluff] [just] [stuff]

org.apache.lucene.analysis.StopAnalyzer:

[fluff] [just] [stuff]

org.apache.lucene.analysis.standard.StandardAnalyzer:

[fluff] [just] [stuff]

StandardAnalyser

- support
 - company name
 - XY&Z corporation → [XY&Z] [corporation]
 - Email
 - xyz@example.com --> xyz@example.com
 - Ip address
 - Serial numbers
- Support chinese and japanese

StopAnalyser

- Default stop words:

- "a", "an", "and", "are", "as", "at", "be", "but", "by", "for", "if", "in", "into", "is", "it", "no", "not", "of", "on", "or", "such", "that", "the", "their", "then", "there", "these", "they", "this", "to", "was", "will", "with"
- Note that there are other stop-word lists

- You can pass your own set of stop words

Customized analyzers

- Most applications do not use built-in analyzers
- Customize
 - Stopwords
 - Application specific tokens (product part number)
 - Synonym expansion
 - Preserve case for certain tokens
 - Choosing stemming algorithm
- Solr configure the tokenizing using solrconfig.xml

N-gram filter

```
private static class NGramAnalyzer extends Analyzer {  
    public TokenStream tokenStream(String fieldName, Reader reader) {  
        return new NGramTokenFilter(new KeywordTokenizer(reader), 2, 4);  
    }  
}
```

Lettuce →

1: [le]

2: [et]

3: [tt]

4: [tu]

5: [uc]

6: [ce]

...

Example of SnowballAnalyzer

- stemming

- English

```
Analyzer analyzer = new SnowballAnalyzer(Version.LUCENE_30, "English");
```

```
AnalyzerUtils.assertAnalyzesTo(analyzer, "stemming algorithms", new String[]  
{"stem", "algorithm"});
```

- Spanish

```
Analyzer analyzer = new SnowballAnalyzer(Version.LUCENE_30, "Spanish");
```

```
AnalyzerUtils.assertAnalyzesTo(analyzer, "algoritmos", new String[] {"algoritmo"});
```

