Lucene

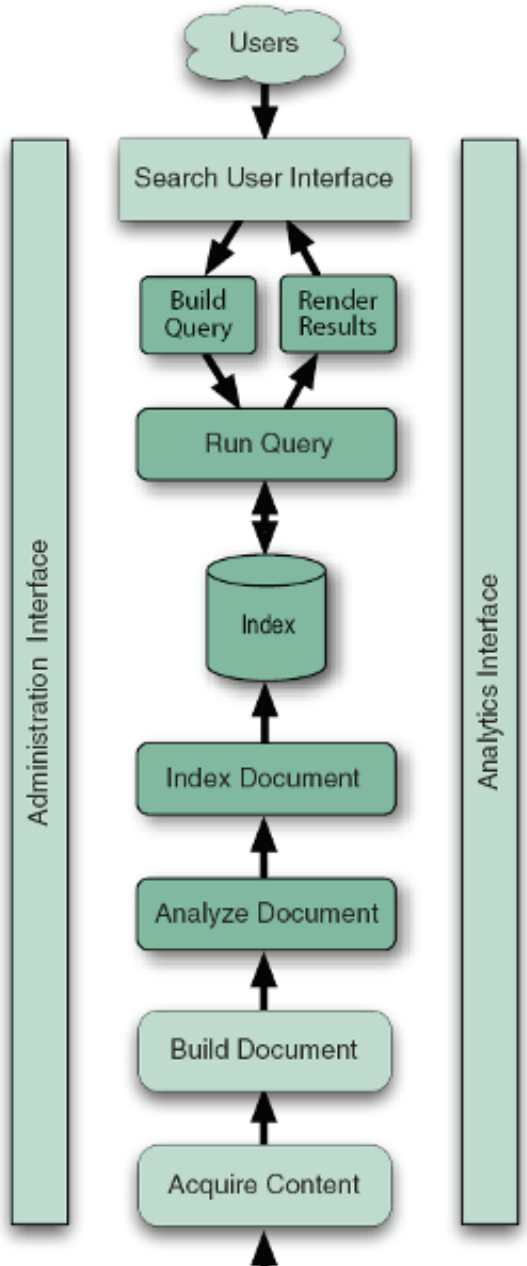
Jianguo Lu

September 15, 2024

Overview

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



- 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, 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

Code Snippets

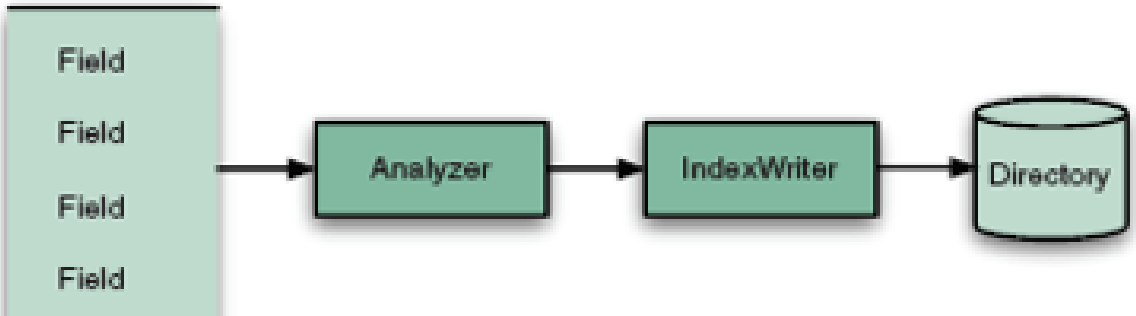
```
1  static void indexDoc(IndexWriter writer, Path file) throws Exception {
2    InputStream stream = Files.newInputStream(file);
3    BufferedReader br = new BufferedReader(new InputStreamReader(stream,
         StandardCharsets.UTF_8));
4    String title = br.readLine();
5    Document doc = new Document();
6    doc.add(new StringField("path", file.toString(), Field.Store.YES));
7    doc.add(new TextField("contents", br));
8    doc.add(new StringField("title", title, Field.Store.YES));
9    writer.addDocument(doc);
10   counter++;
11   if (counter % 1000 == 0)
12     ^^ISystem.out.println("indexing " + counter + "-th file " + file.
           getFileName());
13   }
14 }
```

