Applied Databases

Lecture 12
Lucene, JDBC

Sebastian Maneth

University of Edinburgh - February 25th, 2016

Outline

1. New Marking Scheme for Assignement 1
2. Lucene
3. JDBC

Marking Assignment 1

Previous marking scheme was not adequate!

Our mistake.

New Marking Scheme

→ 15 points

Breakdown

1) Large part of the assignment was about
parsing XML and writing CSV files of tables

5 Points XML-to-CSV

-1 Point if you write CSV's but *not the full data*
(e.g., items are missing, or sellers, bids etc.)

New Marking Scheme

→ 15 points

Breakdown

1) Large part of the assignment was about
parsing XML and writing CSV files of tables

5 Points XML-to-CSV

-1 Point if you write CSV's but *not the full data*
(e.g., items are missing, or sellers, bids etc)

2) 1 Point LOAD into mysql

New Marking Scheme

→ 15 points

Breakdown

1) 6 Points XML-to-CSV & LOAD

2) 3.5 Points Queries (0.5 per query)

→ even if your load is incomplete, we
→ check every query by hand, or test it against our sample solution over your incomplete data

New Marking Scheme

→ 15 points

Breakdown

1) 6 Points XML-to-CSV & LOAD

2) 3.5 Points Queries (0.5 per query)

→ even if your load is incomplete, we
→ check every query by hand, or test it against our sample solution over your incomplete data

3) 2 Points Schema (1 Point for keys, 1 Point for NFs)

New Marking Scheme

→ 15 points

Breakdown

1) 6 Points XML-to-CSV & LOAD

2) 3.5 Points Queries (0.5 per query)

→ even if your load is incomplete, we
→ check every query by hand, or test it against our sample solution over your incomplete data

3) 2 Points Schema (1 Point for keys, 1 Point for NFs)

4) 3.5 Points Misc. issues (0.7 per issue)

(a) nulls
(b) duplicates
(c) truncation of description
(d) drop (e.g. exists forgotten)
(e) runLoad (e.g., DOS returns, CSVs not removed, try to remove files that don't exist)

New Marking Scheme

→ 15 points

1)	6 Points	XML-to-CSV & LOAD
2)	3.5 Points	Queries (0.5 per query)
3)	2 Points	Schema (1 Point for keys, 1 Point for NFs)
4)	3.5 Points	Misc. issues (0.7 per issue)

Average now: 80%
Almost everyone > 50%

→ you should receive an email with new feedback & mark this afternoon

→ new marks will be reported to the ITO, should be in the system within the next few days

Back to TFIDF, and Lucene

tf-idf weighting

$$w_{t,d} = \mathrm{tf}_{t,d} \times \log N / \mathrm{df}_t$$

- Best known weighting scheme in information retrieval
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection
- Works surprisingly well!
- Works in many other application domains

Binary → count → weight matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

Each document is now represented by a real-valued vector of tf-idf weights $\in \mathrm{R}^{|V|}$

We then calculate the similarity using cosine similarity with these vectors

Many variations of TFIDF scoring

→ IDF(T) = log(N / DF(T)) [as on previous slide]
gives weight zero, to a term appearing in each document!

→ IDF alternatives: log(1+ N / DF(T)) or 1 + log(N / DF(T))

→ alternatives to TF:
- divide by largest TF of that term (normalization)
- take 1 + ln TF ("log-frequencey weighting")
- SQRT(TF)

Many variations of TFIDF scoring

→ IDF(T) = log(N / DF(T)) [as on previous slide]
gives weight zero, to a term appearing in each document!
→ IDF alternatives: log(1+ N / DF(T)) or 1 + log(N / DF(T))

→ alternatives to TF:
- divide by largest TF of that term (normalization)
- take 1 + ln TF ("log-frequencey weighting")
- SQRT(TF)

Explanations for taking log of N / DF(T) ("damping")

→ Probability that *random document* contains term T:
P(T) = DF(T) / N

→ IDF(T) = – log(**P**(T))

Many variations of TFIDF scoring

→ IDF(T) = log(N / DF(T)) [as on previous slide]
gives weight zero, to a term appearing in each document!

