Information Access with Apache Lucene

Metodi per il Ritrovamento dell'Informazione

Laurea Triennale in Informatica

Università degli Studi di Bari Aldo Moro

Marco Polignano

marco.polignano@uniba.it

Code Repository & Requirements

Code repository

https://github.com/marcopoli/MRI_2023_24

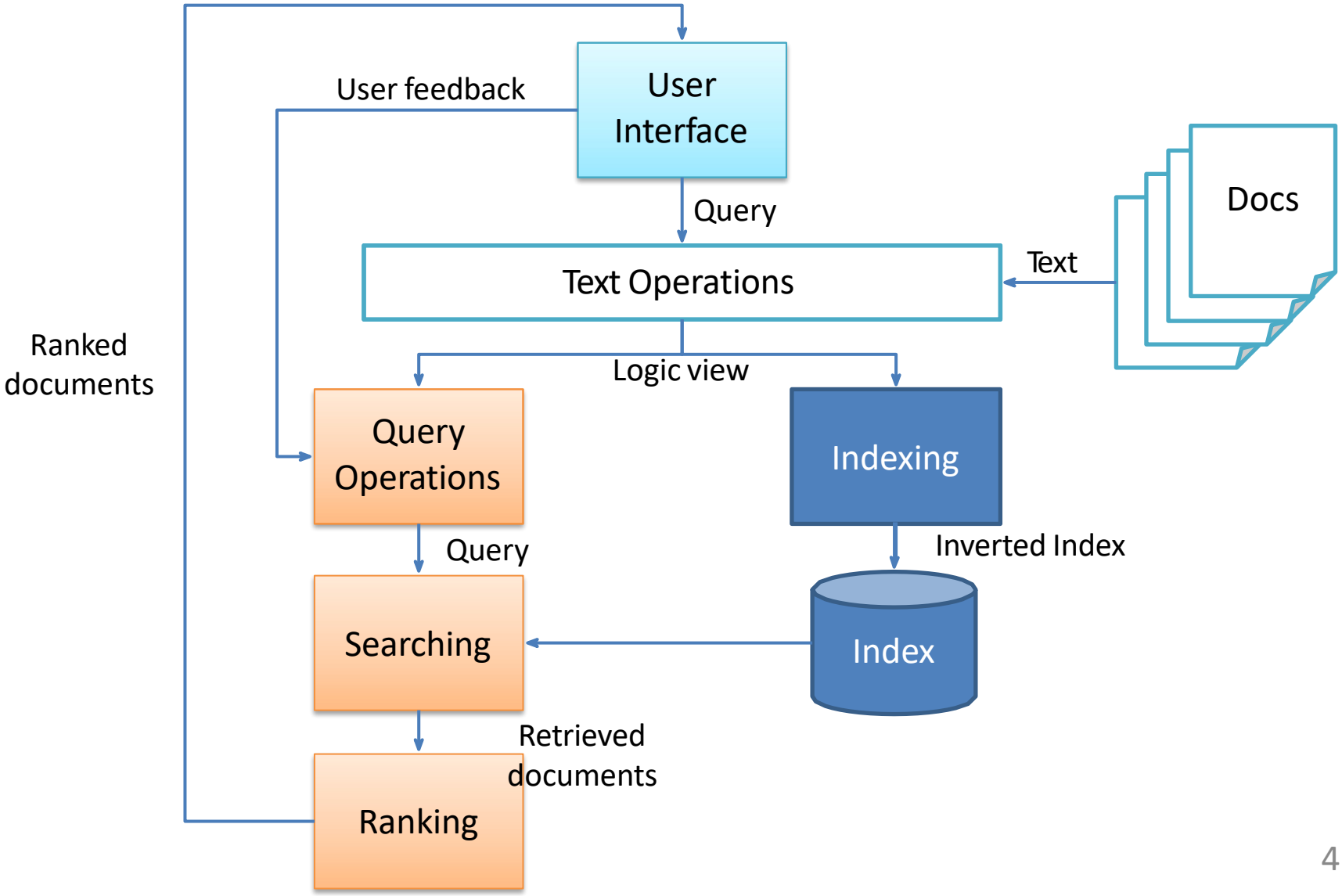
Requirements

- Java SDK 1.8+ https://www.java.com/en/download/
- IDE: NetBeans, IntelliJ, Eclipse, …
- **Maven:** https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

Recap

SEARCH ENGINE

Information Retrieval Process



Information Retrieval Model

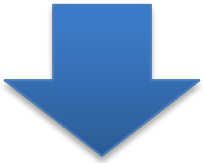
<D, Q, F, $R(q_i, d_j)$>

- D: document representation
- Q: query representation
- F: query/document representation function
- $R(q_i, d_j)$: ranking function

Bag-of-words representation

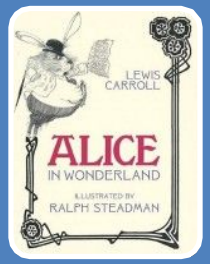
Document/query as unordered collection of words

John likes to watch movies. Mary likes too. John also likes to watch football games.

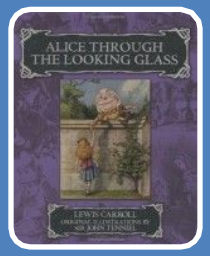


{"John": 1, "likes": 2, "to": 3, "watch": 4, "movies": 5, "also": 6, "football": 7, "games": 8, "Mary": 9, "too": 10}

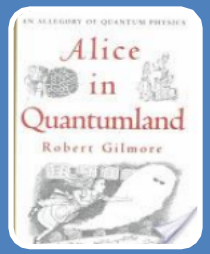
Term-Document matrix



- ‘It’s a friend of mine — a Cheshire Cat,’ said Alice: ‘allow me to introduce it.’
- ‘It’s the oldest rule in the book,’ said the King. ‘Then it ought to be Number One,’ said Alice.



- Alice watched the White King as he slowly struggled up from bar to bar, till at last she said, ‘Why, you’ll be hours and hours getting to the table, at that rate.
- Alice looked round eagerly, and found that it was the Red Queen. ‘She’s grown a good deal!’ was her first remark.



- In the pool of light was a billiards table, with two figures moving around it. Alice walked toward them, and as she approached they turned to look at her.
- Alice lay back, and closed her eyes. There was the Red Queen again, with that incessant grin. Or was it the Cheshire cat's grin?

Term-Document matrix

	D1	D2	D3
Cheshire Cat	1	0	1
Alice	2	2	2
book	1	0	0
King	1	1	0
table	0	1	1
Queen	0	1	1
grin	0	0	2

Term-Document matrix

	D1	D2	D3
Cheshire Cat	1	0	1
Alice	2	2	2
book	1	0	0
King	1	1	0
table	0	1	1
Queen	0	1	1
grin	0	0	2

Query:

Alice AND Queen

0
1
0
0
0
1
0

Inverted Index

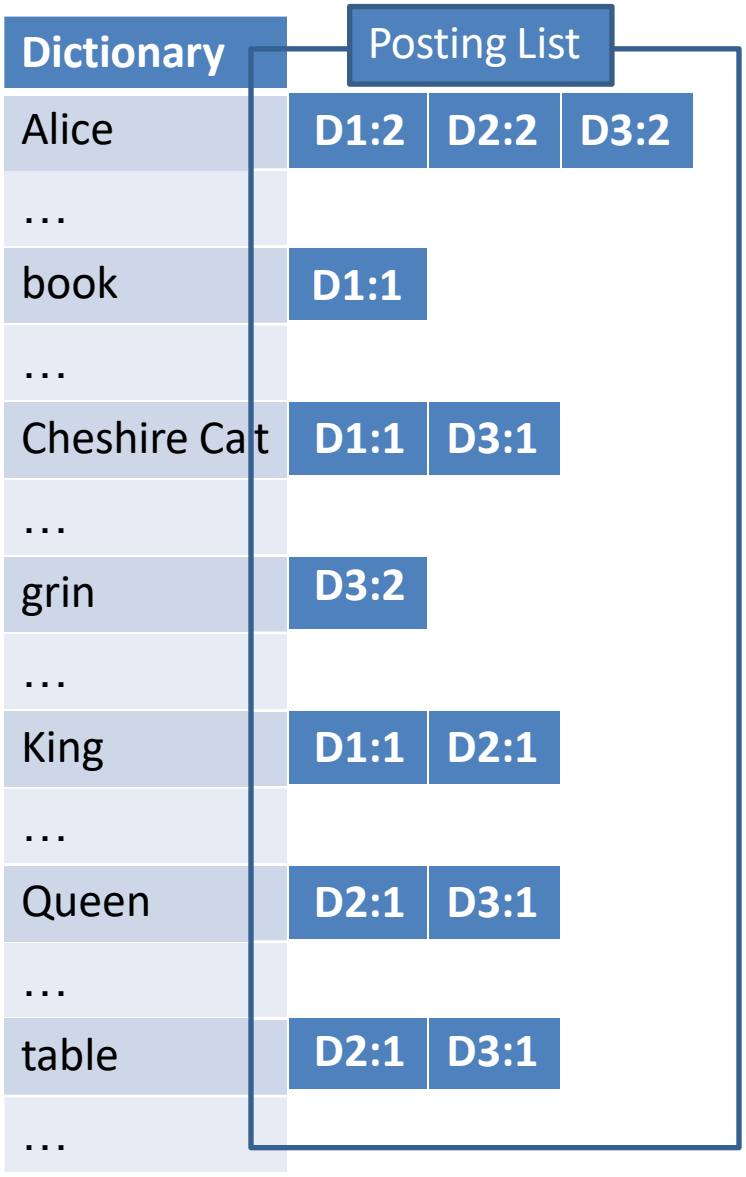
Index

Page numbers in **bold face** refer to key term definitions
Page numbers in *italics* refer to images or diagrams
Page numbers followed by a "t" indicate a table