Search Documents

```
1  public class SearchIndexedDocs {
2  public static void main(String[] args) throws Exception {
3    String index = "/Users/jianguolu/data/citeseer2_index";
4    IndexReader reader = DirectoryReader.open(FSDirectory.open(Paths.get(
         index)));
5    IndexSearcher searcher = new IndexSearcher(reader);
6    QueryParser parser = new QueryParser("contents", new StandardAnalyzer
         ());
7    Query query = parser.parse("information AND retrieval");
8    TopDocs results = searcher.search(query, 6);
9    System.out.println(results.totalHits + " total matching documents");
10   for (int i = 0; i < 6; i++) {
11      Document doc = searcher.doc(results.scoreDocs[i].doc);
12      System.out.println((i + 1) + ". " + doc.get("path") +"\n\t"+ doc.
            get("title"));^^I
13   }
14   reader.close();
15 }}
```

Indexing

- Just the same as the index at the end 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`



Indexing Code Snippet

```
1  Directory dir = FSDirectory.open(new File(indexDir));
2
3  writer = new IndexWriter(
4           dir, new StandardAnalyzer(Version.LUCENE_30),
5           true,
6           IndexWriter.MaxFieldLength.UNLIMITED);……
7
8
9  Document doc = new Document();
10 doc.add(new Field("contents", new FileReader(f)));
11 doc.add(new Field("filename", f.getName(),
12                Field.Store.YES, Field.Index.NOT_ANALYZED));
13 doc.add(new Field("fullpath", f.getCanonicalPath(),
14                Field.Store.YES, Field.Index.NOT_ANALYZED));
15 writer.addDocument(doc);
```

- Key classes in Lucene Indexing
  - Document, Analyzer, IndexWriter

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 TextField("contents", new FileReader(f)));
```

- Newer version uses `TextField`
- 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

Example of Analyzers

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]
```

Standard analyzer

Support

- company names
  XY&Z corporation $\Rightarrow$ [*XY&Z*][*corporation*]
- Email
  xyz@example.com $\Rightarrow$ xyz@example.com
- IP addresses
- Serial numbers
- ...

Support Chinese, Japanese, ...

StopAnalyser

- Default stop words in Lucene:
  `"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

```
1 Analyzer analyzer = new SnowballAnalyzer(Version.LUCENE_30, "English")
      ;
2 AnalyzerUtils.assertAnalyzesTo(analyzer, "stemming "algorithms, new
      String[] {"stem", "algorithm"});
```

Spanish

```
1 Analyzer analyzer = new SnowballAnalyzer(Version.LUCENE_30, "Spanish");
2 AnalyzerUtils.assertAnalyzesTo(analyzer, ""algoritmos, new String[] {"
      algoritm"});
```

