Chair for Algorithms
and Data Structures
Prof. Dr. Hannah Bast
Claudius Korzen

Information Retrieval
WS 2018/2019

http://ad-wiki.informatik.uni-freiburg.de/teaching

Exercise Sheet 1

Submit until Tuesday, October 23 at **12:00 noon**.

Exercise 1 (10 points)

Extend the code from the lecture by the following functionality. See the back side of this sheet for our (mandatory) guidelines about coding exercises in this course.

1. Make sure that each inverted list contains a particular record id at most once, even if the respective word occurs multiple time in the same record. Make sure that the whole construction algorithm still runs in time linear in the number of words in the input. (2 points)

2. Write a function *intersect* that computes the intersection of two inverted lists. The function must run in time linear in the total number of elements in the two lists and you must not use a library function. In particular, don't implement the lists as *sets*. (3 points)

3. Write a function *process_query* that, given a keyword query, fetches the inverted lists for each of the keywords and computes their intersection (empty, if there is no inverted list for a keyword), using your *intersect* function. (3 points)

4. Write a *main* function that constructs an inverted index from a given text file (one record per line, file name given as first argument on the command line) and then, in an infinite loop, asks the user for keyword queries and outputs 3 matching records. Optionally (= not mandatory to get full points) highlight the query words in the output, e.g., using ANSI escape codes. (2 points)

Exercise 2 (5 points)

Try your code on the file *movies.txt* provided on the Wiki. Find a query that gives good results (the records shown meet your expectations) and one that does not. Write them in your *experiences.txt* (see below), and very concisely (in one or two sentences) explain why one works and the other doesn't.

Exercise 3 (5 points)

Register with our course system Daphne (using your RZ account + password for authentication). Make sure that you are reachable under the specified e-mail address. Check out a working copy of your folder in the SVN repository of the course, and add your code to a new subdirectory *sheet-01*, and commit it. Make sure that everything runs through without errors on Jenkins.

[please turn over with anticipation]

Also commit, in that subdirectory, a text file *experiences.txt* where you briefly describe your experiences with the first exercise sheet and the corresponding lecture. As a minimum, say how much time you invested and if you had major problems, and if yes, where.

Guidelines for the coding exercises (mandatory + valid throughout the course):

1. You can code in Python, Java, or C++. You can make different choices for different exercise sheets. In the lecture, we will often use Python, which will allow us to focus on the conceptual issues. When efficiency is a central issue, we will use Java or C++.

2. For some exercise sheets, the code from the lecture will be available only in Python. You are still free to use Java or C++ for those sheets, but it will be more work for you then.

3. If the exercise sheet comes with a TIP file (the sheet above does), read it. It contains the exact specifications of what to compute and may also contain valuable implementation advice (which you can but do not have to follow). It also contains test cases for some of the functions.

You must implement these test cases whenever they are specified, also for future exercise sheets. Otherwise your submission will not be graded. The content of the test cases is important, not the exact syntax.

4. Our coding conventions must be followed at all times. In particular:

4.1 You have to write a unit test for each non-trivial method. One non-trivial example per unit test is enough. All unit tests must be fast. If your test reads an input file, that file must be small.

4.2 You have to adhere to our coding style.

4.3 You have to provide a standard build/make file along with your code and you have to make sure that everything runs through without errors on our continuous build system (Jenkins).

You find example code for all three languages in the *public/code/lecture-01* folder in the SVN.

Submissions that do not pass Jenkins will not be graded — Lecture 1 explains why

5. Make sure that you do not upload any "by-products" to our SVN (e.g., class files or executables or any stuff from your local environment). Also *never ever* upload large data files to our SVN; this is considered sin and will result in excommunication.

6. When you encounter implementation problems, proceed as follows. First do the obvious Google search: very often, this leads to a page describing exactly the problem you are having and the solution for it. Then search/read our forum (link on the Wiki) to see if someone has already asked a similar question. Then you are very welcome to ask a question on the forum yourself. Do ask before you spent a lot of time on minor implementation issues.

7. You are allowed to work on the exercises in groups of at most two. If you want to work in such a group, send a mail with the name of both of your RZ accounts to Axel Lehmann (lehmann@cs.uni-freiburg.de), with your partner in the CC (just to make sure that he/she agrees to work in a group). Axel will then create a joint subfolder for you in our SVN.