Information Retrieval Exercises

Assignment 2:

Boolean Information Retrieval

Mario Sänger (saengema@informatik.hu-berlin.de)

Boolean IR on full text

- Last assignment: hard-coded queries on small data
- This time: arbitrary queries on large(r) data
- I will provide an movie corpus *plot.list*
  - Only for use in the exercise, do not redistribute!
  - Plain text are roughly 400 MB
  - Link: https://box.hu-berlin.de/f/93546ea9bdef4fac811b/?dl=1

- Task: Write a program which can find
  - arbitrary terms
  - arbitrary phrases
  - arbitrary conjunctions of them

Corpus structure (excerpt)

```
MV: Moonraker (1979)

PL: James Bond is back for another mission and this time, he is blasting off
PL: into space. A spaceship traveling through space is mysteriously hi-jacked
PL: and Bond must work quickly to find out who was behind it all. He starts
PL: with the rockets creators, Drax Industries and the man behind the
PL: organisation, Hugo Drax. On his journey he ends up meeting Dr. Holly
PL: Goodhead and encounters the metal-toothed Jaws once again.

BY: simon

PL: A Boeing 747 carrying a US space shuttle on loan to the UK crashes into the
PL: Atlantic Ocean. When the British examine the wreckage they can find no
PL: trace of the spacecraft and send agent James Bond to the shuttle's
PL: manufacturers, Drax Industries, to investigate.

BY: Dave Jenkins
```

Query syntax

- Searchable fields are as follows:
  - title
  - plot (if a document has multiple plot descriptions they can be appended)
  - type (movie, series, episode, television, video, videogame; see next slides)
  - year (optional)
  - episodetitle (optional, only for episodes)

Query syntax

- Token query syntax: <field>:<token>
  - Example: plot:Love
- Phrase query syntax: <field>:"<phrase>”
  - Example: title:”Robin Hood”
- Conjunction syntax: <query> AND <query> (where <query> can be a token, phrase, or AND query)
  - Example: title:”James Bond” AND plot:Russia AND plot:kill
- “ AND ” and double quotes not allowed in tokens or phrases
  - Don’t worry about queries like *title:"BATMAN AND ROBIN"*

IMDB Corpus

- Supported document types and their syntax:
  - movie: MV: <title> (<year>)
  - series: MV: "<title>" (<year>)
  - episode: MV: "<title>" (<year>) {<episodetitle>}
  - television: MV: <title> (<year>) (TV)
  - video: MV: <title> (<year>) (V)
  - videogame: MV: <title> (<year>) (VG)

