D5: Databases and Information Systems
Advanced Topics in Information in IR, WS 2014/15
Dr. Klaus Berberich
Assignment 5, due: **22 Jan 2015**

max planck institut informatik

Please submit your solution as a PDF to `atir2014@mpi-inf.mpg.de` *by the indicated due date!*

Static Index Pruning

Problem 1. Static index pruning reduces the size of the index and improves query-processing performance by systematically removing postings from the index. In the lecture, we discussed a term-centric method by Carmel et al. [3] and a document-centric method by Büttcher et al. [2]. Both methods assume knowledge about the document collection but no knowledge about the query workload.

Please discuss how you would modify both methods if you had precise knowledge about the query workload Q. That is, for every query $q \in Q$, a set of terms from the vocabulary V, you know how many times $c(q)$ users issue this query.

Temporal Query Classification

Problem 2. Assume that you have access to a large query log with entries of the form

```
u167823     fifa world cup 1990     2015/01/11:10:46:22
```

telling you which user issued which query at which time. With this, how would yo

(a) identify *implicitly temporal queries*

(b) distinguish between *atemporal*, *temporally unambiguous*, and *temporally ambiguous* queries?

Non-Redundant Indexing

Problem 3. The non-redundant indexing method by Zhang and Suel [11] breaks up documents into smaller segments and indexes them using a two-level index structure. The first level maps words to segments, so that one can find segments that contain a specific word; the second level maps segments to documents, so that one can find out which documents contain the previously identified segments. While ideal for web archives, the method does not support time-travel text search, i.e., evaluating a query "as of" a given time in the past.

How would you modify the two-level index structure, so that it supports it? Please also discuss how you would process time-travel text queries on the modified index structure.

Temporal Coalescing

Problem 4. Prove that the temporal coalescing method discussed in the lecture is indeed optimal, i.e., the output sequence O produced has the minimal number $|O|$ of postings.

Note: The method discussed is an improved version of what is described in [2].

Cross-Time Information Retrieval

Problem 5. Search in historical document collections needs to bridge a vocabulary gap between queries issued by today's user and old documents written in archaic language. In the lecture we saw one example [6] how language evolution (e.g., different spellings) can be dealt with. Terminology evolution is a highly related problem: sometimes entities change their name (e.g., Saint Petersburg was known as Leningrad thirty years ago) or new concepts drive away old ones with similar functionality (e.g., walkman/ipod, fax/e-mail, altavista/google).

Assuming that you have a large collection of timestamped documents at hand (e.g., a newspaper archive), devise a similarity measure sim which allows you to compare two words u and v, in their semantics, when used at times (e.g., during a specific year) t_u and t_v respectively. For instance, your similarity measure should indicate high similarity $\text{sim}(\text{walkman}, 1987, \text{ipod}, 2007)$ between walkman when used in 1987 and ipod when used in 2007.

Winnowing (Programming Assignment)

Problem 6. Implement the winnowing method discussed in the lecture in a programming language of your choice. The original paper by Schleimer et al. [10] provides good clues on how to do so efficiently.

Use again the articles published in June 2002 by The New York Times from Assignment 4. Apply your implementation of winnowing to these articles (using $b = 3$ and $w \in \{5, 10, 25, 50\}$) and determine the following for each choice of w:

1. Number of distinct segments
2. Minimum, mean, and maximum segment length
3. Distribution of the number of documents that use a segment (as a plot)
4. Top-5 most reused segments

Hint: In Java the static method `Arrays.hashCode()` gives you a content-dependent hash code for an array, whereas the object's `hashCode()` does not. Using the latter you may obtain different hash codes for content-wise equal arrays.