→ IDF alternatives: log(1+ N / DF(T)) or 1 + log(N / DF(T))

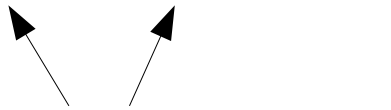
→ alternatives to TF: – divide by largest TF of that term (normalization)
– take 1 + ln TF ("log-frequencey weighting")

Explanations for taking log of N / DF(T) ("damping")

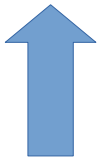
→ Probability that *random document* contains term T:
P(T) = DF(T) / N

→ IDF(T) = – log(**P**(T))

→ IDF(T1 'and' T2) = – log(**P**(T1) * **P**(T2)) = IDF(T1) **+** IDF(T2)



statistically independent



good!

Recall from Information Theory:

Message probabilities p1, p2, p3, …, pN (sum equals 1)

Information of Message k: $I(k) = - \log p_k$

→ see **Robertson's paper** linked on course web page

Recall from Information Theory:

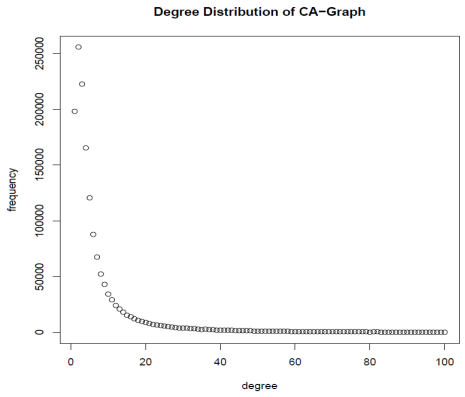
Message probabilities p1, p2, p3, …, pN (sum equals 1)

Information of Message k: *I*(k) = – log pk

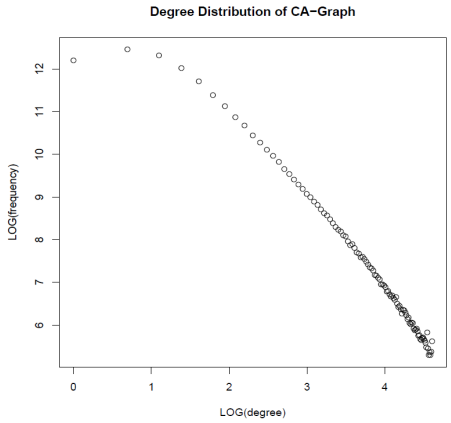
→ see **Robertson's paper** linked on course web page

→ slight relationship to Zipf's law

Mentioned in original article introducing IDF [Spärck Jones, 1972]



gives a line on log/log-scale



Log-frequency weighting

- Want to reduce the effect of multiple occurrences of a term
- A document about “Clinton” will have “Clinton” occuring many times
- Rather than use the frequency, us the log of the frequency

$$w_{t,d} = \begin{cases} 1 + \log \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- $0 \rightarrow 0$, $1 \rightarrow 1$, $2 \rightarrow 1.3$, $10 \rightarrow 2$, $1000 \rightarrow 4$, etc.

tf-idf weighting has many variants

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

→ how does the base of the logs influence scoring / *ranking*?

→ take document length into account
(favour shorter documents)

→ e.g. divide by square root of document length
(done by Lucene, via the “LengthNorm”)

Lucene's Scoring Function

$$score(q,d) = \sum [tf(t_d) \times idf(t) \times boost(t.field_d) \times lengthNorm(t.field_d)] \times coord(q,d) \times qNorm(q)$$

where q is the query, d a document, t a term, and:

1. *tf* is a function of the term frequency within the document (default: $\sqrt{freq}$);
2. *idf*: Inverse document frequency of t within the whole collection (default: $\log(\frac{numDocs}{docFreq+1}) + 1$);
3. *boost* is the boosting factor, if required in the query with the "^ " operator on a given field (if not specified, set to the default field);
4. *lengthNorm*: field normalization according to the number of terms. Default: $\frac{1}{\sqrt{nbTerms}}$
5. *coord*: overlapping rate of terms of the query in the given document. Default: $\frac{overlap}{maxOverlap}$
6. *qNorm*: query normalization according to its length; it corresponds to the sum of square values of terms' weight, the global value is multiplied by each term's weight.

Lucene's Scoring Function

$$score(q,d) = \sum [tf(t_d) \times idf(t) \times boost(t.field_d) \times lengthNorm(t.field_d)] \times coord(q,d) \times qNorm(q)$$

where q is the query, d a document, t a term, and:

1. *tf* is a function of the term frequency within the document (default: $\sqrt{freq}$);
2. *idf*: Inverse document frequency of t within the whole collection (default: $\log(\frac{numDocs}{docFreq+1}) + 1$);

   natural log (base e)
3. *boost* is the boosting factor, if required in the query with the "^ " operator on a given field (if not specified, set to the default field);
4. *lengthNorm*: field normalization according to the number of terms. Default: $\frac{1}{\sqrt{nbTerms}}$
5. *coord*: overlapping rate of terms of the query in the given document. Default: $\frac{overlap}{maxOverlap}$
6. *qNorm*: query normalization according to its length; it corresponds to the sum of square values of terms' weight, the global value is multiplied by each term's weight.