Lucene tokenizers and filters

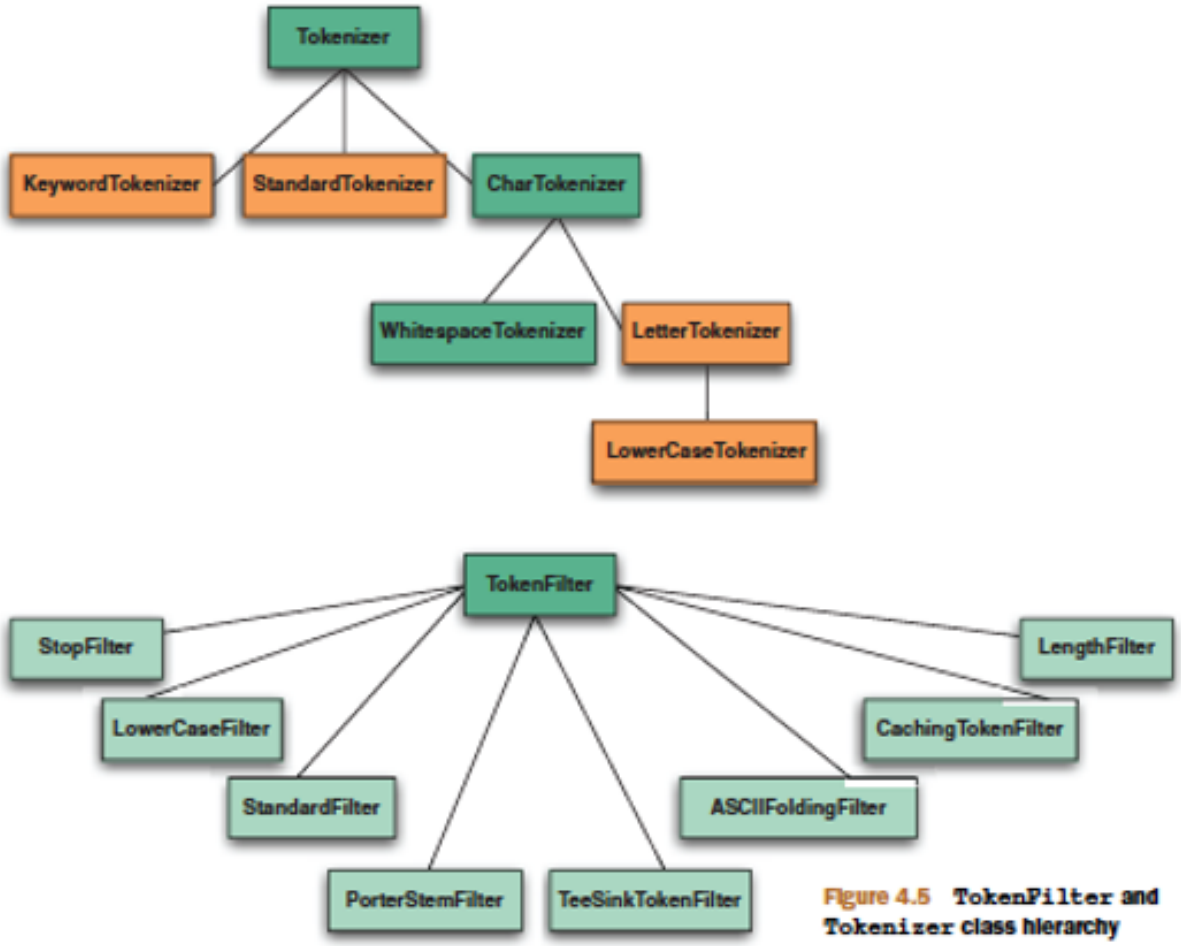
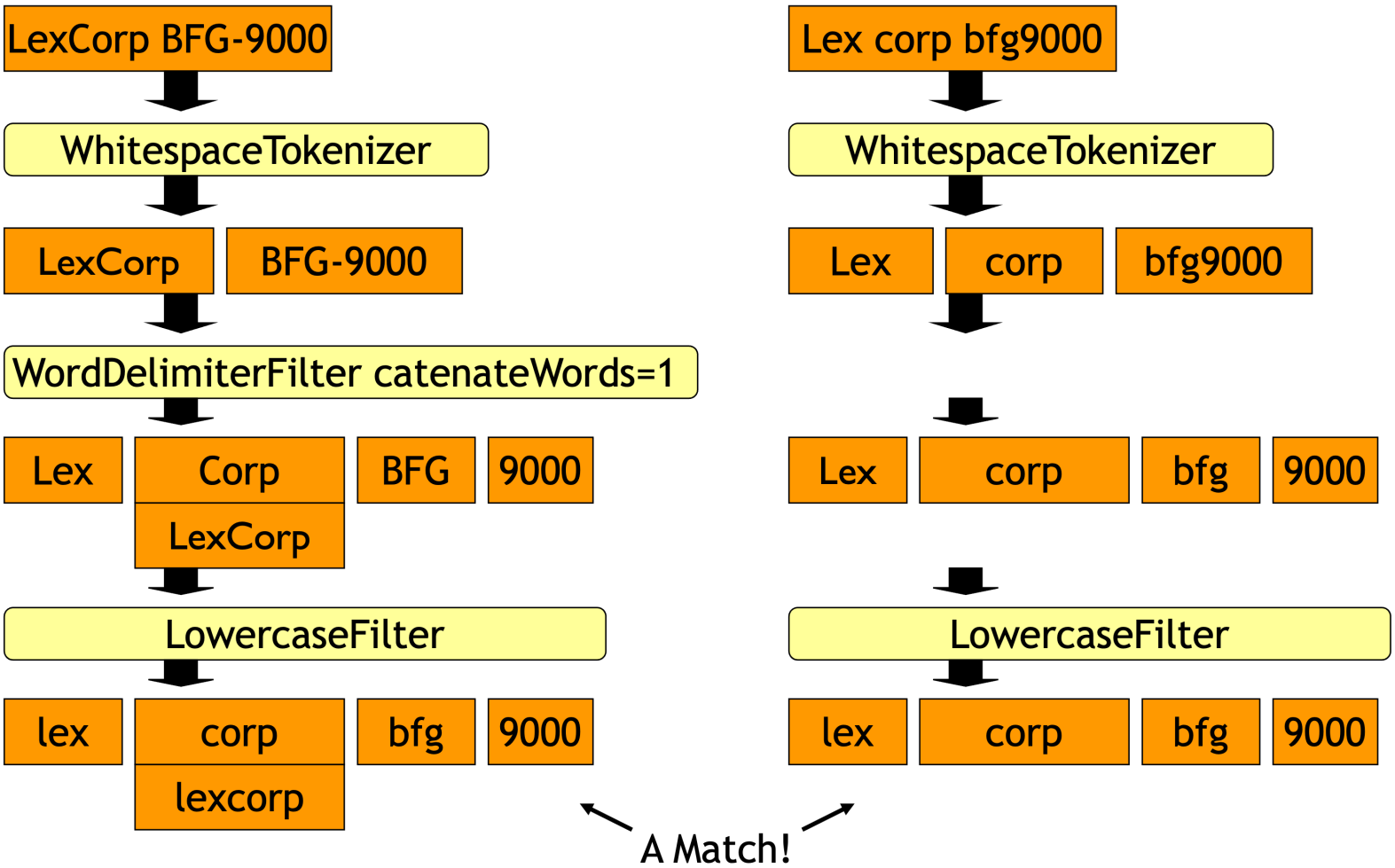