Inverted Index

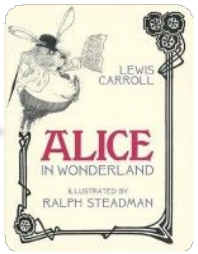
Dictionary	
Alice	D1:2 D2:2 D3:2
…	
book	D1:1
…	
Cheshire Cat	D1:1 D3:1
…	
grin	D3:2
…	
King	D1:1 D2:1
…	
Queen	D2:1 D3:1
…	
table	D2:1 D3:1
…	

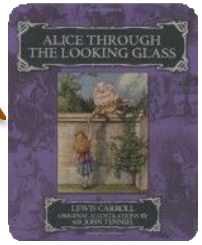
Inverted Index

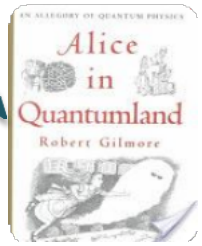


Inverted Index

Dictionary			
Alice	D1:2	D2:2	D3:2
…			
book	D1:1		
…			
Cheshire Cat	D1:1	D3:1	
…			
grin	D3:2		
…			
King	D1:1	D2:1	
…			
Queen	D2:1	D3:1	
…			
table	D2:1	D3:1	
…			

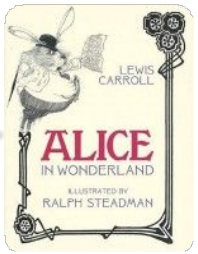


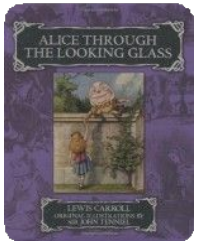


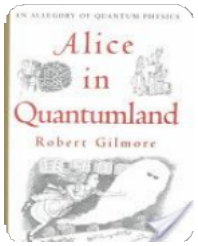


Inverted Index

Dictionary			
Alice	D1:2	D2:2	D3:2
…			
book	D1:1		
…			
Cheshire Cat	D1:1	D3:1	
…			
grin	D3:2		
…			
King	D1:1	D2:1	
…			
Queen	D2:1	D3:1	
…			
table	D2:1	D3:1	
…			

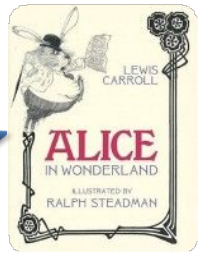


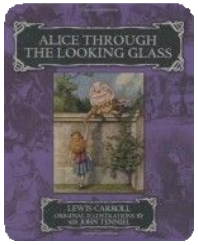


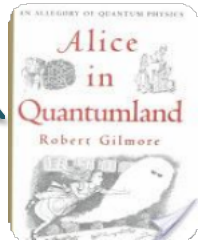


Inverted Index

Dictionary	
Alice	D1:2 D2:2 D3:2
…	
book	D1:1
…	
Cheshire Cat	D1:1 D3:1
…	
grin	D3:2
…	
King	D1:1 D2:1
…	
Queen	D2:1 D3:1
…	
table	D2:1 D3:1
…	

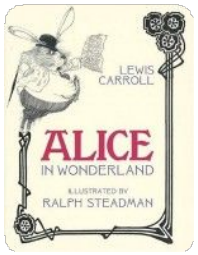


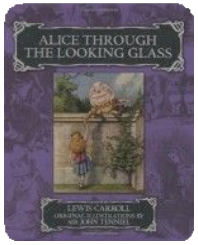


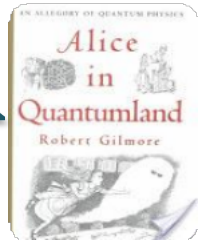


Inverted Index

Dictionary			
Alice	D1:2	D2:2	D3:2
…			
book	D1:1		
…			
Cheshire Cat	D1:1	D3:1	
…			
grin	D3:2		
…			
King	D1:1	D2:1	
…			
Queen	D2:1	D3:1	
…			
table	D2:1	D3:1	
…			

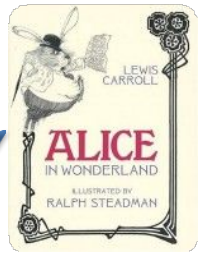


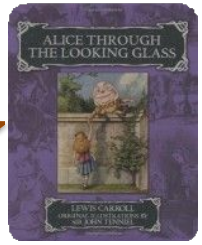


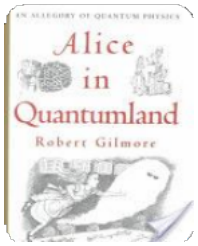


Inverted Index

Dictionary			
Alice	D1:2	D2:2	D3:2
…			
book	D1:1		
…			
Cheshire Cat	D1:1	D3:1	
…			
grin	D3:2		
…			
King	D1:1	D2:1	
…			
Queen	D2:1	D3:1	
…			
table	D2:1	D3:1	
…			

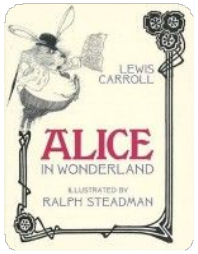


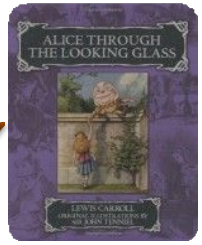


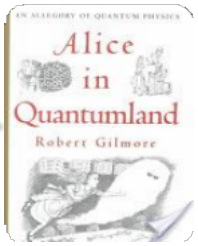


Inverted Index

Dictionary	
Alice	D1:2 D2:2 D3:2
…	
book	D1:1
…	
Cheshire Cat	D1:1 D3:1
…	
grin	D3:2
…	
King	D1:1 D2:1
…	
Queen	D2:1 D3:1
…	
table	D2:1 D3:1
…	

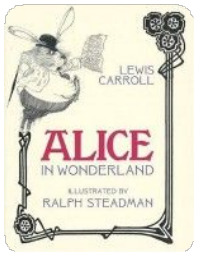


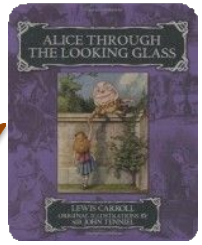


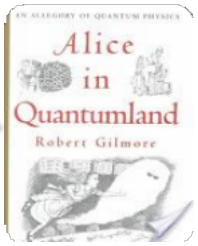


Inverted Index

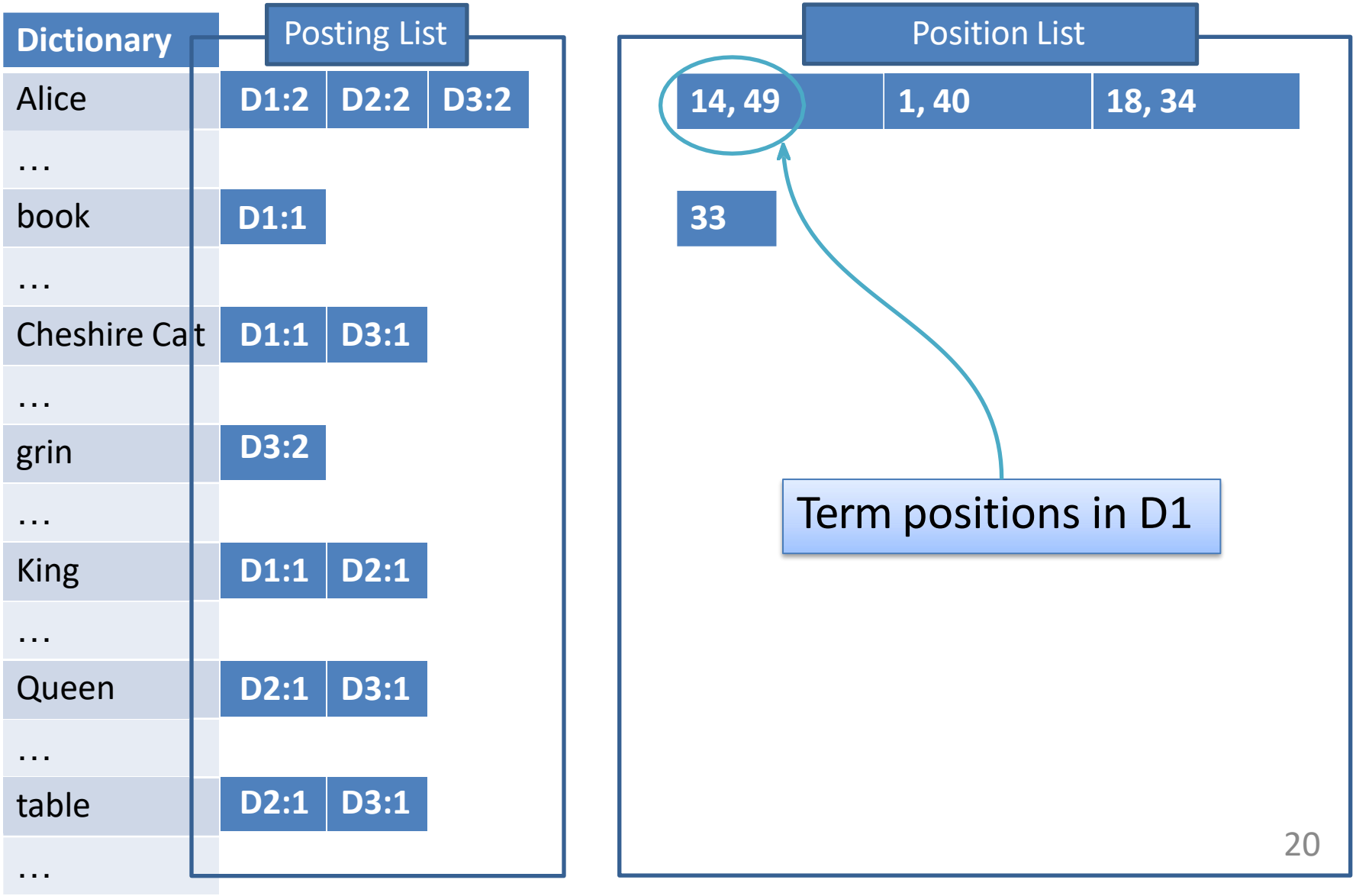
Dictionary	
Alice	D1:2 D2:2 D3:2
…	
book	D1:1
…	
Cheshire Cat	D1:1 D3:1
…	
grin	D3:2
…	
King	D1:1 D2:1
…	
Queen	D2:1 D3:1
…	
table	D2:1 D3:1
…	