Lucene tokenizers and filters

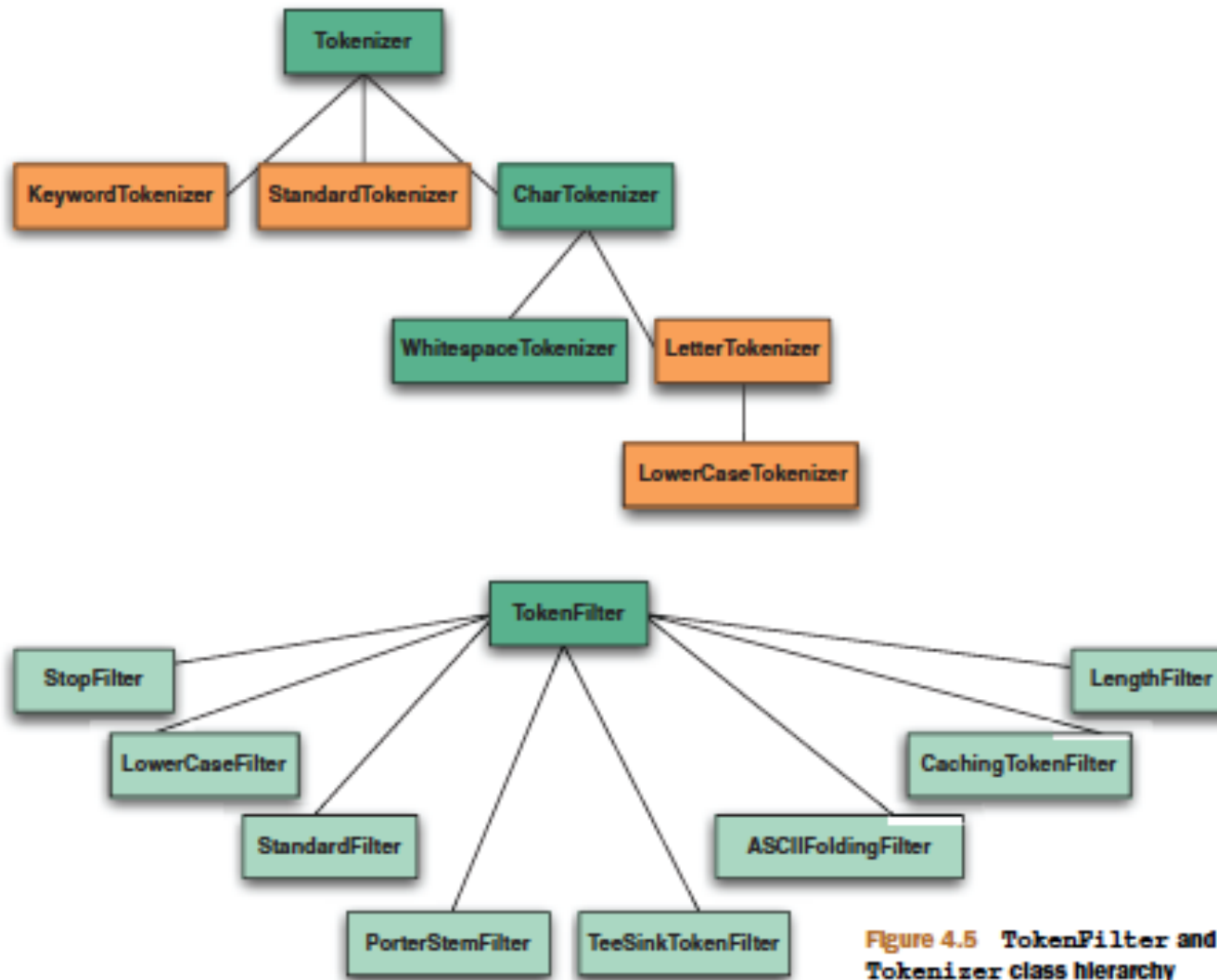