Figure 4.5 `TokenFilter` and `Tokenizer` class hierarchy

An Example


LexCorp BFG-9000
WhitespaceTokenizer
LexCorp
BFG-9000
WordDelimiterFilter catenateWords=1
Lex
Corp
LexCorp
BFG
9000
LowercaseFilter
lex
corp
lexcorp
bfg
9000
Lex corp bfg9000
WhitespaceTokenizer
Lex
corp
bfg9000
Lex
corp
bfg
9000
LowercaseFilter
lex
corp
bfg
9000
A Match!

Search

- 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

Starter code for searching

```
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);
// Search the query

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

- 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

Our project

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

Inspect the data

```
1
2  Jianguos-MBP:1562 jianguolu$ pwd
3  /Users/jianguolu/data/citeseer2/1562
4  Jianguos-MBP:1562 jianguolu$
5
6
7  Jianguos-MBP:1562 jianguolu$ head 10.1.1.2.1562.txt
8  Acting and Deliberating using Golog in Robotic Soccer
9  | A Hybrid Architecture |
10
11 Frank Dylla, Alexander Ferrein, and Gerhard Lakemeyer
12 Department of Computer Science V
13 Aachen University of Technology
14 52056 Aachen, Germany
15
16 fdylla,ferrein,gerhardg@cs.rwth-aachen.de
17 Abstract…
18 .
```

Index all files in a directory

```
1  public class IndexAllFilesInDirectory {
2  static int counter = 0;
3  public static void main(String[] args) throws Exception {
4    String indexPath = "/Users/jianguolu/data/citeseer2_index";
5    String docsPath = "/Users/jianguolu/data/citeseer2";
6    System.out.println("Indexing to directory '" + indexPath + "'...");
7    Directory dir = FSDirectory.open(Paths.get(indexPath));
8    IndexWriterConfig iwc = new IndexWriterConfig(new StandardAnalyzer())
         ;
9    IndexWriter writer = new IndexWriter(dir, iwc);
10   indexDocs(writer, Paths.get(docsPath));
11   writer.close();
12 }
```

Traverse subdirectories

```
1 static void indexDocs(final IndexWriter writer, Path path) throws
      Exception {
2   Files.walkFileTree(path, new SimpleFileVisitor<Path>() {
3     public FileVisitResult visitFile(Path file, BasicFileAttributes
          attrs) throws Exception {
4       indexDoc(writer, file);
5       return FileVisitResult.CONTINUE;
6     }
7   }
8 }
```

- Starts from a given root directory (path).
- Recursively visits every file in the directory and subdirectories.
- For each file, it calls the `indexDoc` method, which is responsible for adding that file to the index.
- The process continues until all files are visited.

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
- Tika is 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

Lucene is just a library

- How to set up a web server for search engine?
- How to handle large volume of incoming queries?
- Queries distributed to different machines
- Data/index also need to be distributed
- Log/monitoring
- ...

Lucene is a

- Low-level library for text indexing and search.
- Provides core capabilities like tokenization, stemming, and scoring.
- Highly customizable with detailed control over indexing and querying.
- Requires manual setup for advanced configurations.

Elasticsearch

- Distributed search and analytics engine built on top of Lucene.
- Provides a higher-level RESTful API for interaction.
- Built-in tools for cluster management, monitoring, and performance analysis.
- Supports distributed deployment, automatic sharding, and replication.
- Designed for horizontal scalability with automatic sharding and replication.
- Supports scaling out by adding more nodes to the cluster.

Indexing in ElasticSearch

```
curl -X POST "localhost:9200/my_index/_doc/1" -H 'Content-Type:
    application/json' -d'
{
  "name": "John Doe",
  "age": 30,
  "join_date": "2021-01-01"
}'
```

Lucene is more powerful

- Lucene offers a fine-grained level of control that sometimes goes beyond what Elasticsearch provides.
- Elasticsearch abstracts many complexities to offer simplicity and scalability.
- e.g:
  - custom scoring and ranking at the index level using non-standard algorithms that require deep integration with Lucene internals.