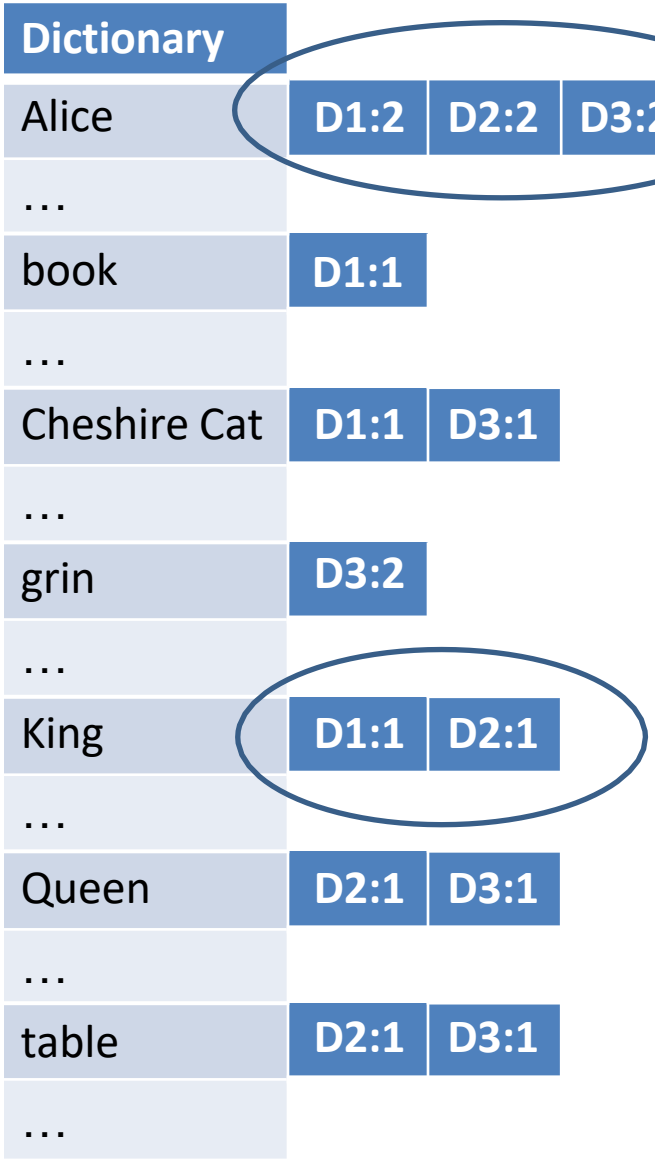




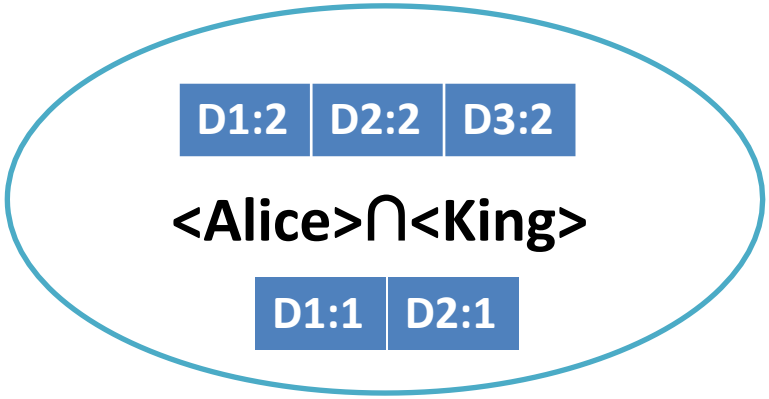
Inverted Index



Inverted Index: query processing



AND (intersection ∩)

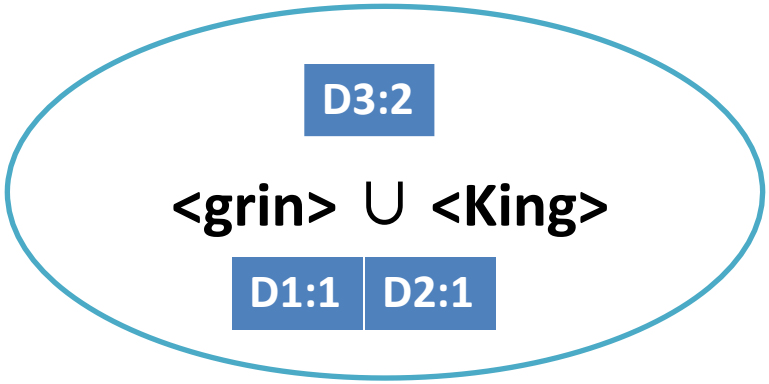


Alice AND King -> (D1, D2)

Inverted Index: query processing

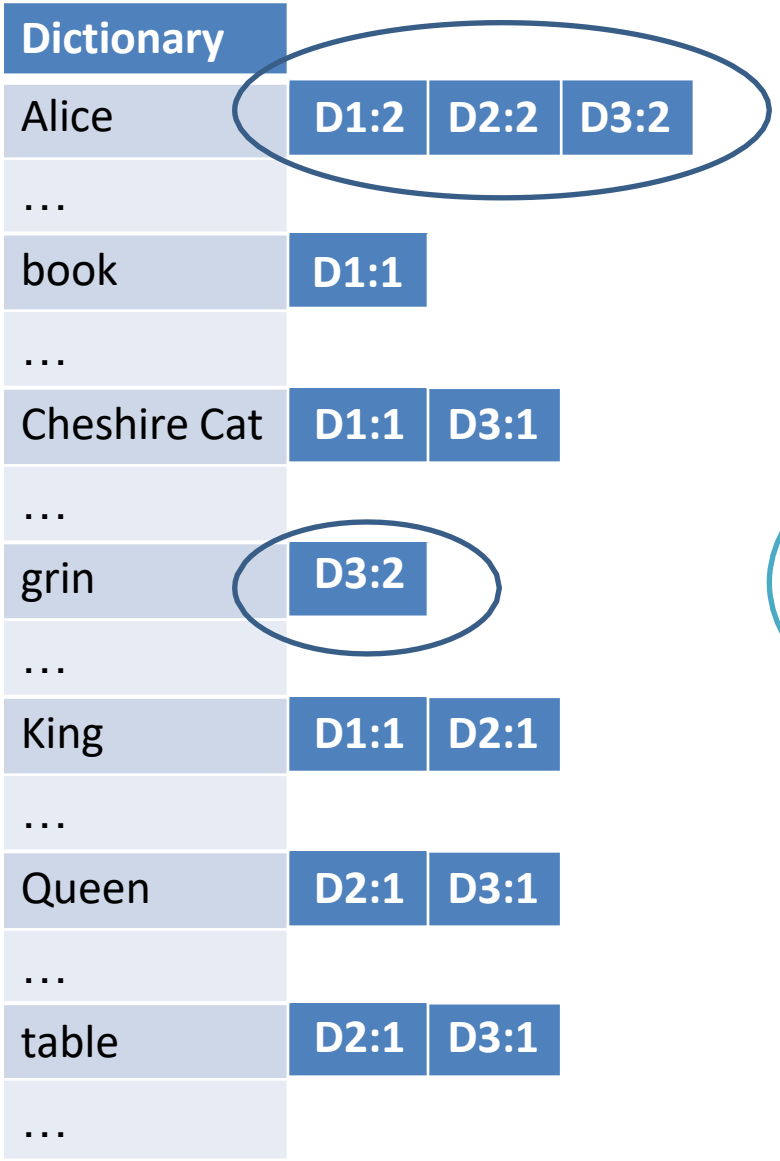
Dictionary	
Alice	D1:2 D2:2 D3:2
…	
book	D1:1
…	
Cheshire Cat	D1:1 D3:1
…	
grin	D3:2
…	
King	D1:1 D2:1
…	
Queen	D2:1 D3:1
…	
table	D2:1 D3:1
…	

OR (union ∪)

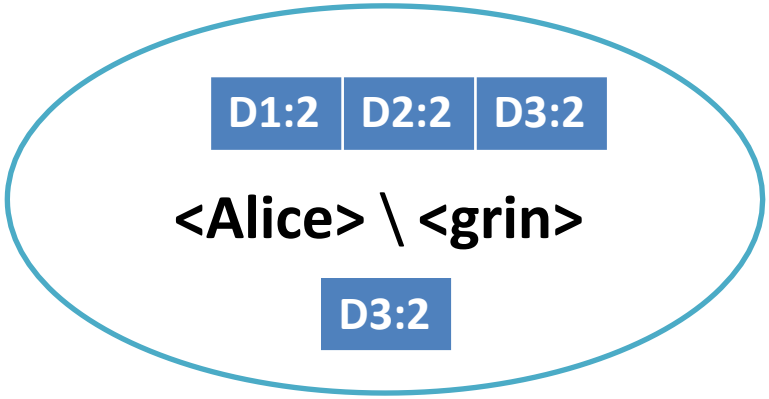


Alice OR King -> (D1, D2, D3)

Inverted Index: query processing



NOT (complement \)



Alice NOT grin -> (D1, D2)

Term-weight

- Measures the term relevance in a document
  - component value in the document representation
  - TF*IDF
    - TF (term frequency): term occurrences in the document
    - IDF (inverse document frequency): inverse to the number of documents in which the term occurs

$$tf * idf(t,d) = tf(t,d) * \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

Term-weight

- Measures the term relevance in a document
  - component value in the document representation
  - TF*IDF
    - TF (term frequency): term occurrences in the document
    - IDF (inverse document frequency): inverse to the number of documents in which the term occurs

$$tf * idf(t,d) = tf(t,d) * \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