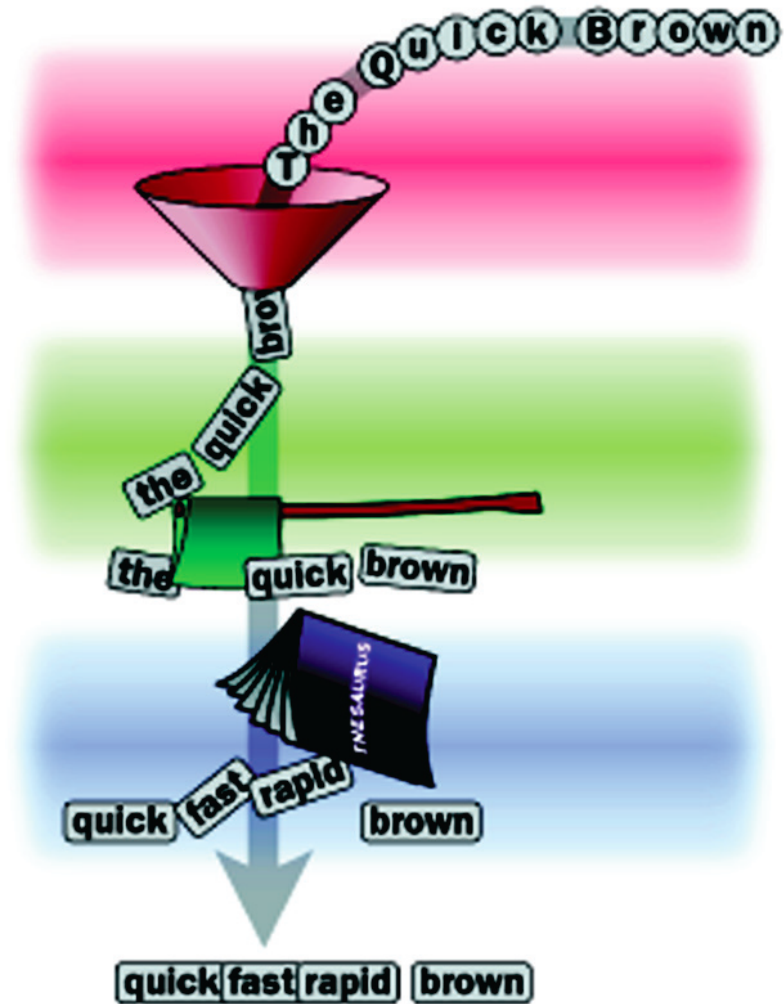
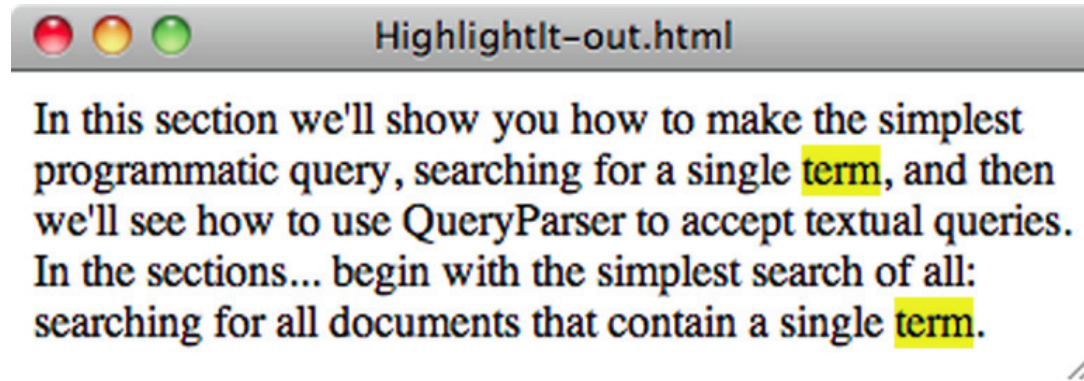