- The corpus is in ISO-8859-1 format

  -
    ```
    BufferedReader reader =  new BufferedReader(
      new InputStreamReader(new FileInputStream(path),
        StandardCharsets.ISO_8859_1));
    ```

Documents

- An entry in the corpus file
  - Starts with “MV: ”
  - Ends with horizontal lines (“-------------”) or end-of-file
- Each entry must be treated as one document
  - A document can either match a query or not
  - Identified by their full title line in the corpus: e.g., *MV: Moonraker (1979)*

- Again, every document has up to five searchable fields: title, plot, type, year, episodetitle
- Other information (e.g., “BY: “) can be discarded

Peculiarities in the documents

- *MV: Disparity (2013) {{SUSPENDED}}*
  MV: "Moments" (2011) {Dreams (#1.1)} {{SUSPENDED}}
  → {{SUSPENDED}} can be discarded

- *MV: Disparity (????)*
  → Not all entries have a year field

- *MV: Displaced (2014/II)*
  MV: Displaced (2014/III)
  → Different documents may have identical name, year, and type

- *MV: Þegar það gerist (1998) (TV)*
  → Make sure to parse the file using ISO-8859-1 encoding!

Preprocessing

- The corpus text has to be split (“tokenized”) into terms to build indices
  - Use blanks, dots, commas, colons, exclamation marks, and question marks as term delimiter => (**.,:!?**)
  - Leave all other special characters untouched; they become parts of tokens

- Examples
  - “The Lord of the Rings: The Two Towers”
    - “the”, “lord”, “of”, “the”, “rings”, “the”, “two”, “towers”
  - “Marvel’s The Avengers”
    - “marvel’s”, “the”, “avengers”

Preprocessing

- Convert terms (for indices, term queries, and phrase queries) to lower case
  - Case-insensitive search!

- In phrase searches: the query is a consecutive sequence of terms
  - Document: "The Lord of the Rings: The Two Towers"
    - "the", "lord", "of", "the", "rings", "the", "two", "towers"
  - "the lord" matches the document
  - "he lord" doesn't match the document!
  - "lord the" doesn't match the document!

Program

- Implement the functions for building indices and running queries in BooleanSeach.java:
  - public void buildIndices(Path plotFile)
  - public Set<String> booleanQuery(String queryString)

- Keep attention to:
  - Only add classes and code, do not change or remove any code
  - Do not alter the functions�much signatures (types of parameters, return values)
  - Do not change the class or package name
  - Only use the default constructor and don‛t change its parameters

Challenges

- Parse “indexable” documents from an unstructured text file
  - Handle special characters
  - Handle unexpected syntax variants

- Conceptualize and implement indices
  - Separate indices different fields (title, plot, year, type)?
  - Index size will not be evaluated

- Efficient computation of document lists per term
  - Might be large (e.g., searching for “the”)

Challenges

- Efficient implementation of AND operator
  - Fast intersection of document lists

- Implementation of phrase search
  - How to efficiently index the terms for phrase searches?
  - Build separate indices for phrase searches?

- Efficient implementation of evaluating entire query
  - Choose an efficient evaluation order of the separate query parts

Test your program!

- We provide you with:
  - queries.txt: file containing exemplary queries
  - results.txt: file containing the expected results of running these queries
  - A main method for testing your code (which expects as parameters the corpus file, the queries file and the results file)

- Additionally, you can write your own test queries
  - check the plausibility of your results using GREP:
    grep " <search-token> " <corpus-file>
  - use -G or -P parameter for regular expressions

Submission

- **Group 1: Wednesday, 30.05., 23:59 (midnight)**
- **Group 2: Friday, 01.06., 23:59 (midnight)**

