DD2476 Search Engines and Information Retrieval Systems

Assignment 1: Boolean Retrieval

Johan Boye, Jussi Karlgren, Hedvig Kjellström

The purpose of Assignment 1 is to learn how to build an inverted index. You will learn 1) how build a basic inverted index; 2) how to handle multiword queries; 3) how to handle phrase queries; 4) how to evaluate a search system; and 5) techniques for handling large indexes.

The recommended reading for Assignment 1 is that of Lectures 1-3.

Assignment 1 is graded, with the requirements for different grades listed below. In the beginning of the oral review session, the teacher will ask you what grade you aim for, and ask questions related to that grade. All the tasks have to be presented at the same review session – you can not complete the assignment with additional tasks after it has been examined and given a grade. ***Come prepared to the review session!*** *The review will take 15 minutes or less, so have all papers in order.*

E: *Completed Task 1.1, 1.2 and 1.3 with some mistakes that could be corrected at the review session, completed Task 1.4.*
D: *E + Completed Task 1.1, 1.2 and 1.3 without mistakes.*
C: *E + Completed Task 1.5.*
B: *C + UNDER DEVELOPMENT, WILL BE DECIDED BEFORE FEBRUARY 7.*
A: *B + UNDER DEVELOPMENT, WILL BE DECIDED BEFORE FEBRUARY 7.*

These grades are valid for review February 18, 2014. See the web pages www.kth.se/social/course/DD2476, ir14 - Computer assignments in the menu, for grading of delayed assignments.

Assignment 1 is intended to take around 50h to complete.

Computing Framework

The code needed for the assignment can be found in the course directory `/info/DD2476/ir14/lab` on the CSC UNIX system. Copy the sub-directory `ir` to your own home directory. This directory contains the source code skeleton which you will use as the basis for the assignments 1-3. The code of each assignment build on that of the previous assignment.

If you are using your own computer, you will need to copy the sub-directories `pdfbox`, `megamap`, `pics` and parts of `svwiki` as well. The `svwiki` directory is very large (several Gigabytes), but for task 1.1-1.4 it is sufficient if you copy the

subdirectories `svwiki/files/1000`, `svwiki/files/2000`, etc, up to `svwiki/files/5000`.

If you are using your own computer you will furthermore need to modify the following line in the file `ir/SearchGUI.java`:

```
public static final String homeDir =
"/info/DD2476/ir14/lab";
```
and instead put the name of the directory you are using on your computer, e.g.

```
public static final String homeDir = "c:/ir-course";
```

When you have done this, you are ready to start working on the assignments on your own computer.

When you have copied the `ir` directory to your home directory, compile the lab skeleton as follows:

```
javac -Xlint:none -cp .:/info/DD2476/ir14/lab/pdfbox ir/*.java
```

If you are working in Windows, you should replace `:` with `;`:

```
javac -Xlint:none -cp .;pdfbox ir/*.java
```

You might see some warnings; these may be ignored. When you have compiled the program, you can run it by:

```
java -Xmx1024m -cp .:/info/DD2476/ir14/lab/pdfbox ir.SearchGUI
-d /info/DD2476/ir14/lab/svwiki/files/1000 -d
/info/DD2476/ir14/lab/svwiki/files/2000
```

You should now see the GUI where search queries are entered and search results presented. The `-d` flag on the line above tells the program which directory to index. You might specify any number of directories by `-d directory1 -d directory2 -d directory3`, etc. The program indexes all text files and readable PDF files in the specified directories.

Enter some words in the search text box and press Enter. You will now see in the result text area:

```
Found 0 matching document(s)
```

In order for the system to actually find some matching documents, you will now need to add code to some of the classes in the `ir` directory.

Task 1.1: Basic Inverted Index

The `processFiles` method in the `Indexer` class reads files and produces a stream of tokens to be indexed. However, currently the program is incomplete, and no index is built. **Your task is to complete the classes `HashedIndex`, `PostingsList` and `PostingsEntry` (and possibly add classes of your own), so that the program builds an inverted index.** In the lectures we have used a linked-list data structure for

postings lists. However, you should in your implementation use hash tables to increase computational efficiency.

When you have finished adding to the program, compile and run it, indexing the 10 data sets `svwiki/files/1000`, `svwiki/files/2000`, …, `svwiki/files/10000`. Try the search queries

antiken

which should result in **167** retrieved documents, and

fysik

which should result in **286** retrieved documents.