Number of documents in the collection

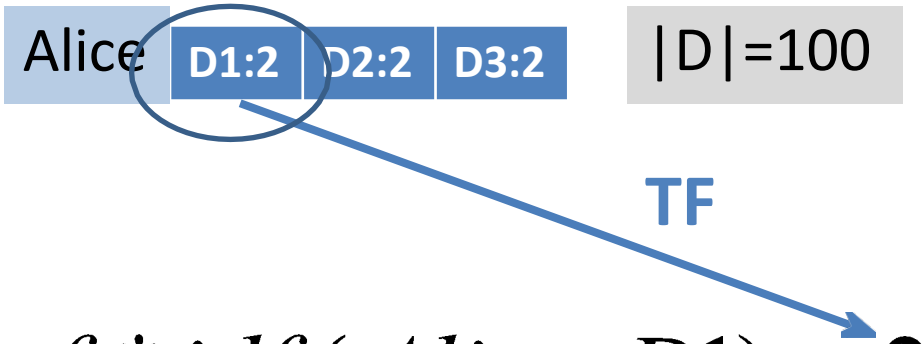
idf

TF*IDF insights

- **increases proportionally** to the term frequency in the document
- **decreases** to the number of documents in which the term belongs
  - common words are generally more frequent in the collection
- **IDF** depends on the collection, **TF** on the document

Inverted index/TF*IDF

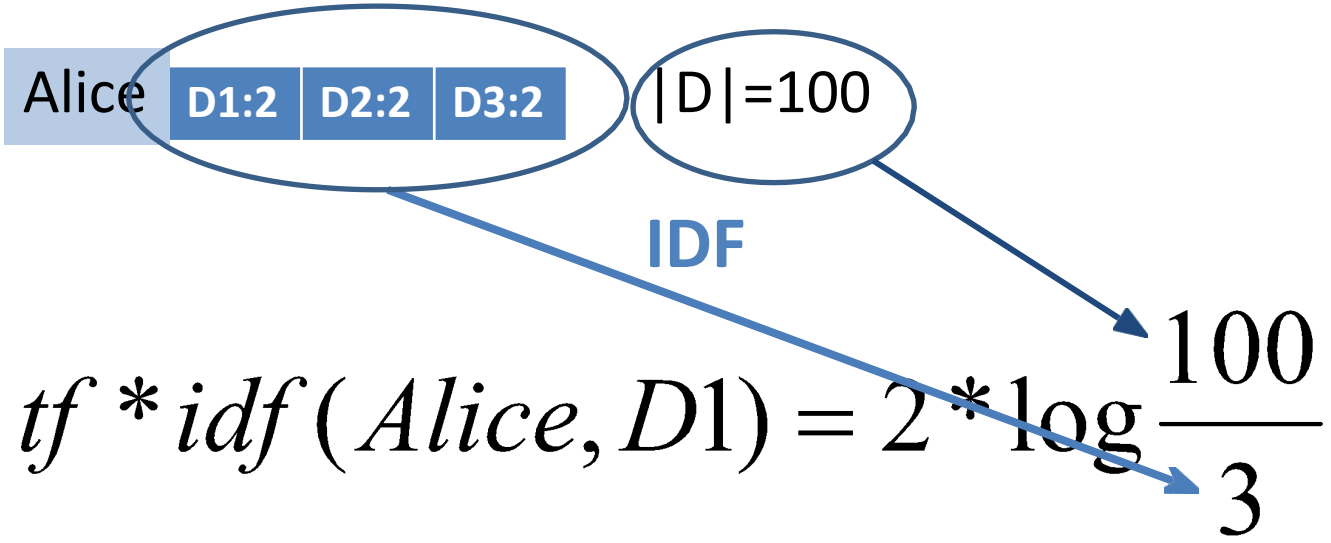
- TF: computed by term occurrences in the posting list
- IDF: computed by the posting list cardinality



$$tf * idf(Alice, D1) = 2 * \log \frac{100}{3}$$

Inverted index/TF*IDF

- TF: computed by term occurrences in the posting list
- IDF: computed by the posting list cardinality



$$tf * idf(Alice, D1) = 2 * \log \frac{100}{3}$$

http://lucene.apache.org

APACHE LUCENE

Documentation

https://lucene.apache.org/core/8_11_2/index.html

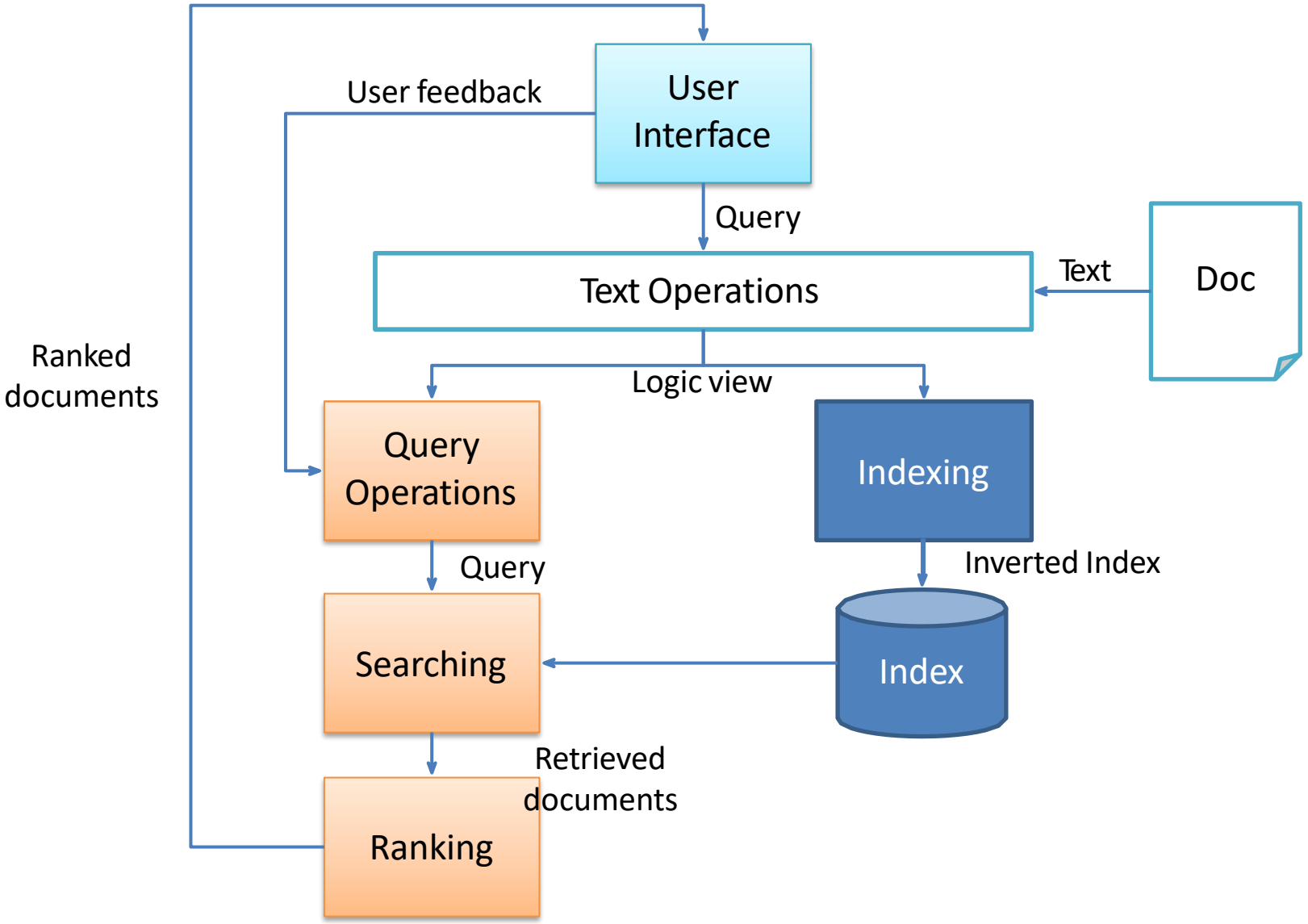
Apache Lucene

- Apache Software Foundation project
  - http://lucene.apache.org
- What Lucene is
  - Search library: indexing and searching Application Programming Interface (API)
- What Lucene is not
  - Search engine (no crawling, server, user interface, etc.)

