Information Retrieval Exercises

Assignment 2:

Boolean Information Retrieval

Samuele Garda (gardasam@informatik.hu-berlin.de)

Boolean IR

- Arbitrary queries on large(r) data
- I will provide a corpus (file name: “*plot.list*”):
  - Only for use in the exercise, **do not redistribute**!
  - Plain text of roughly 400 MB
- Retrieve movies according to:
  - arbitrary terms & phrases
  - conjunctions of both
- You CANNOT use Apache Lucen library

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

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)*
- Other information (e.g., “BY: “) can be discarded

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));

Peculiarities in the documents

- {{SUSPENDED}} can be discarded
  - *MV: Disparity (2013) {{SUSPENDED}}*
- Not all entries have a year field
  - *MV: Disparity (????)*
- Same name, year, and type but **different plot**:
  - *MV: Displaced (2014/II)*
  - *MV: Displaced (2014/III)*
- *Non-latin alphabets (use* ISO-8859-1 encoding*):*
  - *MV: Þegar það gerist (1998) (TV)*

Preprocessing

- The corpus text has to be **tokenized** (split into terms) to build indices
  - Use **blanks**, **dots**, **commas**, **colons**, **exclamation and question marks** as delimiters => (**.,:!?**)
  - **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”

Query syntax

- Searchable **fields** are as follows:
  - title
  - plot (if a document has multiple plot descriptions they can be joined)
  - type (movie, series, episode, television, video, videogame)
  - year (optional, e.g. *(????)*)
  - episodetitle (optional, only for episodes)

Query syntax: boolean IR (AND)

- 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*"

Query syntax

- Phrase search:
  - Only for fields containing text (title, episodetitle, plot)
  - the query is a **consecutive sequence of terms**
  - e.g. “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**!

Query syntax

- **Case-insensitive search**:
  - convert terms to lower case
  - plot:Love = plot:love

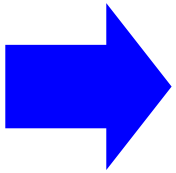
- Query might not match anything!

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”

	term 1	term 2	term 3
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

Merge

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:
  - $term_i$ ^ $term_j$: $Doc_i \cap Doc_j$
- Example:
  - plot:time AND plot:past AND plot:the

    $= Doc_{plot:time} \cap Doc_{plot:past} \cap Doc_{plot:the}$

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

    $= \{2\}$

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

Challenges

- Parse documents from an unstructured text file
  - Handle special characters
  - Handle unexpected syntax variants
- Conceptualize and implement indices
  - Separate indices (title, plot, year, type)?
  - How to efficiently index the terms for phrase searches?
  - Build separate indices for phrase searches?
- Index size will not be evaluated

Challenges

- Efficient computation of document lists per term
  - Might be large (e.g., searching for “the”)
- Efficient implementation of AND operator
  - Fast intersection of document lists
- Efficient implementation of evaluating entire query
  - Choose an efficient evaluation order of the separate query parts


ENOUGH BLABLA!!
JUST SHOW ME THE CODE!!
imgflip.com

The code

```java
import static java.nio.charset.StandardCharsets.ISO_8859_1;

public class BooleanQuery {

    /**
     * DO NOT CHANGE THE CONSTRUCTOR. DO NOT ADD PARAMETERS TO THE CONSTRUCTOR.
     */
    public BooleanQuery() {
    }

    /**
     * A method for reading the textual movie plot file and building indices. The
     * purpose of these indices is to speed up subsequent boolean searches using
     * the {@link #booleanQuery(String) booleanQuery} method.
     * <p>
     * DO NOT CHANGE THIS METHOD'S INTERFACE.
     *
     * @param plotFile the textual movie plot file 'plot.list' for personal, non-commercial
     *                 use.
     */
    public void buildIndices(Path plotFile) {
        // TODO: insert code here
    }

    /**
     * DO NOT CHANGE THIS METHOD'S INTERFACE.
     *
     * @param queryString the query string, formatted according to the Lucene query syntax,
     *                    but only supporting term search, phrase search, and the AND
     *                    operator
     * @return the exact content (in the textual movie plot file) of the title
     * lines (starting with "MV: ") of the documents matching the query
     */
    public Set<String> booleanQuery(String queryString) {
        // TODO: insert code here
        return new HashSet<>();
    }
```

The code

```
public static void main(String[] args) {
    BooleanQuery bq = new BooleanQuery();
    if (args.length < 3) {
        System.err.println("Usage: java -jar BooleanQuery.jar <plot list file> <queries file> <results file>");
        System.exit(-1);
    }

    Path moviePlotFile = Paths.get(args[0]);
    Path queriesFile = Paths.get(args[1]);
    Path resultsFile = Paths.get(args[2]);

    // build indices
    System.out.println("Building indices...");
    long tic = System.nanoTime();
    Runtime runtime = Runtime.getRuntime();
    long mem = runtime.totalMemory();
    bq.buildIndices(moviePlotFile);

    System.out.println("Runtime: " + (System.nanoTime() - tic) + " nanoseconds");
    System.out.println("Memory: " + ((runtime.totalMemory() - mem) / (1048576L)) + " MB (rough estimate)");

  // run queries
  for (int i = 0; i < queries.size(); i++) {
      String query = queries.get(i);
      Set<String> expectedResult = i < results.size() ? results.get(i) : new HashSet<>();
      System.out.println();
      System.out.println("Query:           " + query);
      tic = System.nanoTime();
      Set<String> actualResult = bq.booleanQuery(query);

      // sort expected and determined results for human readability
      List<String> expectedResultSorted = new ArrayList<>(expectedResult);
      List<String> actualResultSorted = new ArrayList<>(actualResult);
      Comparator<String> stringComparator = Comparator.naturalOrder();
      expectedResultSorted.sort(stringComparator);
      actualResultSorted.sort(stringComparator);

      System.out.println("Runtime:         " + (System.nanoTime() - tic) + " nanoseconds.");
      System.out.println("Expected result: " + expectedResultSorted.toString());
      System.out.println("Actual result:   " + actualResultSorted.toString());
      System.out.println(expectedResult.equals(actualResult) ? "SUCCESS" : "FAILURE");
  }
}
```

DO NOT MODIFY THIS

EXACTLY 3 ARGUMENTS:
- "plot.list" file
- queries file
- result file

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)

- **To pass the assignment you must solve correctly all the queries in “queries.txt”!**

Test your program!

- 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 checklist

- 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 BooleanQueryLucene.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. -Xmx8g)

Submission

- Submission:
  - Group 1: **Tuesday, 25.05, 23:59 (midnight)**
  - Group 2: **Wednesday, 26.05, 23:59 (midnight)**
- Presentation:
  - Group 1: **Tuesday, 01 June**
  - Group 2: **Wednesday, 02 June**
- Presentation of the following aspects:
  - Corpus parser
  - Indexing indexing
  - Term/Phrase/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 an evaluation 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

Questions?