Lucene's Scoring Function

$$score(q,d) = \sum [tf(t_d) \times idf(t) \times boost(t.field_d) \times lengthNorm(t.field_d)] \times coord(q,d) \times qNorm(q)$$

where q is the query, d a document, t a term, and:

1. *tf* is a function of the term frequency within the document (default: $\sqrt{freq}$);
2. *idf*: Inverse document frequency of t within the whole collection (default: $\log(\frac{numDocs}{docFreq+1}) + 1$);
3. *boost* is the boosting factor, if required in the query with the "^ " operator on a given field (if not specified, set to the default field);
4. *lengthNorm*: field normalization according to the number of terms. Default: $\frac{1}{\sqrt{nbTerms}}$

natural log (base e)

very useful:
→ e.g., boost weight of title-field
or of categories-field

2. Lucene

2. Lucene

→ choose appropriate Analyzer for
- casefolding
- stemming (wrt a given language)
- stopping (wrt a given language)

→ insert documents (per “field”) into a collection and generate inverted files

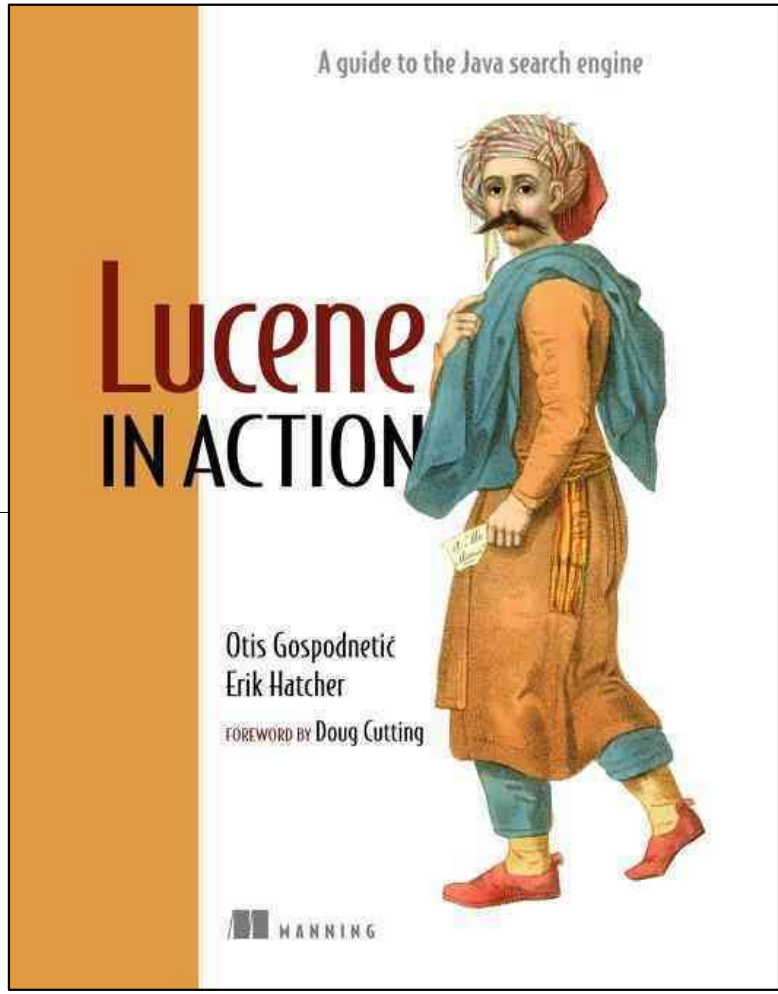
---

→ retrieve top-K ranked documents

→ retrieve score of a document

2. Lucene

→ choose appropriate Analyzer for
  – casefolding
  – stemming (wrt a given language)
  – stopping (wrt a given language)
→ insert documents (per "field") and generate inverted files
→ retrieve top-K docs with scores



