Assignment 1

Information Retrieval and Text Mining 17/18

2017-10-26; to be submitted 2017-11-09

Roman Klinger, Florian Strohm

- **Deadline:** These assignment will be discussed on 2017-11-14. Submissions are accepted via Ilias until end of day 2017-11-09.
- **Groups:** Working in groups of up to three people is encouraged, up to four people is allowed. More people are not allowed. Copying results from one group to another is not allowed. Changing groups during the term is allowed.
- **Grading:** Passing the assignments is a requirement for participation in the exam in all modules IRTM can be part of. Altogether 80 points need to be reached. There are five assignments with 20 pen & paper points and 10 programming points each. That means, altogether, 150 points can be reached.
- **Submission:** First make a group in Ilias, then submit the PDF. Write all group members on each page of the PDF. Only submit *one* PDF file. If you are technically not able to make a group (it seems that happens on Ilias from time to time), do not submit a PDF multiple times by multiple people – only submit it once. Submission for the programming tasks should also be in the same PDF.

Pen and Paper Task 1 (3 points)

Given the following documents:

- Document 1: `he is he is he is he is he is nice`
- Document 2: `he she`
- Document 3: `she nice`
- Document 4: `he`
- Document 5: `he`
- Document 6: `she`
- Document 7: `she`
- Document 8: `she`
- Document 9: `he nice`

Subtask A

Build the full inverted *positional* index for these documents. Do not perform normalization.

Subtask B

Add skip pointers to the index from the previous task. Provide an example query which demonstrates the usefulness of skip pointers for your index and explain why this query can be answered in a more efficient way with skip pointers than without.

Pen and Paper Task 2 (1 points)

Are skip pointers helpful for queries in which two terms are connected by `OR`? Explain your answer.

Pen and Paper Task 3 (3 points)

Which strings are stored in the permuterm index for the word `car`?
How is this permuterm index queried when the user requests `c*r`?
Which of the strings in the permuterm is the one which answers the query?

Pen and Paper Task 4 (4 points)

Consider the following fragment of a positional index in the format
`term: (freq.) docID: <position, position, position, ...>; docID: ...:`

```
Gates:     (4) 1: <3>; 2: <6>;    3: <2,17>; 4: <1>;
IBM:       (2) 4: <3>; 7: <14>;
Microsoft: (4) 1: <1>; 2: <1,21>; 3: <3>;    5: <16,22,51>;
```

The $/k$ operator, word1 $/k$ word2 finds occurrences of word1 within k words of word2 (on either side), where k is a positive integer argument. Thus k = 1 means that word1 should be directly next to word2.

Which comparisons (on document ids or on the positions) are made when querying for `Gates /2 Microsoft`?

What is the result?

Pen and Paper Task 5 (4 points)

The permuterm index and the k gram index are both approaches to make wildcard queries possible. Describe in your own words what the advantage and disadvantages of each approach are. Explain!

Pen and Paper Task 6 (2 points)

Calculate the Levenshtein distance for the two terms “flower” and “flour”. Draw the matrix and explain.

Pen and Paper Task 7 (3 points)

Given the query `re*v*l`. One possibility to deal with multiple wildcards is to query for the margins: `l$re`. Does that work well? What are problems and how could these be addressed?

Programming Task 1 (10 points)

In the Ilias exercise session you found this PDF in, there is a file available called tweets.gz. Unpack this:

```
gunzip tweets.gz
```

Each line corresponds to one Tweet. The first column is the date, the second column is the Tweet ID, the third column and fourth column are user id and username, the fifth column is the Tweet text.

Implement a method `index(filename)` which takes the path to the file as an argument and puts all documents into a non-positional inverted index. You can assume that your computer's memory is sufficient to store all postings lists (in the unlikely event that it is not, only index a subset of the documents).

The index should consist of a dictionary and postings lists. Each entry of the dictionary should contain three values: The (normalized) term, the size of the postings list, the pointer to the postings list. Your data structure should be prepared to be able to store the postings lists separately from the dictionary, therefore do not just put a List data structure into an attribute of the dictionary.

The postings list itself consists of postings which contain each a document id and a pointer to the next postings. For the dictionary, you can use hashing methods included in your programming language (like Dictionary in Python or HashMap in Java) or tree structures as available in your programming language (for instance TreeMap in Java). For the postings lists, you can either implement the lists from scratch or use existing data structures (like Lists in Python or LinkedList in Java).

Then implement a method `query(term)`, where the argument represents one term as a `string` represents one term. It should return the postings list for that term.

Then, implement a method `query(term1, term2)`, where you assume that both terms are connected with a logical AND. Implement the intersection algorithm as discussed in the lecture for intersecting two postings lists. Do not access the lists array-style (for instance `listname[5]` where 5 is the position of the element you want to get). Use an iterator (in Python `listiter = iter(listname); next(listiter)` or in Java `iterator.next()`).

You can choose the programming language. Comment your code! Submit all code in the same PDF as the other tasks (pretty printed).

In addition, show the output of your program for the query `"stuttgart" AND "bahn"`. This output should at least show the Tweet ID and the text of all Tweets fulfilling the query.