At the review

There will not be any examination of Task 1.1, it is merely a preparation for Task 1.2.

Task 1.2: Multiword Queries

The search engine implemented in Task 1.1 can only handle single word queries; it will now be extended to queries consisting of several words.

Extend the `search` method in the `HashedIndex` class **to implement the intersection algorithm** (page 11 in the textbook), so that you can do multiword searches.

(You can choose between intersection queries and phrase queries in the "Search options" menu. The "Intersection query" option should be selected by default, so you do not need to do anything. The option ”Ranked retrieval” and the ”Ranking score” menu relates to Assignment 2, and the button bar at the bottom of the GUI relates to Assignment 3.)

When you have finished adding to the program, compile and run it, indexing the 10 data sets `svwiki/files/1000`, `svwiki/files/2000`, …, `svwiki/files/10000`. Try the search queries

modern historia

which should result in **378** retrieved documents,

november eller december

which should result in **428** retrieved documents, and

tillvarons yttersta grunder

which should result in **2** retrieved documents.

At the review

To pass Task 1.2, you should show that the search engine in intersection query mode indeed returns the correct number of documents in response to the specified queries. You should also be able to explain all parts of the code that you edited, draw the data structure on paper, and explain from that figure how an intersection query is executed.

Task 1.3: Phrase Queries

Modify your program so that it is possible to search for contiguous phrases like "modern historia" using the techniques described in Section 2.4.2 in the textbook. Only documents that contain that exact phrase should be returned; documents that include the words at separate places should **not** be returned. Note that the algorithm on page 39 is **not** exactly what you are looking for.

You will need to add code to the `search` method in the `HashedIndex` class, so that when this method is called with the `queryType` parameter set to `Index.PHRASE_QUERY`, the search query should be treated as a phrase query. When the method is called with `queryType` parameter set to `Index.INTERSECTION_QUERY`, the program should behave as in Task 2 above.

Compile the program and run it, indexing the 10 data sets `svwiki/files/1000`, `svwiki/files/2000`, ..., `svwiki/files/10000`. You can choose between intersection queries and phrase queries in the "Search options" menu. Choose the "Phrase query" option and try the same search queries as before,

modern historia

which should result in **15** retrieved documents,

november eller december

which should result in **1** retrieved documents, and

tillvarons yttersta grunder

which should result in **2** retrieved documents.

At the review

To pass Task 1.3, you should show that the search engine in phrase query mode indeed returns the correct number of documents in response to the specified queries. You should also be able to explain all parts of the code that you edited, draw the data structure on paper, and explain from that figure how a phrase query is executed.

You should also prepare a comprehensive answer to the question *Why are fewer documents generally returned in phrase query mode than in intersection query mode?*

Task 1.4: What is a good search result?

The objective of this task is to **reflect on what constitutes a good answer to a query**. Run the program from Task 1.3, indexing the 10 data sets `svwiki/files/1000`, `svwiki/files/2000`, ..., `svwiki/files/10000`. Choose the "Intersection query" option in the "Search options" menu.

Search the indexed data sets with each of the following 10 queries:

antikens underverk (3 hits)

olympiska spel och fred (9 hits)

europacupen (16 hits)

konflikten i palestina (17 hits)

snowboard (13 hits)

den europeiska bilindustrin (3 hits)

enhetlig europeisk valuta (2 hits)

sex i reklam (23 hits)

lutande tornet i pisa (4 hits)

genteknik (6 hits)

For each of the above ten queries, save the list of returned documents in some format you are comfortable reading through. Assess the relevance of each document for the query. Use the following four-point scale:

(0) Irrelevant document. The document does not contain any information about the topic.
(1) Marginally relevant document. The document only points to the topic. It does not contain more or other information than the topic description.
(2) Fairly relevant document. The document contains more information than the topic description but the presentation is not exhaustive.
(3) Highly relevant document. The document discusses the themes of the topic exhaustively.

[E. Sormunen. Liberal relevance criteria of TREC—Counting on negligible documents? *ACM SIGIR*, 2002]

Edit the results into a **plain text file** using the following space-separated format, one line per assessed document:

```
QUERY_ID DOC_ID RELEVANCE_SCORE
```

where `QUERY_ID` = [1, ..., 10], `DOC_ID` = the name of the text file, e.g., 365 for `365.txt`, `RELEVANCE_SCORE` = [0, 1, 2, 3]. Name the text file `FirstnameLastname.txt` and send it to `hedvig@kth.se`.