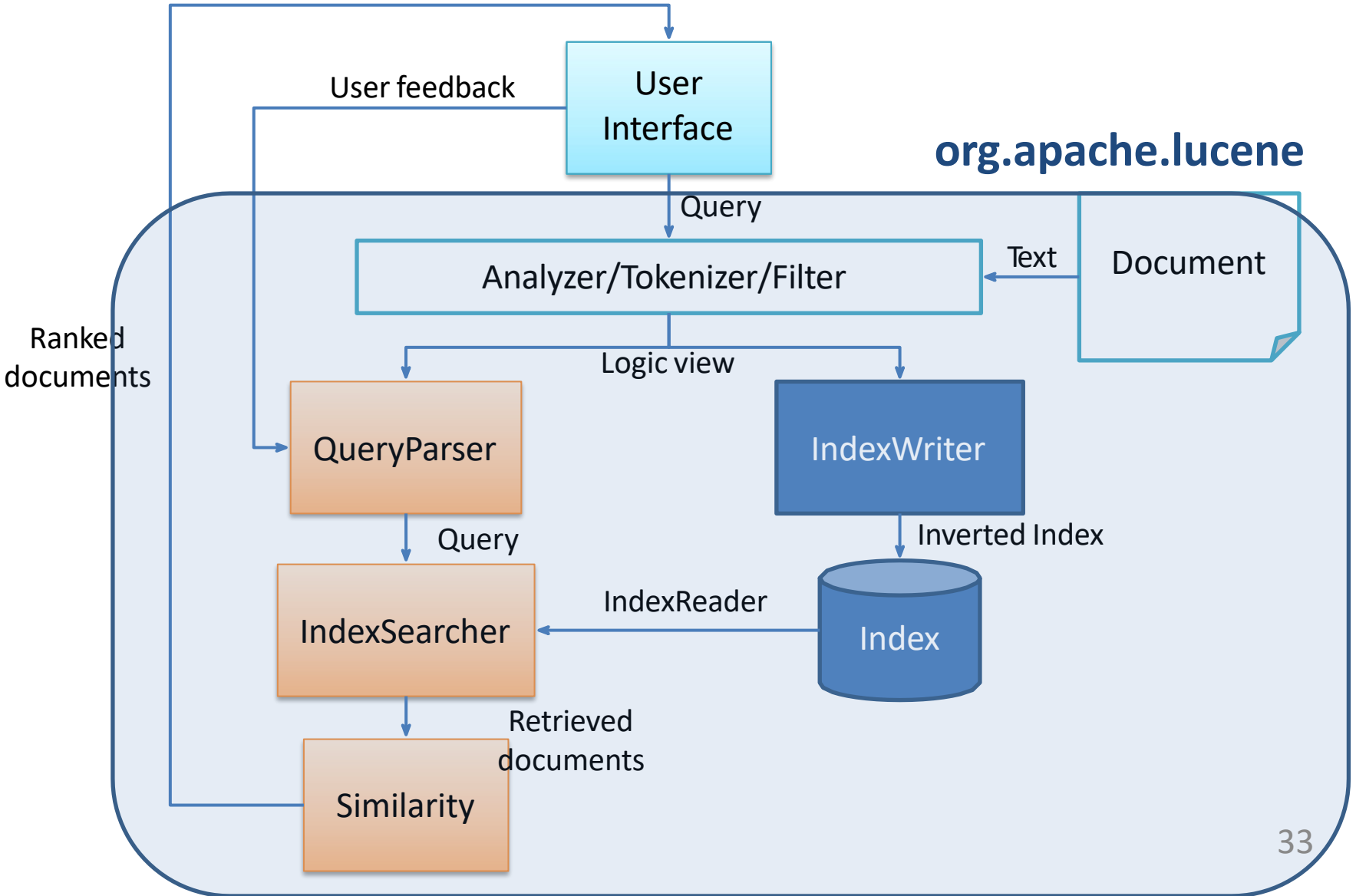
Lucene

- Full-text search
- High performance
- Scalable
- Cross-platform
- 100%-pure Java
- Porting in other languages:
  - Perl, C#, C++, Python, Ruby e PHP

Information Retrieval Process



Information Retrieval Process



Lucene Model 1/2

- Based on **Vector Space Model**
- Features:
  - **multi-field** document
  - **tf–term** frequency: number of matching terms in field
  - **lengthNorm**: number of tokens in field
  - **idf**: inverse document frequency
  - **coord**: coordination factor, number of matching
- Terms
  - field boost
  - query clause boost

Lucene Model 1/2

$$score(q,d) = coord(q,d) \cdot queryNorm(q,d) \cdot \sum_{t \in q} (tf \cdot idf^2 \cdot boost(t) \cdot norm(t,d))$$

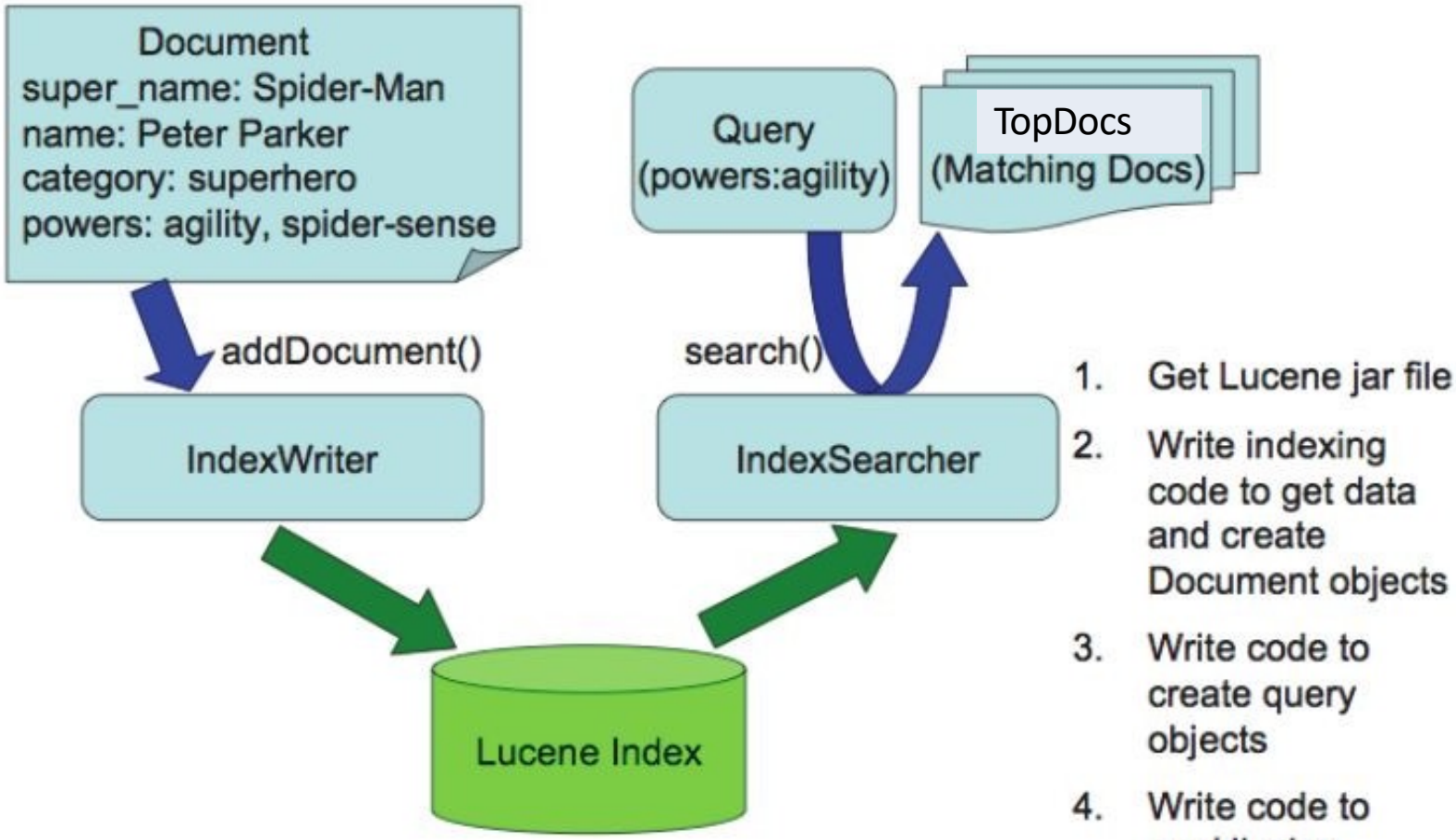
$$tf(t) = (\#occur(t))^{1/2}$$

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

- coor(q,d) score factor based on how many query terms are found in the specified document
- queryNorm(q,d) is a normalizing factor used to make scores between queries comparable
- boost(t) term boost
- norm(t,d) encapsulates a few (indexing time) boost and length factors

More info https://lucene.apache.org/core/8_6_2/core/index.html

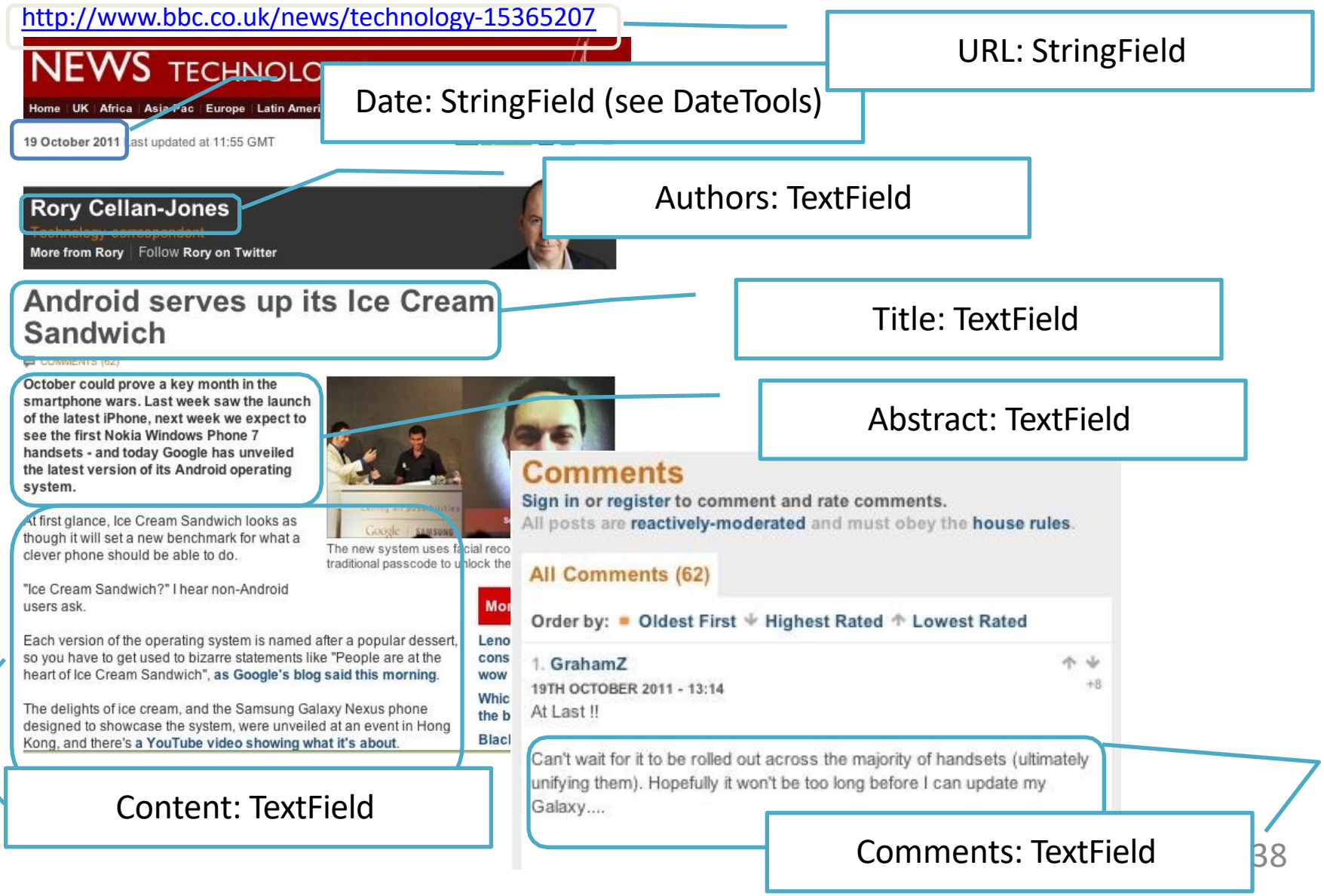
Lucene


Document
super_name: Spider-Man
name: Peter Parker
category: superhero
powers: agility, spider-sense
addDocument()
IndexWriter
Lucene Index
Query
(powers:agility)
search()
IndexSearcher
TopDocs
(Matching Docs)
1. Get Lucene jar file
2. Write indexing code to get data and create Document objects
3. Write code to create query objects
4. Write code to use/display results