Lucene is a huge library

→ we use Version 5.4.0

→ **most books** use older Versions, e.g. Versions 4 or 3

→ the Versions are **not** downward compatible **:-(**

Lucene Indexing

store original text

```
public static void insertDoc(IndexWriter i, String doc_id, String line){
  Document doc = new Document();
  doc.add(new TextField("doc_id", doc_id, Field.Store.YES));
  doc.add(new TextField("line", line,Field.Store.YES));
  try { i.addDocument(doc); } catch (Exception e) { e.printStackTrace(); }
}

public static void rebuildIndexes(String indexPath) {
  try {
    . . .
    IndexWriterConfig config=new IndexWriterConfig(new SimpleAnalyzer());
    IndexWriter i = new IndexWriter(directory, config);
    i.deleteAll();
    insertDoc(i, "1", "The old night keeper keeps the keep in the town");
    insertDoc(i, "2", "In the big old house in the big old gown.");
    . . .
```

doc_id field

line field

Full code for indexing documents to an index on disk (directory indexPath)

path name (from command line)

```
public static void insertDoc(IndexWriter i, String doc_id, String line){
  Document doc = new Document();
  doc.add(new TextField("doc_id", doc_id, Field.Store.YES));
  doc.add(new TextField("line", line,Field.Store.YES));
  try { i.addDocument(doc); } catch (Exception e) { e.printStackTrace(); }
}

public static void rebuildIndexes(String indexPath) {
  try {
    Path path = Paths.get(indexPath);
    System.out.println("Indexing to directory " + indexPath);
    Directory directory = FSDirectory.open(path);
    IndexWriterConfig config = new IndexWriterConfig(new SimpleAnalyzer());
    IndexWriter i = new IndexWriter(directory, config);
    i.deleteAll();
    insertDoc(i, "1", "The old night keeper keeps the keep in the town");
    insertDoc(i, "2", "In the big old house in the big old gown.");
    insertDoc(i, "3", "The house in the town had the big old keep");
    insertDoc(i, "4", "Where the old night keeper never did sleep.");
    insertDoc(i, "5", "The night keeper keeps the keep in the night");
    insertDoc(i, "6", "And keeps in the dark and sleeps in the light.");
    i.close();
    directory.close();
    }
  catch (Exception e) { e.printStackTrace(); }
}
```

2. Lucene

```
public static void rebuildIndexes(String indexPath) {
  try {
    . . .
    IndexWriterConfig config = new IndexWriterConfig(new SimpleAnalyzer());
    IndexWriter i = new IndexWriter(directory, config);
```

SimpleAnalyzer
→ Analyzer that filters LetterTokenizer with LowerCaseFilter

LetterTokenizer
→ divides text at non-letters.
→ tokens are maximal strings of adjacent letters,
as defined by java.lang.Character.isLetter() predicate.

Note: this does a decent job for most European languages, but does a terrible job for some Asian languages, where words are not separated by spaces.

LowerCaseFilter
→ Normalizes token text to lower case.

Keyword Search

```
Private static TopDocs search(String searchText);
  . . .
  IndexReader indexReader =  DirectoryReader.open(directory);
  IndexSearcher indexSearcher = new IndexSearcher(indexReader);
  QueryParser queryParser = new QueryParser(searchField, new SimpleAnalyzer());

  Query query = queryParser.parse(searchText);
  TopDocs topDocs = indexSearcher.search(query,10000);
  System.out.println("Number of Hits: " + topDocs.totalHits);
  for (ScoreDoc scoreDoc:topDocs.scoreDocs) {
    Document doc = indexSearcher.doc(scoreDoc.doc);
    System.out.println("doc_id: " + doc.get("doc_id")
                     + ", score: " + scoreDoc.score
                     + " [" + doc.get("line") +"]");
    }
```

Top-K (K= 10000)

Output
→ doc_id
→ score
→ content (line)

```
1    The old night keeper keeps the keep in the town
2    In the big old house in the big old gown.
3    The house in the town had the big old keep
4    Where the old night keeper never did sleep.
5    The night keeper keeps the keep in the night
6    And keeps in the dark and sleeps in the light.
```

Fig. 1. The Keeper database. It consists of six one-line documents.

```
$ java Searcher "old"

Running search(old, line)
Number of Hits: 4
doc_id: 2, score: 0.5225172 [In the big old house in the big old gown.]
doc_id: 1, score: 0.36947548 [The old night keeper keeps the keep in the town]
doc_id: 3, score: 0.36947548 [The house in the town had the big old keep]
doc_id: 4, score: 0.36947548 [Where the old night keeper never did sleep.]
```

```
1    The old night keeper keeps the keep in the town
2    In the big old house in the big old gown.
3    The house in the town had the big old keep
4    Where the old night keeper never did sleep.
5    The night keeper keeps the keep in the night
6    And keeps in the dark and sleeps in the light.
```