Figure 4.5 TokenFilter and Tokenizer class hierarchy

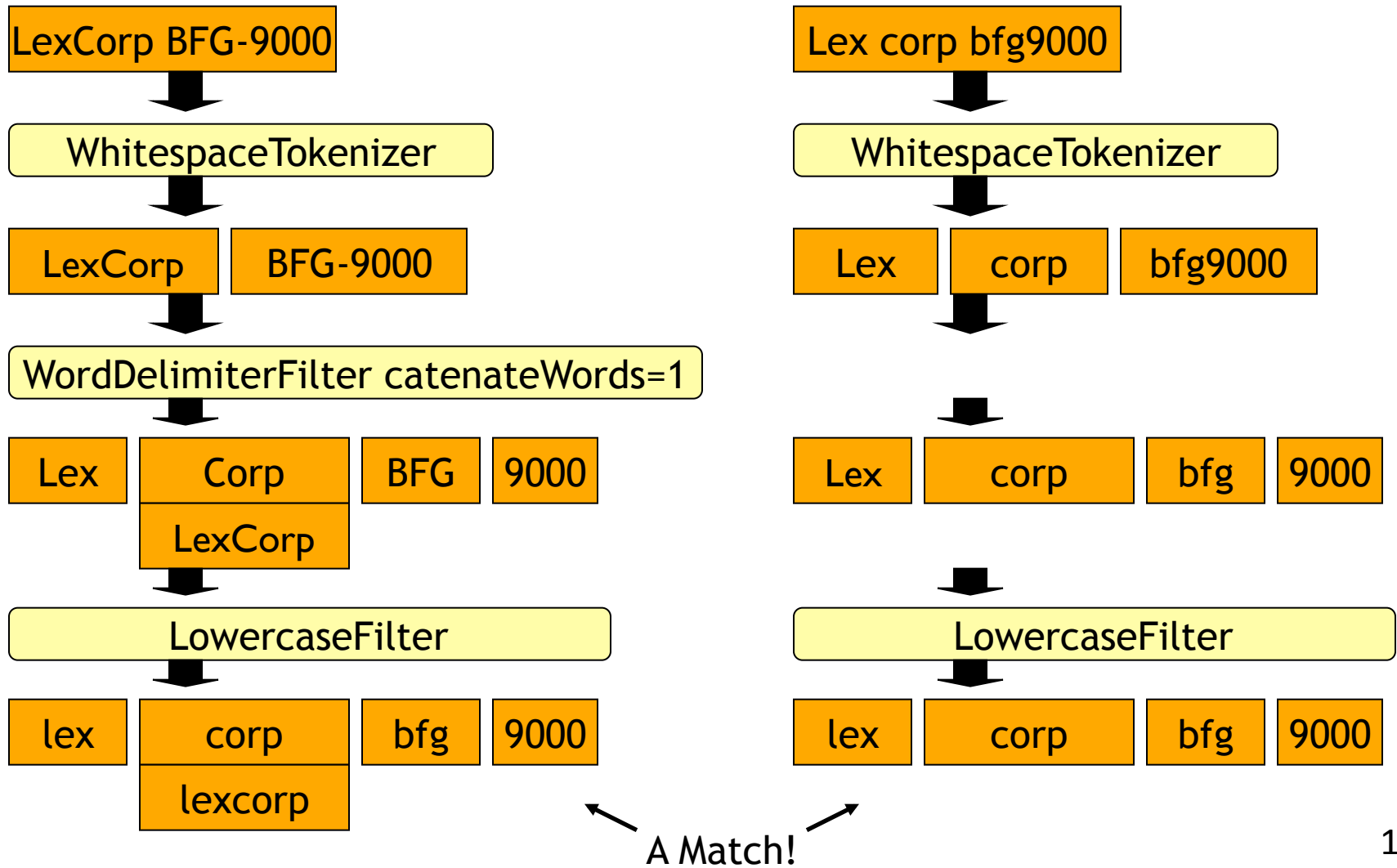
Synonym filter



Highlight query terms



Analyzers (solr)




Search

- Three models of search
 - Boolean
 - Vector space
 - Probabilistic
- Lucene supports a combination of boolean and vector space model
- Steps to carry out a search
 - Build a query
 - Issue the query to the index
 - Render the returns
 - Rank pages according to relevance to the query

Search code

```
Directory dir = FSDirectory.open(new File(indexDir));  
IndexSearcher is = new IndexSearcher(dir);
```

Indicate to search
which index




```
QueryParser parser = new QueryParser(Version.LUCENE_30, "contents",  
    new StandardAnalyzer(Version.LUCENE_30));  
Query query = parser.parse(q);
```

Parse the query




```
TopDocs hits = is.search(query, 10);  
for(ScoreDoc scoreDoc : hits.scoreDocs) {  
    Document doc = is.doc(scoreDoc.doc);  
    System.out.println(doc.get("fullpath"));  
}
```

Search the query



Process the returns one
by one. Note that
'fullpath' is a field added
while indexing



Result ranking

- The default is relevance ranking based on tf.idf
- Can customize the ranking
 - Sort by index order
 - Sort by one or more attributes
- Pageranking is not used

What if the document is not a text file?

Extracting text with Apache Tika

- A toolkit detects and extracts metadata and text from over a thousand different file types
- There are many document formats
 - rtf, pdf, ppt, outlook, flash, open office
 - Html, xml
 - Zip, tar, gzip, bzip2, ..
- There are many document filters
 - Each has its own api
- A framework that hosts plug-in parsers for each document type
 - Standard api for extracting text and meta data
- Tika itself does not parse documents

Extracting from various file formats

- Microsoft's OLE2 Compound Document Format (Excel, Word, PowerPoint, Visio, Outlook)
 - Apache POI
- Microsoft Office 2007 OOXML
 - Apache POI
- Adobe Portable Document Format (PDF)
 - PDFBox
- Rich Text Format (RTF)—currently body text only (no metadata)
 - Java Swing API (RTFEditorKit)
- Plain text character set detection ICU4J library
 - HTML CyberNeko library
- XML
 - Java's javax.xml classes

Compressed files

- ZIP Archives
 - Java's built-in zip classes, Apache Commons Compress
- TAR Archives
 - Apache Ant, Apache Commons Compress
- AR Archives, CPIO Archives
 - Apache Commons Compress
- GZIP compression
 - Java's built-in support (GZIPInputStream) , Apache Commons Compress
- BZIP2 compression
 - Apache Ant, Apache Commons Compress

multimedia

- Image formats (metadata only)
 - Java's javax.imageio classes
- MP3 audio (ID3v1 tags)
 - Implemented directly
- Other audio formats (wav, aiff, au) J
 - Java's built-in support (javax.sound.*)
- OpenDocument
 - Parses XML directly
- Adobe Flash
 - Parses metadata from FLV files directly
- MIDI files (embedded text, eg song lyrics)
 - Java's built-in support (javax.audio.midi.*)
- WAVE Audio (sampling metadata)
 - Java's built-in support (javax.audio.sampled.*)

Program code

- Java class files
 - ASM library (JCR-1522)
- Java JAR files
 - Java's built-in zip classes and ASM library, Apache Commons Compress

Example:

Search engine of source code (e.g., from github)