Document: Fields Representation

http://www.bbc.co.uk/news/technology-15365207

Document: Fields Representation



Document

- A collection of *Field*s which give structure to the document
- Each Field is added to the document
- A Field has three parts
  - Name
  - Type
  - Value
    - text (String, Reader or pre-analyzed TokenStream)
    - binary (byte[])
    - numeric (a Number)
- A Field may be stored in the index (TYPE_STORED), so it may be returned with hits on the document.
- Use FieldType for personalized types

Hello World 1/3

- Setup a Java Maven project in your IDE

```
<dependencies>
        <dependency>
            <groupId>org.apache.lucene</groupId>
            <artifactId>lucene-core</artifactId>
            <version>8.11.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.lucene</groupId>
            <artifactId>lucene-analyzers-common</artifactId>
            <version>8.11.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.lucene</groupId>
            <artifactId>lucene-queryparser</artifactId>
            <version>8.11.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.lucene</groupId>
            <artifactId>lucene-queries</artifactId>
            <version>8.11.2</version>
        </dependency>
</dependencies>
```

Hello World 2/3

See the class `di.uniba.it.mri2223.lucene.HelloWorld`

```
//Open a directory from the file system (index directory)
FSDirectory fsdir = FSDirectory.open(new File("./resources/helloworld").toPath());
//IndexWriter configuration
IndexWriterConfig iwc = new IndexWriterConfig(new StandardAnalyzer());
//Index directory is created if not exists or
overwritten
iwc.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
//Create IndexWriter
IndexWriter writer = new IndexWriter(fsdir, iwc);
//Create document and add
fields Document doc = new
Document();
doc.add(new TextField("super_name", "Spider-Man", Field.Store.NO));
doc.add(new TextField("name", "Peter Parker", Field.Store.NO));
doc.add(new TextField("category", "superhero", Field.Store.NO));
doc.add(new TextField("powers", "agility, spider-sense",
Field.Store.NO));
//add document to index
writer.addDocument(doc);
//close IndexWriter
writer.close();
```

Hello World 3/3

```
//Create the IndexSearcher
IndexSearcher searcher = new IndexSearcher(DirectoryReader.open(fsdir));
//Create the query parser with the default field and analyzer
QueryParser qp = new QueryParser("name", new StandardAnalyzer());
//Parse the query
Query q = qp.parse("name:parker powers:agility");
//Search
TopDocs topdocs = searcher.search(q, 10);
System.out.println("Found " + topdocs.totalHits.value + " document(s).");
```

Field Constructors

- `Field(String name, byte[] value, IndexableFieldType type)`
  Create field with binary value.
- `Field(String name, byte[] value, int offset, int length, IndexableFieldType type)`
  Create field with binary value.
- `Field(String name, BytesRef bytes, IndexableFieldType type)`
  Create field with binary value.
- `Field(String name, Reader reader, IndexableFieldType type)`
  Create field with Reader value.
- `Field(String name, CharSequence value, IndexableFieldType type)`
  Create field with CharSequence value.
- `Field(String name, TokenStream tokenStream, IndexableFieldType type)`
  Create field with TokenStream value.

Adding documents to an Index

- `addDocument(Iterable<? extends IndexableField> doc)`

  Adds a document to this index.

- `addDocuments(Iterable<? extends Iterable<? extends IndexableField>> docs)`

  Atomically adds a block of documents with sequentially assigned document IDs, such that an external reader will see all or none of the documents.

Exercise 1

- Index the famous novel **Alice In Wonderland** in a Lucene searchable index.

- You should index each **Chapter** as a separate Lucene Document.

- Each Document should contain:
  1. The Title of the Book
  2. The Author of the Book
  3. The title of the Chapter
  4. The text of the Chapter

- The text of the Chapter should store the Term Vector and tokens offsets.

Text Operations

- Tokenization
  - split text in token
- Stop word elimination
  - remove common words and closed word class (e.g. function words)
- Stemming
  - reducing inflected (or sometimes derived) words to their stem

More info
https://lucene.apache.org/core/8_6_2/analyzers-common/index.html

Text Analysis

- Analyzer: encapsulates the analysis process
  - Tokenize a text by performing any number of operations on it
  - Responsible for building the TokenStream consumed by the indexing and searching processes
- Tokenizer: is a TokenStream
  - Responsible for breaking up incoming text into tokens
- TokenFilter: is also a TokenStream and is responsible for modifying tokens that have been created by the Tokenizer

Analyzers

- KeywordAnalyzer
  - Returns a single token
- WhitespaceAnalyzer
  - Splits tokens at whitespace
- Simple Analyzer
  - Divides text at non letter characters and lowercases
- StopAnalyzer
  - Divides text at non letter characters, lowercases, and removes stop words
- StandardAnalyzer
  - Tokenizes based on sophisticated grammar that recognizes e-mail addresses, acronyms, etc.; lowercases and removes stop words (optional)

Analyzers

The quick brown fox jumped over the lazy dogs

- WhitespaceAnalyzer:
  [The] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dogs]
- SimpleAnalyzer:
  [the] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dogs]
- StopAnalyzer:
  [quick] [brown] [fox] [jumped] [lazy] [dogs]
- StandardAnalyzer:
  [quick] [brown] [fox] [jumped] [over] [lazy] [dogs]

Analyzers

"XY&Z Corporation - xyz@example.com"

- WhitespaceAnalyzer:
  [XY&Z] [Corporation] [-] [xyz@example.com]
- SimpleAnalyzer:
  [xy] [z] [corporation] [xyz] [example] [com]
- StandardAnalyzer:
  [xy&z] [corporation] [xyz@example.com]

Tokenizers

- Tokenization is the process of breaking input text into small indexing elements, e.g. tokens
- A Tokenizer is a TokenStream and is responsible for breaking up incoming text into tokens. Usually Analyzer will use a Tokenizer as the first step in the analysis process

source string: “full-text lucene.apache.org”

- StandardTokenizer
  - “full” “text” “lucene.apache.org”
- WhitespaceTokenizer
  - “full-text” “lucene.apache.org”
- LetterTokenizer
  - “full” “text” “lucene” “apache” “org”

TokenFilters

- A TokenFilter is also a TokenStream and is responsible for modifying tokens that have been created by the Tokenizer. LowerCaseFilter
- StopFilter
- LengthFilter
- PorterStemFilter
  - stemming: reducing words to root form
  - rides, ride, riding => ride
  - country, countries => countri

Test Analyzer

See the class `di.uniba.it.mri2324.lucene.TestAnalyzer`

```
public static List<String> getTokens(Reader reader, Analyzer analyzer) throws
   IOException {
    List<String> tokens = new ArrayList<>();
    TokenStream tokenStream = analyzer.tokenStream("text", reader);
    tokenStream.reset();
    CharTermAttribute cattr = tokenStream.addAttribute(CharTermAttribute.class);
    while (tokenStream.incrementToken()) {
         String token = cattr.toString();
         tokens.add(token);
    }
    tokenStream.end();
    return tokens;
    }
```

Custom Analyzer

See the class `di.uniba.it.mri2223.lucene.MyAnalyzer`

```
public static final CharArraySet STOP_WORDS;

static {
    final List<String> stopWords = Arrays.asList("a", "an", "and", "are",
"the", "is", "but", "by");
    final CharArraySet stopSet = new CharArraySet(stopWords, false);
    STOP_WORDS = CharArraySet.unmodifiableSet(stopSet);
}

@Override
protected TokenStreamComponents createComponents(String fieldName) {
    Tokenizer source = new LetterTokenizer();
    TokenStream filter = new LowerCaseFilter(source);
    filter = new StopFilter(filter, STOP_WORDS);
    return new TokenStreamComponents(source, filter);
}
```

Lucene Search

- IndexReader: interface for accessing an index
- QueryParser: parses the user query
- IndexSearcher: implements search over a single IndexReader
  - search(Query query, int numDoc)
  - TopDocs -> result of search
    - TopDocs.scoreDocs returns an array of retrieved documents (ScoreDoc)

Lucene Search

See the class `di.uniba.it.mri2324.lucene.TestSearch1`

```
FSDirectory fsdir = FSDirectory.open(new
File("./resources/helloworld").toPath()); IndexSearcher searcher = new
IndexSearcher(DirectoryReader.open(fsdir));
//Single term query
//Query q = new TermQuery(new Term("name", "parker"));
//Boolean query
BooleanQuery.Builder qb = new BooleanQuery.Builder();
qb.add(new TermQuery(new Term("name", "parker")), BooleanClause.Occur.SHOULD);
qb.add(new TermQuery(new Term("powers", "agility")),
BooleanClause.Occur.SHOULD); Query q = qb.build();
TopDocs topdocs = searcher.search(q, 10);
System.out.println("Found " + topdocs.totalHits.value + " document(s).");
```