It should be noted that there is no objectively correct relevance label for a certain query-document combination! It is a matter of judgement. For difficult cases, write a short note on why you chose the label you did. At the review, you will present three difficult cases for each query.

The documents are readable in a standard text editor. However, a first fast way of seeing what articles are about, is to look in the file `articleTitles.txt`, as:

```
> egrep "^365;"
  /info/DD2476/ir14/lab/svwiki_links/articleTitles.txt
365;Danmark
```

For each of the ten queries, compute the **precision** (relevant documents = documents with relevance > 0) of the returned list. (The **recall** is not possible to estimate without further information.)

At the review

To pass Task 1.4, you should show the text file with labeled documents for the 10 queries, in the correct format. You should have emailed it before presenting. You should be able to explain the concepts precision and recall, and give account for the precision of each of the returned document lists.

You should also, for at least 3 documents per query, which you thought were hard to label, expand on why you chose the labels you did.

NOTE: You will not be failed in the review for selecting the ”wrong” labels. However, you might be asked to redo this task if you are unable to motivate your labels in a satisfactory manner.

Task 1.5: What is a good query? (C)

In this task, you will again use the same search settings as in Task 1.4. The objective is here to **reflect on how a user formulates a query based on an information need**, and how the query is merely an incomplete representation of the latent information need.

Consider the following list of 3 information needs (English translation provided):

Ta fram statistik över skilsmässofrekvensen i olika länder. (Find statistics about the divorce frequency of different countries.)

Sök efter dokument som tar upp den politiska och ekonomiska framtiden för exklaven Kaliningrad. (Find documents treating the political and economic future of the Kaliningrad exclave.)

Leta efter information om FN-konventionen om barns rättigheter. (Find information about the UN convention on children’s rights.)

For each information need description, design a query (in Swedish) which you think will return documents relevant to the information need. Do an intersection search using this query, and label the returned documents as ”relevant” or ”non-relevant” in exactly the

same manner as in Task 1.4. **Remember to save all queries and all returned lists for the review**.

Now, go back to the information need description and ponder if you can improve the precision and recall of the returned list by removing, adding or exchanging words in the query. Do a search using the new query, and see if a higher share of relevant documents were returned. Repeat this until you have found an ”optimal” query for each of the 10 information need descriptions.

At the review

To pass Task 1.4, you should show the optimized query for each information need, as well as the full list of query variants leading up to the final query. You should also, for each information need, expand on why you think that the final query gave better precision and/or recall than the earlier variants.

Furthermore, prepare a comprehensive answer to the question *Why can we not simply set the query to be the entire information need description?*

Task 1.6: Large Inverted Indexes (B or higher)

In realistic applications, the complete index is too large to fit in working memory. In this task, you will implement a methodology for storing the index on disk, reading to and from working memory during query execution.

> Tip: If you are working on the CSC UNIX system, you do not have enough disc space on your account. You can then use the `tmp/` directory on the local hard drive of the Linux machine you are using.

You are free to use any kind of solution. A requirement is however that the **computational complexity of evaluating a query on this index should be sub-linear, O(<n)**. That is, evaluation of a query on an index of 20 000 documents should take **significantly less than 10 times** longer than the evaluation of the same query on an index of 2 000 documents. (This excludes, for instance, solutions in which all postings lists are saved in one file.)

NOTE: There are no absolute time limits for a query execution in your implementation. Furthermore, the initial indexing step is very computationally expensive. It is natural that indexing of such a large data set takes several hours.

NOTE: The indexing should have been done before the review session. Therefore you need to provide functionality for indexing, storing the index when the GUI is terminated, and then accessing the same stored index when the GUI is restarted. There are menu options for this in the File menu of the GUI.

At the review

To pass Task 1.6, you should be able to explain the storing solution you are using, and show both how the GUI can be started, using an index collected beforehand, as well as execution of a query (that is, allow the teacher to search the index).

You should show proof of evaluation of the computational complexity, in the form of a diagram over computation times for the same query evaluated on datasets of different numbers of documents (5 000, 10 000, 15 000, etc). This graph should show a sub-linear time complexity. You should also be able to explain all parts of your code.

UNDER DEVELOPMENT, REQUIREMENTS WILL BE DECIDED BEFORE FEBRUARY 7.