Fig. 1. The Keeper database. It consists of six one-line documents.

```
$ java Searcher "old"

Running search(old, line)
Number of Hits: 4
doc_id: 2, score: 0.5225172 [In the big old house in the big old gown.]
doc_id: 1, score: 0.36947548 [The old night keeper keeps the keep in the town]
doc_id: 3, score: 0.36947548 [The house in the town had the big old keep]
doc_id: 4, score: 0.36947548 [Where the old night keeper never did sleep.]
```

How is this computed?

```
1    The old night keeper keeps the keep in the town
2    In the big old house in the big old gown.
3    The house in the town had the big old keep
4    Where the old night keeper never did sleep.
5    The night keeper keeps the keep in the night
6    And keeps in the dark and sleeps in the light.
```

Fig. 1. The Keeper database. It consists of six one-line documents.

```
$ java Searcher "old"

Running search(old, line)
Number of Hits: 4
doc_id: 2, score: 0.5225172 [In the big old house in the big old gown.]
doc_id: 1, score: 0.36947548 [The old night keeper keeps the keep in the town]
doc_id: 3, score: 0.36947548 [The house in the town had the big old keep]
doc_id: 4, score: 0.36947548 [Where the old night keeper never did sleep.]
```

How is this computed?

→ in the for-loop, add:

```
Explanation ex = indexSearcher.explain(query, rank);
System.out.println("Explanation: " + ex.toString());
```

Integer, gives the k-th ranked doc
→ for equal rank: uses *indexing order*!

```
1    The old night keeper keeps the keep in the town
2    In the big old house in the big old gown.
3    The house in the town had the big old keep
4    Where the old night keeper never did sleep.
5    The night keeper keeps the keep in the night
6    And keeps in the dark and sleeps in the light.
```

Fig. 1. The Keeper database. It consists of six one-line documents.

```
$ java Searcher "old"

Running search(old, line)
Number of Hits: 4
doc_id: 2, score: 0.5225172 [In the big old house in the big old gown.]
doc_id: 1, score: 0.36947548 [The old night keeper keeps the keep in the town]
doc_id: 3, score: 0.36947548 [The house in the town had the big old keep]
doc_id: 4, score: 0.36947548 [Where the old night keeper never did sleep.]

Explanation: 0.5225172 = weight(line:old in 1) [DefaultSimilarity], result of:
  0.5225172 = fieldWeight in 1, product of:
    1.4142135 = tf(freq=2.0), with freq of:
      2.0 = termFreq=2.0
    1.1823215 = idf(docFreq=4, maxDocs=6)
    0.3125 = fieldNorm(doc=1)
```

https://lucene.apache.org/core/5_0_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html

$$tf(t \text{ in } d) = frequency^{1/2}$$

```
$ java Searcher "old"

Running search(old, line)
Number of Hits: 4
doc_id: 2, score: 0.5225172 [In the big old house in the big old gown.]
doc_id: 1, score: 0.36947548 [The old night keeper keeps the keep in the town]
doc_id: 3, score: 0.36947548 [The house in the town had the big old keep]
doc_id: 4, score: 0.36947548 [Where the old night keeper never did sleep.]

Explanation: 0.5225172 = weight(line:old in 1) [DefaultSimilarity], result of:
  0.5225172 = fieldWeight in 1, product of:
    1.4142135 = tf(freq=2.0), with freq of:
      2.0 = termFreq=2.0
    1.1823215 = idf(docFreq=4, maxDocs=6)
    0.3125 = fieldNorm(doc=1)
```

https://lucene.apache.org/core/5_0_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html

$$tf(t \text{ in } d) = \text{frequency}^{1/2}$$

$$idf(t) = 1 + \log\left(\frac{\text{numDocs}}{\text{docFreq}+1}\right)$$

```
$ java Searcher "old"

Running search(old, line)
Number of Hits: 4
doc_id: 2, score: 0.5225172 [In the big old house in the big old gown.]
doc_id: 1, score: 0.36947548 [The old night keeper keeps the keep in the town]
doc_id: 3, score: 0.36947548 [The house in the town had the big old keep]
doc_id: 4, score: 0.36947548 [Where the old night keeper never did sleep.]

Explanation: 0.5225172 = weight(line:old in 1) [DefaultSimilarity], result of:
  0.5225172 = fieldWeight in 1, product of:
    1.4142135 = tf(freq=2.0), with freq of:
      2.0 = termFreq=2.0
    1.1823215 = idf(docFreq=4, maxDocs=6)
    0.3125 = fieldNorm(doc=1)
```