Lucene QueryParser

- Example:
  - `queryParser.parse("name:SpiderMan");`
- good human entered queries, debugging
- does text analysis and constructs appropriate queries
- not all query types supported

QUERY SYNTAX 1/2

- title:"The Right Way" AND text:go
  - Phrase query + term query
- Wildcard Searches
  - tes* (test - tests - tester)
  - te?t (test – text)
  - te*t (tempt)
  - tes?
- Fuzzy Searches (Levenshtein Distance) (TERM)
  - roam~ (foam – roams)
  - roam~0.8
- Range Searches
  - mod_date:[20020101 TO 20030101]
  - title:{Aida TO Carmen}
- Proximity Searches (PHRASE)
  - "jakarta apache"~10

QUERY SYNTAX 2/2

- Boosting a Term
  - jakarta^4 apache
  - "jakarta apache"^4 "Apache Lucene”
- Boolean Operator
  - NOT, OR, AND
  - + required operator
  - - prohibit operator
- Grouping by ( )
- Field Grouping
- title:(+return +"pink panther")
- Escaping Special Characters by \

QUERY (BY CODE) 1/2

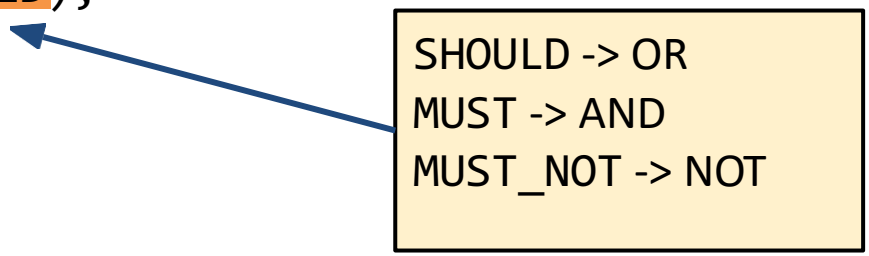
```
TermQuery query = new TermQuery(new Term("name","Spider-Man"))
```

- explicit, no escaping necessary
- does not do text analysis for you
- Query
  - TermQuery
  - BooleanQuery
  - PhraseQuery
  - FuzzyQuery / WildcardQuery /
- Se `TestSearch1` for examples of TermQuery and BooleanQuery
  e

QUERY (BY CODE) 2/2

```
TermQuery tq1 = new TermQuery(new Term(“name”,”pippo”))
TermQuery tq2 = new TermQuery(new Term(“name”,”pluto”))

BooleanQuery.Builder qb = new BooleanQuery.Builder();
qb.add(new TermQuery(new Term("name", "parker")),
   BooleanClause.Occur.SHOULD);
qb.add(new TermQuery(new Term("powers", "agility")),
   BooleanClause.Occur.SHOULD);
```



DELETING DOCUMENTS

IndexWriter

- `deleteDocuments(Term… terms)`
- `deleteDocuments(Query… queries)`
- `updateDocument(Term term, Iterable<? extends IndexableField> doc)`
  Updates a document by first deleting the document(s) containing term and then adding the new document.
- `updateDocuments(Term delTerm, Iterable<? extends Iterable<? extends IndexableField>> docs)`
  Deletes and adds a block of documents with sequentially assigned document IDs, such that an external reader will see all or none of the documents.
- deleting does not immediately reclaim space

Develop a “little” search engine

FIRST PROJECT

Exercise 1

Build a “little” search engine that indexes and searches text files into a folder

- main class `IndexSE` takes the folder name and the index directory name as arguments
  - the class indexes all the “.txt” into the folder
- main class `SearchSE` takes the index directory name and the query as arguments
  - the class searches the index

Exercise 1

See classes:

- `di.uniba.it.mri2324.lucene.se.IndexSE`
- `di.uniba.it.mri2324.lucene.se.SearchSE`

IndexSE

```
FSDirectory fsdir = FSDirectory.open(new File(args[1]).toPath());
IndexWriterConfig iwc = new IndexWriterConfig(new StandardAnalyzer());
iwc.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
IndexWriter writer = new IndexWriter(fsdir, iwc);
File[] files = dir.listFiles();
for (File file : files) {
   if (file.isFile() && file.getName().endsWith(".txt")) {
       Document doc = new Document();
       doc.add(new StringField("id", file.getAbsolutePath(),
Field.Store.YES));
       doc.add(new TextField("text", new FileReader(file)));
       writer.addDocument(doc);
    }
}
writer.close();
```

SearchSE

```
FSDirectory fsdir = FSDirectory.open(new File(args[0]).toPath());
IndexSearcher searcher = new IndexSearcher(DirectoryReader.open(fsdir));
//Create the query parser with the default field and analyzer
QueryParser qp = new QueryParser("text", new StandardAnalyzer());
//Parse the query
Query q = qp.parse(args[1]);
//Search
TopDocs topdocs = searcher.search(q, 10);
for (ScoreDoc sdoc : topdocs.scoreDocs) {
   System.out.println("Found doc, path=" +
searcher.doc(sdoc.doc).get("id") + ", score" + sdoc.score);
}
```

POSTING API

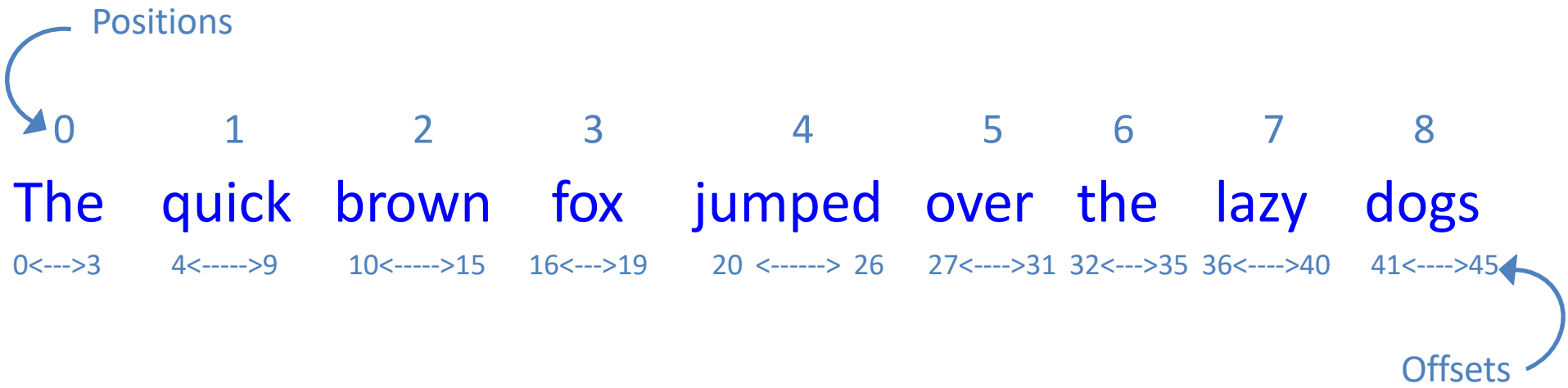
Posting API

- Getting access to term **frequency, positions and offsets**
- Required for:
  - Relevance Feedback and “More Like This”
  - Clustering
  - Similarity between two documents
  - Highlighter
    - needs offsets info

Posting API

- `Fields` is the initial entry point into the postings APIs
- `Terms` represents the collection of terms within a field
- `TermsEnum` provides an iterator over the list of terms within a field
- `PostingsEnum` is an extension of `DocIdSetIterator` that iterates over the list of documents for a term, along with the term frequency within that document
- `PostingsEnum` also allows iteration of the positions a term occurred within the document, and any additional per-position information (offsets and payload)

Term Vector: Positions and Offsets



Store Posting 1/2

- During indexing, create a `Field` that stores TermVectors through `FieldTyp` (we used TextField as a predefined IndexableField)
  - `FieldType ft = new FieldType(TextField.TYPE_NOT_STORED);`
  - `TYPE_STORED|TYPE_NOT_STORED`
- Invoke in cascade the following methods to set the field type:
  - `ft.setTokenized(true); //done as default`
  - `ft.setStoreTermVectors(true);`
  - `ft.setStoreTermVectorPositions(true);`
  - `ft.setStoreTermVectorOffsets(true);`
  - `ft.setStoreTermVectorPayloads(true);`

Store Posting 2/2

See class: `di.uniba.it.mri2324.lucene.se.post.IndexPostExample`

```
//define a custom field type that stores post information
FieldType ft = new FieldType(TextField.TYPE_NOT_STORED);
ft.setStoreTermVectors(true);
ft.setStoreTermVectorPositions(true);
ft.setStoreTermVectorOffsets(true);
…
Document doc1 = new Document();
doc1.add(new StringField("id", "1", Field.Store.YES));
doc1.add(new Field("text", new FileReader("./resources/text/es1.txt"), ft));
writer.addDocument(doc1);
```

Posting API

- Getting the BoW of a document: `di.uniba.it.mri2324.lucene.se.post.PostExample`

```
Fields fields = ireader.getTermVectors(docid);
for (String field : fields) {
      Terms terms = fields.terms(field);
      TermsEnum termsEnum = terms.iterator();
      BytesRef term = null;
      while ((term = termsEnum.next()) != null)
           {
           System.out.print(term.utf8ToString());
           PostingsEnum postings = termsEnum.postings(null,
           PostingsEnum.FREQS); while (postings.nextDoc() !=
           DocIdSetIterator.NO_MORE_DOCS) {
               System.out.println(":" + postings.freq());
           }
      }
}
```

More info
https://lucene.apache.org/core/8_6_2/core/org/apache/lucene/index/package-summary.html#postings