- Submit a ZIP archive named *ass2_<group-name>.zip*
  - Java source files of your solution
  - Compiled and executable BooleanQuery.jar

- Upload archive to the HU-BOX:
  https://hu.berlin/ire18_assignment2

Test your solution!

- Test your jar before submitting by running the examples queries on gruenau2
  - java -jar BooleanQuery.jar <plot list file> <queries file> <results file>
  - You might have to increase the JVM‘s heap size (e.g., -Xmx8g)
  - Your jar must run and answer all test queries correctly!

- Your program has to correctly answer all example queries correctly to pass the assignment!

Submission checklist

- Before submitting your results, make sure that you ...
  1. ... did not change or remove any code from BooleanQuery.java
  2. ... did not alter the functions‵ signatures (types of parameters, return values)
  3. ... only use the default constructor and don‵t change its parameters
  4. ... did not change the class or package name
  5. ... named your jar BooleanQuery.jar
  6. .. tested your jar on a gruenau host by running
     java -jar BooleanQuery.jar plot.list queries.txt results.txt
     (you might have to increase Java heap space, e.g. -Xmx6g)
  7. ... ascertained that the 15 queries in queries.txt were answered correctly

Solution presentation

- The presentation of the solutions will be given on 04.06. resp. 06.06

- You are be able to pick when and what you‘d like to present (first-come-first-served):
  - Group 1 (Mo): https://dudle.inf.tu-dresden.de/ire_ass2_mo/
  - Group 2 (We): https://dudle.inf.tu-dresden.de/ire_ass2_we/

- Presentation of the following aspects:
  - Corpus parser
  - Term search and indexing
  - Phrase search
  - AND search

Competition

- Search as fast as possible
- Build as many indices as you deem necessary
  - But: stay under 50 GB memory usage!

- I will call the program using a eval tool
  - I will use 9 different queries and -Xmx50g parameter

- The time for building the index counts as much as a single query
  - i.e., one tenth of the total achievable competition points

Possible solution: Inverted files

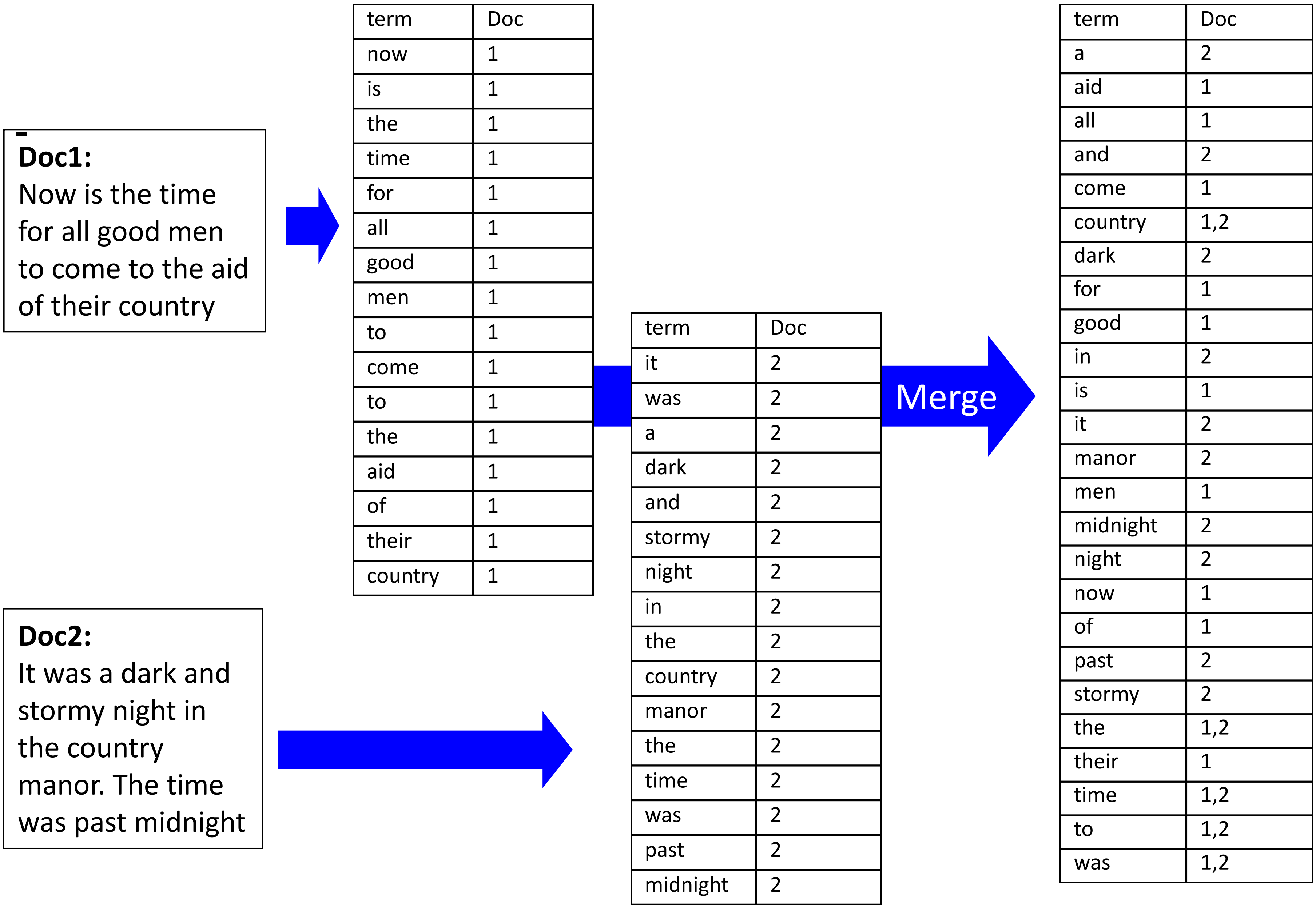
- Simple and effective index structure for searching terms in a collection of documents
  - Considers documents as “bag of words”

- “Inverted” view of documents:
  - Instead of “docs contain terms”, we use “terms appear in docs”

	term1	term2	term3
Doc1	1	0	1
Doc2	1	0	0
Doc3	0	1	1
Doc4	1	0	0
Doc5	1	1	1
Doc6	1	1	0
Doc7	0	1	0
Doc8	0	1	0

➡

	Doc1	Doc2	Doc3	Doc4	Doc5	Doc6	Doc7	Doc8
term1	1	1	0	1	1	1	0	0
term2	0	0	1	0	1	1	1	1
term3	1	0	1	0	1	0	0	0



Doc1:
Now is the time for all good men to come to the aid of their country

term	Doc
now	1
is	1
the	1
time	1
for	1
all	1
good	1
men	1
to	1
come	1
to	1
the	1
aid	1
of	1
their	1
country	1

Doc2:
It was a dark and stormy night in the country manor. The time was past midnight

term	Doc
it	2
was	2
a	2
dark	2
and	2
stormy	2
night	2
in	2
the	2
country	2
manor	2
the	2
time	2
was	2
past	2
midnight	2

term	Doc
a	2
aid	1
all	1
and	2
come	1
country	1,2
dark	2
for	1
good	1
in	2
is	1
it	2
manor	2
men	1
midnight	2
night	2
now	1
of	1
past	2
stormy	2
the	1,2
their	1
time	1,2
to	1,2
was	1,2

Boolean retrieval

- We can now efficiently implement Boolean queries
- For each query term term_i, look up document list Doc_i containing term_i
- Evaluate query in the usual order:
  - $\text{term}_i \wedge \text{term}_j : \text{Doc}_i \cap \text{Doc}_j$
- Example:
  - plot:time AND plot:past AND plot:the

    $= \text{Doc}_{\text{plot:time}} \cap \text{Doc}_{\text{plot:past}} \cap \text{Doc}_{\text{plot:the}}$

    $= \{1,2\} \cap \{2\} \cap \{1,2\}$

    $= \{2\}$

term	Doc
a	2
aid	1
all	1
and	2
come	1
country	1,2
dark	2
for	1
good	1
in	2
is	1
it	2
manor	2
men	1
midnight	2
night	2
now	1
of	1
past	2
stormy	2
the	1,2
their	1
time	1,2
to	1,2
was	1,2