1 + ln(6/5) = 1 + ln(1.2) = **1.1823215**567 (natural logarithm!)

https://lucene.apache.org/core/5_0_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html

$$tf(t \text{ in } d) = frequency^{1/2}$$

$$idf(t) = 1 + \log\left(\frac{numDocs}{docFreq+1}\right)$$

fieldNorm = 1/SQRT(#terms in doc)

(because we have't specified any "field boost" at indexing time)

```
$ java Searcher "old"

Running search(old, line)
Number of Hits: 4
doc_id: 2, score: 0.5225172 [In the big old house in the big old gown.]
doc_id: 1, score: 0.36947548 [The old night keeper keeps the keep in the town]
doc_id: 3, score: 0.36947548 [The house in the town had the big old keep]
doc_id: 4, score: 0.36947548 [Where the old night keeper never did sleep.]

Explanation: 0.5225172 = weight(line:old in 1) [DefaultSimilarity], result of:
  0.5225172 = fieldWeight in 1, product of:
    1.4142135 = tf(freq=2.0), with freq of:
      2.0 = termFreq=2.0
    1.1823215 = idf(docFreq=4, maxDocs=6)
    0.3125 = fieldNorm(doc=1)
```

1 / SQRT(10) = **0.31**6227

→ why such crude rounding?
→ uses only 1 BYTE
- three-bit mantissa
- five-bit exponent
- zero-exponent point at 15

e.g. decode(encode(0.89)) = 0.75

fieldNorm = 1/SQRT(#terms in doc)

(because we have't specified any "field boost" at indexing time)

```
$ java Searcher "old"

Running search(old, line)
Number of Hits: 4
doc_id: 2, score: 0.5225172 [In the big old house in the big old gown.]
doc_id: 1, score: 0.36947548 [The old night keeper keeps the keep in the town]
doc_id: 3, score: 0.36947548 [The house in the town had the big old keep]
doc_id: 4, score: 0.36947548 [Where the old night keeper never did sleep.]

Explanation: 0.5225172 = weight(line:old in 1) [DefaultSimilarity], result of:
  0.5225172 = fieldWeight in 1, product of:
    1.4142135 = tf(freq=2.0), with freq of:
      2.0 = termFreq=2.0
    1.1823215 = idf(docFreq=4, maxDocs=6)
    0.3125 = fieldNorm(doc=1)
```

1 / SQRT(10) = 0.316227

(1/2)	(1/4)	(1/8)	(1/16)
	1	0	1

= 0.3125

```
1    The old night keeper keeps the keep in the town
2    In the big old house in the big old gown.
3    The house in the town had the big old keep
4    Where the old night keeper never did sleep.
5    The night keeper keeps the keep in the night
6    And keeps in the dark and sleeps in the light.
```

Fig. 1. The Keeper database. It consists of six one-line documents.

```
$ java Searcher "big old house"

Running search(big old house, line)
Number of Hits: 4
doc_id: 2, score: 1.0412337 [In the big old house in the big old gown.]
doc_id: 3, score: 0.83452004 [The house in the town had the big old keep]
doc_id: 1, score: 0.054527204 [The old night keeper keeps the keep in the town]
doc_id: 4, score: 0.054527204 [Where the old night keeper never did sleep.]
```

```
1    The old night keeper keeps the keep in the town
2    In the big old house in the big old gown.
3    The house in the town had the big old keep
4    Where the old night keeper never did sleep.
5    The night keeper keeps the keep in the night
6    And keeps in the dark and sleeps in the light.
```

Fig. 1. The Keeper database. It consists of six one-line documents.

```
$ java Searcher "the"

Running search(the, line)
Number of Hits: 6
doc_id: 1, score: 0.4578294 [The old night keeper keeps the keep in the town]
doc_id: 3, score: 0.4578294 [The house in the town had the big old keep]
doc_id: 5, score: 0.4578294 [The night keeper keeps the keep in the night]
doc_id: 2, score: 0.37381613 [In the big old house in the big old gown.]
doc_id: 6, score: 0.37381613 [And keeps in the dark and sleeps in the light.]
doc_id: 4, score: 0.2643279 [Where the old night keeper never did sleep.]
```

```
1    The old night keeper keeps the keep in the town
2    In the big old house in the big old gown.
3    The house in the town had the big old keep
4    Where the old night keeper never did sleep.
5    The night keeper keeps the keep in the night
6    And keeps in the dark and sleeps in the light.
```