Posting API

- Getting positions and offsets:
  di.uniba.it.mri2324.lucene.se.post.PostExample

```
Fields fields = ireader.getTermVectors(docid);
for (String field : fields) {
      Terms terms = fields.terms(field);
      TermsEnum termsEnum = terms.iterator();
      BytesRef term = null;
      while ((term = termsEnum.next()) != null)
           {
           System.out.print(term.utf8ToString());
           PostingsEnum postings = termsEnum.postings(null,
           PostingsEnum.ALL); while (postings.nextDoc() !=
           DocIdSetIterator.NO_MORE_DOCS) {
               int position = postings.nextPosition();
               System.out.println("\t" + positio + "\t{" +
               for (int i = 0; i < postings.f ++) {
postings.startOffset( + ", " + postings.endOffset( + "}");
)                 }                 )
           }
      }
}
```

Exercise 2

- Take the `IndexSE` and add a third argument
  - the argument corresponds to different Field.TermVector... options
    - tv: term vectors
    - tvp: term vectors with positions
    - tvo: term vectors with positions and offsets
- Try Phrase queries in `SearchSE` when positions are stored

Implement an evaluation pipeline using the cranfield paradigm

Evaluation

Exercise 3

- Implement an evaluation pipeline
- Given
  - A test collection
  - Relevance assessments
  - Evaluation Metrics

  Find out the best indexing/searching configurations
- Experiment with
  - Analyzers
  - Query construction
  - Field structuring
  - Term/field boosting

Test collection

- All files are in ./resources/cran
- Cranfield Collection: collection of abstracts
- cran.all.1400.json: documents collection in JSON
  - fields: *id, text, authors, title, biblio*
- cran.qry.json: topics (queries) in JSON
  - fields: *id, query*
- See `di.uniba.it.mri2324.lucene.cran.HowToUseGson` for how to read JSON files
- cranqrel: relevance judgments
  - a map between a query and the set of relevant documents
    <qid> <run_num> <docid> <relevance>

Pipeline

- Index the documents collection
- Implement an evaluation class that for each topic (query) stores the top 100 retrieved documents
  - see the output format in the next slide
- Try different analyzers, queries formulation, term boosting … and measure the performance of your pipeline

The output format

- Output format, six fields separated by a space char:
  - query_id
  - run_identifier: may be every number (it is used to keep reference to the experiment)
  - doc_id
  - doc_rank
  - doc_score
  - exp_name: exp_name is a short reference string to the experiment
- `example.out` contains an output example

Trec_eval

- Program for running the evaluation
- Outputs many common evaluation measures:
  - Precision/recall
  - Number of [relevant] document retrieved
  - MAP, GMAP, P@K
- How to execute
  - `trec_eval` relevance_judgment_file output_file
- See: http://trec.nist.gov/trec_eval/index.html
- Versions already compiled for Linux, Mac and Win are in ./resources/trec_eval

Relevance feedback

Documents similarity

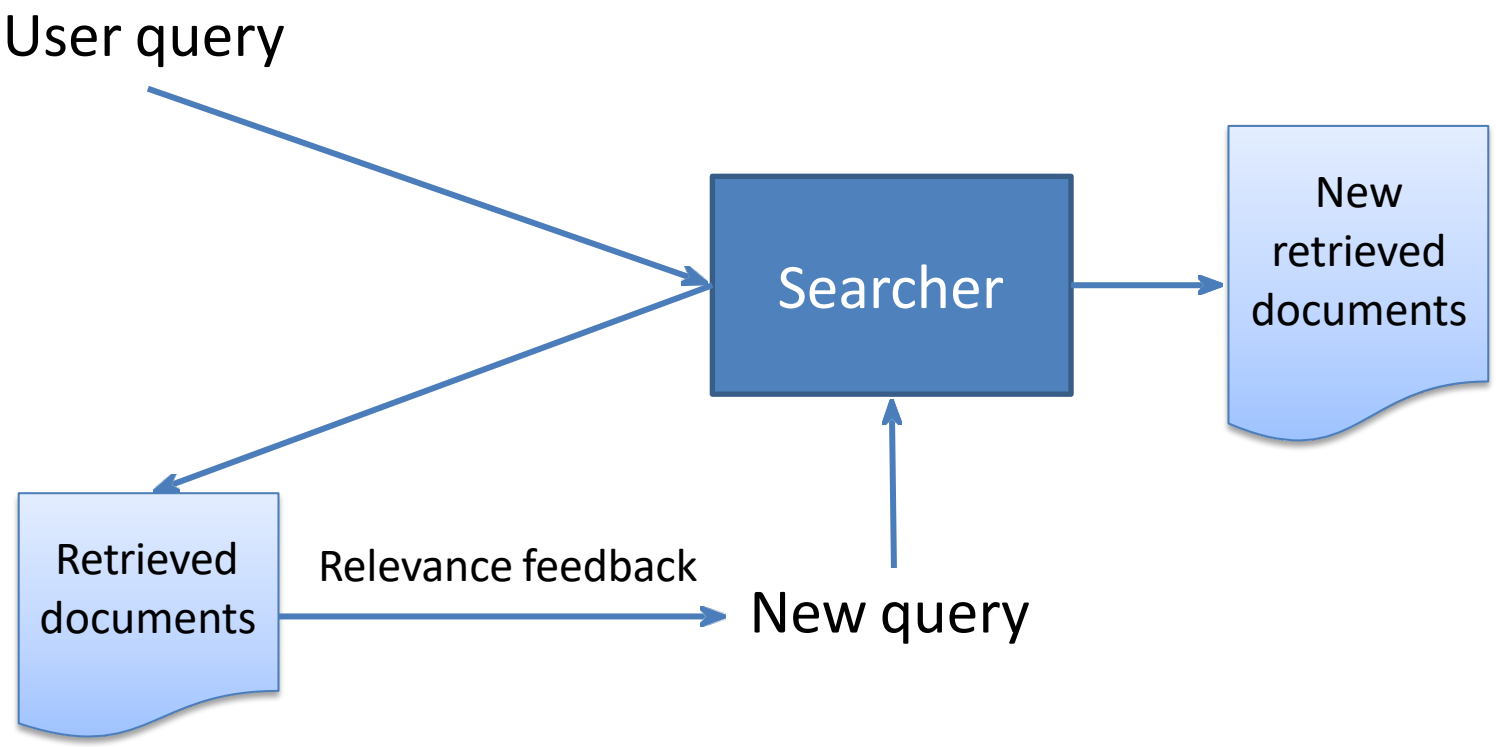
Apache Tika

ADVANCED TOPICS

Relevance feedback

- Improve retrieval performance using information about document relevance
  - Explicit: relevance indicated by the user
  - Implicit: from user behavior
  - Blind (or pseudo): using information about the top *k* retrieved documents

Relevance feedback



Rocchio Algorithm

$$Q_{new} = \alpha \cdot Q + \beta \cdot \frac{1}{|D_{rel}|} \cdot \sum_{D_j \in D_{rel}} D_j - \delta \cdot \frac{1}{|D_{norel}|} \cdot \sum_{D_k \in D_{norel}} D_k$$

Original query

Relevant document

Non-relevant document

Q_{new}, Q, D_j, D_i are vectors: BoW of a query or a document

Rocchio Algorithm (blind)

$$Q_{new} = \alpha \cdot Q + \beta \cdot \frac{1}{|D_{rel}|} \cdot \sum_{D_j \in D_{rel}} D_j - \delta \cdot \frac{1}{|D_{norel}|} \cdot \sum_{D_k \in D_{norel}} D_k$$

In blind (pseudo) relevance feedback we know only relevant documents (supposed to be the top *K* documents)

(generally adopted by search engine)

Relevance feedback

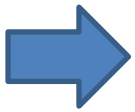
Q = Alice

It's a friend of mine — a Cheshire Cat,' said Alice: 'allow me to introduce it.

It's the oldest rule in the book,' said the King. 'Then it ought to be Number One,' said Alice

Alice looked round eagerly, and found that it was the Red Queen. 'She's grown a good deal!' was her first remark.

Alice lay back, and closed her eyes. There was the Red Queen again, with that incessant grin. Or was it the Cheshire cat's grin?



Q_{new} = Alice **Cheshire Cat King book**

It's a friend of mine — a Cheshire Cat,' said Alice: 'allow me to introduce it.

It's the oldest rule in the book,' said the King. 'Then it ought to be Number One,' said Alice

Alice book was reprinted and published in 1866.

…

(pseudo) relevance feedback in Lucene

Excercise 4

Relevance feedback in Lucene

- Possible using term vector (e.g. TermsEnum)
  - Retrieve the top *K* documents
  - Build the Q_{new} using term vector from the *K* documents
  - Re-query using Q_{new}
- Suggestions:
  - Map<String, Integer> for representing BoW as sparse vectors

Document similarity

Excercise 5

Document similarity

- Documents are represented by vectors
  - Build the document vector using TermsEnum
  - Compute the similarity using cosine similarity
- Implement a functionality like “similar to this document”
- Suggestions:
  - Map<String, Integer> for representing BoW as sparse vectors
  - Computes cosine similarity between the BoW of two Documents

CONCLUSIONS

Conclusions

- A popular IR model
  - Vector Space Model
  - Lucene supports other IR models: BM25, Language Modeling, ...
- Lucene
  - provides API to build search engine
- Apace Tika
  - extracts metadata and text from files and URLs
- **LET'S GO TO BUILD YOUR SEARCH ENGINE!**

Lucene API

- core: lucene core library
- analyzers-common: analyzers for indexing content in different languages and domains
  - Arabic, Chinese, Italian, …
- queryparser: query parser and parsing framework
- highlighter: a set of classes for highlighting matching terms in search results
- suggest: auto-suggest and spellchecking support

Lucene related project

- Apache Solr: open source enterprise search platform from the Apache Lucene
  - Full-Text Search Capabilities, High Volume Web Traffic, HTML Administration Interfaces

- Apache Nutch: web-search engine based on Solr and Lucene
  - crawler, link-graph database, HTML