New requirements

- Tokenization

- Variable names
 - `deletionPolicy` vs. `deletion policy` vs. `deletion_policy`
- Symbols: normally discarded symbols now important
 - `For (int I =0;i<10;i++)`
- Stop words
 - Better to keep all the words

- Common tokens

- `(,`
- Group into bigrams, to improve performance
 - `if (`

Query syntax

- Search requirements
 - Search for class or method names
 - Search for Structurally similar code
 - Search for questions and answers (e.g. from stackoverflow.com)
 - Deal with Includes and imports
- The query syntax may no longer be keywords search or boolean search
- Need to parse the source code
 - Different programming language has different grammar and parser
 - Parser generators

Substring search

- Wildcard search is more important
- Names in programs are often combinations of several words
- Example: Search for ldap
- Source code contains getLDAPconfig
- The query is: *ldap*
- Implementation
 - new approach of tokenization. getLDAPconfig → [get] [LDAP] [config]
 - Permuterm index
 - Ngram
 - getLDAPconfig → Get etl tld ...

Krugle: source code search

- Described in the book Lucene in Action
- Indexed millions of lines of code

SIREN

- Searching semi-structured documents
- Case study 2 of the lucene book
- Semantic information retrieval
- 50 million structured data
- 1 billion rdf triples

Case study 3 of the lucene book

- Search for people in LinkedIn
- Interactive query refinement
- e.g., search for java programmer

Related tools

- Apache Solr: is the search server built on top of Lucene
 - Lucene focuses on the indexing, not a web application
 - Build web-based search engines
- Apache Nutch: for large scale crawling
 - Can run on multiple machines
 - Support regular expression to filter what is needed
 - support .robots.txt

Our starting point

- Index academic papers
- Start with a subset of the data (10K papers)
- Our course web site has
 - a link to the data
 - simple code for indexing and searching

```
public class IndexAllFilesInDirectory {  
    static int counter = 0;  
    public static void main(String[] args) throws Exception {  
        String indexPath = "/Users/jianguolu/data/citeseer2_index";  
        String docsPath = "/Users/jianguolu/data/citeseer2";  
        System.out.println("Indexing to directory '" + indexPath + "'...");  
        Directory dir = FSDirectory.open(Paths.get(indexPath));  
        IndexWriterConfig iwc = new IndexWriterConfig(new StandardAnalyzer());  
        IndexWriter writer = new IndexWriter(dir, iwc);  
        indexDocs(writer, Paths.get(docsPath));  
        writer.close();  
    }  
}
```

```
static void indexDocs(final IndexWriter writer, Path path) throws IOException {  
    Files.walkFileTree(path, new SimpleFileVisitor<Path>() {  
        public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws IOException {  
            indexDoc(writer, file);  
            return FileVisitResult.CONTINUE;  
        }  
    });  
}
```

```
static void indexDoc(IndexWriter writer, Path file) throws IOException {  
    InputStream stream = Files.newInputStream(file);  
    BufferedReader br = new BufferedReader(new InputStreamReader(stream,  
StandardCharsets.UTF_8));  
    String title = br.readLine();  
    Document doc = new Document();  
    doc.add(new StringField("path", file.toString(), Field.Store.YES));  
    doc.add(new TextField("contents", br));  
    doc.add(new StringField("title", title, Field.Store.YES));  
    writer.addDocument(doc);  
    counter++;  
    if (counter % 1000 == 0)  
        System.out.println("indexing " + counter + "-th file " + file.getFileName());  
}  
}
```

Search documents

```
public class SearchIndexedDocs {  
    public static void main(String[] args) throws Exception {  
        String index = "/Users/jianguolu/data/citeseer2_index";  
        IndexReader reader = DirectoryReader.open(FSDirectory.open(Paths.get(index)));  
        IndexSearcher searcher = new IndexSearcher(reader);  
        QueryParser parser = new QueryParser("contents", new StandardAnalyzer());  
        Query query = parser.parse("information AND retrieval");  
        TopDocs results = searcher.search(query, 6);  
        System.out.println(results.totalHits + " total matching documents");  
        for (int i = 0; i < 6; i++) {  
            Document doc = searcher.doc(results.scoreDocs[i].doc);  
            System.out.println((i + 1) + ". " + doc.get("path") + "\n\t" + doc.get("title"));  
        }  
        reader.close();  
    }  
}
```