Fig. 1. The Keeper database. It consists of six one-line documents.

```
$ java Searcher "the"

Running search(the, line)
Number of Hits: 6
doc_id: 1, score: 0.4578294 [The old night keeper keeps the keep in the town]
doc_id: 3, score: 0.4578294 [The house in the town had the big old keep]
doc_id: 5, score: 0.4578294 [The night keeper keeps the keep in the night]
doc_id: 2, score: 0.37381613 [In the big old house in the big old gown.]
doc_id: 6, score: 0.37381613 [And keeps in the dark and sleeps in the light.]
doc_id: 4, score: 0.2643279 [Where the old night keeper never did sleep.]
```

length 10 vs 9

```
1    The old night keeper keeps the keep in the town
2    In the big old house in the big old gown.
3    The house in the town had the big old keep
4    Where the old night keeper never did sleep.
5    The night keeper keeps the keep in the night
6    And keeps in the dark and sleeps in the light.
```

```
7    The house is the house.
8    The house.
```

Even shorter – encode(0.707)

Shorter – encode(0.447)

```
$ java Searcher "the"

Running search(the, line)
Number of Hits: 8
doc_id: 8, score: 0.55138564 [The house.]
doc_id: 7, score: 0.5458439 [The house is the house.]
doc_id: 1, score: 0.47751394 [The old night keeper keeps the keep in the town]
doc_id: 3, score: 0.47751394 [The house in the town had the big old keep]
doc_id: 5, score: 0.47751394 [The night keeper keeps the keep in the night]
doc_id: 2, score: 0.38988853 [In the big old house in the big old gown.]
doc_id: 6, score: 0.38988853 [And keeps in the dark and sleeps in the light.]
doc_id: 4, score: 0.27569282 [Where the old night keeper never did sleep.]
```

1	The old night keeper keeps the keep in the town
2	In the big old house in the big old gown.
3	The house in the town had the big old keep
4	Where the old night keeper never did sleep.
5	The night keeper keeps the keep in the night
6	And keeps in the dark and sleeps in the light.

7	**The house is the house.**
8	**The house.**
9	**the-the_the__the.**
10	**the-the___the.**
11	**the-thethe__the.**
12	**The.**
13	**The.**
14	**The a b c.**
15	**The a b.**
16	**The a.**

```
$ java Searcher "the"
doc_id: 9, score: 0.9393754 [the-the_the__the.]
doc_id: 12, score: 0.9393754 [The.]
doc_id: 13, score: 0.83029836 [The the.]
doc_id: 10, score: 0.81352293 [the-the___the.]
doc_id: 11, score: 0.6642387 [the-thethe__the.]
doc_id: 8, score: 0.5871096 [The house.]
doc_id: 16, score: 0.5871096 [The a.]
doc_id: 7, score: 0.5812088 [The house is the house.]
doc_id: 1, score: 0.5084518 [The old night keeper keeps the keep in the town]
doc_id: 3, score: 0.5084518 [The house in the town had the big old keep]
doc_id: 5, score: 0.5084518 [The night keeper keeps the keep in the night]
doc_id: 14, score: 0.4696877 [The a b c.]
doc_id: 15, score: 0.4696877 [The a b.]
doc_id: 2, score: 0.41514918 [In the big old house in the big old gown.]
doc_id: 6, score: 0.41514918 [And keeps in the dark and sleeps in the light.]
doc_id: 4, score: 0.2935548 [Where the old night keeper never did sleep.]
```

term frequency = 4

term frequency = 3

term frequency = 2

same encoded lengthNorm

```
1    The old night keeper keeps the keep in the town
2    In the big old house in the big old gown.
3    The house in the town had the big old keep
4    Where the old night keeper never did sleep.
5    The night keeper keeps the keep in the night
6    And keeps in the dark and sleeps in the light.
```

SimpleAnalyzer
→ filters LetterTokenizer with LowerCaseFilter

StandardAnalyzer
→ filters StandardTokenizer with StandardFilter, LowerCaseFilter and StopFilter, using a list of English stop words.

StandardTokenizer
→ grammar-based tokenizer (done in JFlex), implements the Word Break rules from the Unicode Text Segmentation algorithm, as specified in Unicode Standard Annex #29.

Standard Filter
→ normalizes tokens extracted with StandardTokenizer.

StopFilter
→ removes stop words from a token stream.

StandardAnalyzer – Stop Words

```
1    The old night keeper keeps the keep in the town
2    In the big old house in the big old gown.
3    The house in the town had the big old keep
4    Where the old night keeper never did sleep.
5    The night keeper keeps the keep in the night
6    And keeps in the dark and sleeps in the light.
```

```
$ java Searcher "the"

Running search(the, line)
Number of Hits: 0
```

StandardAnalyzer – Stop Words

```
1    The old night keeper keeps the keep in the town
2    In the big old house in the big old gown.
3    The house in the town had the big old keep
4    Where the old night keeper never did sleep.
5    The night keeper keeps the keep in the night
6    And keeps in the dark and sleeps in the light.
```

```
$ java Searcher "the"

Running search(the, line)
Number of Hits: 0

$ java Searcher "and"

Running search(and, line)
Number of Hits: 0
```

StandardAnalyzer – Stop Words

```
1    The old night keeper keeps the keep in the town
2    In the big old house in the big old gown.
3    The house in the town had the big old keep
4    Where the old night keeper never did sleep.
5    The night keeper keeps the keep in the night
6    And keeps in the dark and sleeps in the light.
```

```
$ java Searcher "the"

Running search(the, line)
Number of Hits: 0

$ java Searcher "and"

Running search(and, line)
Number of Hits: 0

$ java Searcher "in"

Running search(in, line)
Number of Hits: 0
```

StandardAnalyzer – Search

1	The old night keeper keeps the keep in the town
2	In the big old house in the big old gown.
3	The house in the town had the big old keep
4	Where the old night keeper never did sleep.
5	The night keeper keeps the keep in the night
6	And keeps in the dark and sleeps in the light.

```
$ java Searcher "keeper"

Running search(keeper, line)
Number of Hits: 3
doc_id: 5, score: 0.614891 [The night keeper keeps the keep in the night]
doc_id: 1, score: 0.5270494 [The old night keeper keeps the keep in the town]
doc_id: 4, score: 0.5270494 [Where the old night keeper never did sleep.]
```

StandardAnalyzer – Stemming?

```
1    The old night keeper keeps the keep in the town
2    In the big old house in the big old gown.
3    The house in the town had the big old keep
4    Where the old night keeper never did sleep.
5    The night keeper keeps the keep in the night
6    And keeps in the dark and sleeps in the light.
```

```
$ java Searcher "keeping"

Running search(keeping, line)
Number of Hits: 0
```

→ stemming?

EnglishAnalyzer

```
1    The old night keeper keeps the keep in the town
2    In the big old house in the big old gown.
3    The house in the town had the big old keep
4    Where the old night keeper never did sleep.
5    The night keeper keeps the keep in the night
6    And keeps in the dark and sleeps in the light.
```

```
$ java Searcher "keeping"

Running search(keeping, line)
Number of Hits: 3
doc_id: 5, score: 0.614891 [The night keeper keeps the keep in the night]
doc_id: 1, score: 0.5270494 [The old night keeper keeps the keep in the town]
doc_id: 3, score: 0.5270494 [The house in the town had the big old keep]
```

Stemming

→ EnglishAnalyzer (in the Query part)

JDBC

→ run SQL queries from Java

→ there will be a file DbManager.java that opens a connection to MySQL (at port 3306) and database "ad"

```
public class DbManager {
    static private String databaseURL = "jdbc:mysql://localhost:3306/";
    static private String dbname = "ad";
    static private String username = "ad";
    static private String password = "ad";
    public static Connection getConnection(boolean readOnly)
    throws SQLException {
    Connection conn = DriverManager.getConnection(
              databaseURL + dbname, username, password);
    conn.setReadOnly(readOnly);

    return conn;
}
```

JDBC

→ run SQL queries from Java

→ there will be a file DbManager.java that opens a connection to MySQL (at port 3306) to the database "ad"

```
public static void runQuery(String indexPath) {
        Connection conn = null;
        Statement stmt = null;
        try {
        conn = DbManager.getConnection(true);
        stmt = conn.createStatement();
        String sql = "SELECT count(*) as count from item;";
        ResultSet rs = stmt.executeQuery(sql);
        while(rs.next()){
            String c = rs.getString("count");
            System.out.println("count: " + c);
            rs.close();
            conn.close();
        }
        } catch (SQLException ex) {
        System.out.println(ex);
        }
    }
```

```
$ java -cp … runQuery
count: 19532
```

END
Lecture 12