Chair for Algorithms
and Data Structures
Prof. Dr. Hannah Bast
Patrick Brosi

Information Retrieval
WS 2016/2017

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

UNI FREIBURG

Exercise Sheet 6

Submit until Tuesday, December 6 (Nikolaus) at **2:00pm**

This exercise sheet and the next are about implementing a fully functional web application for fuzzy prefix search. That is, by entering the right URL in a browser, you get a web page with an input field, and after each letter typed, you get a drop-down menu with a selection of fuzzy completions computed according to ES5 (with the same dataset: city names and their locations). What happens when you select one of these cities is left to your creativity.

For most of you, this will be the first time you build a web application and so there will be quite a number of (simple, yet) new concepts and languages to grasp (socket communication, HTTP, MIME types, HTML, CSS, DOM, AJAX, JSON, JavaScript, jQuery, Unicode). We have therefore split the work on this web application between this exercise sheet (ES6) and the next (ES7). There is a minimum functionality which you have to implement for ES6 (see below), and there are some additional requirements which will only be specified on ES7 (see next week).

Exercise 1 (20 points)

Build a web application with the following components and functionality:

1. Extend the server code provided on the Wiki (you can choose between Java and C++) to serve HTTP requests for HTML, CSS, JavaScript and plain text, with the correct response headers and MIME types. When a requested file is not found, return an appropriate 404 header.

Note: This was done live in the lecture, however, only a part of that code is provided on the Wiki. The reason is that you only remember this stuff by doing it yourself. Also note that Python is not an option for this sheet, because the next item requires that you integrate your code from ES5, for which Python was not an option (because of efficiency reasons).

2. Extend your code to answer fuzzy prefix queries using your code from the last exercise sheet. Requests should be of the form *http://<host>:<port>/?q=<query>*. The list of completions should be returned as a JSON object (with the right MIME type).

Note: Check your implementation by starting the server and trying out the URLs from the TIP file on the Wiki. Your tutor will do the same when correcting your submission.

3. Provide a *Makefile* with a target *start* and two variables *PORT* and *FILE*, such that *make PORT=<port> FILE=<file> start* starts your server such that it computes fuzzy completions from *<file>* and listens to requests on port *<port>*. See the Makefile on the Wiki for an example. C++ users should add the target to the Makefile they have anyway. Java users should have a Makefile (with only the *start* target) in addition to their *build.xml*.

4. Implement a web page, using an HTML and a separate CSS file, as shown in the lecture. The web page must contain an input field and at least some minimal design. Beyond that give free reign to your creativity. The URL of the web page should be *http://<host>:<port>* (without any suffix, think about the GET request produced by this URL and respond with the HTML from your web page in return), where *<host>* is the name of the machine where the server is running and *<port>* is the port on which the server is listening.

The four items above are the minimum requirement for ES6. The remaining items will be part of ES7, but you are free to do them now already. It's certainly more fun to do so, because you can then see your web app in action. Note that you will only get your points for these items when ES7 is corrected (and you are you free to make arbitrary changes to these items when you work on ES7).

5. Write the JavaScript (and include it into your HTML file) such that in your web app, after each keystroke, a list of fuzzy completions of whatever the user typed so far is displayed. The list should contain 10 items (cities), or less if there are less completions. The list should be displayed in a nice way (for example, in a drop-down box). The list of completions must be obtained by communication with your server from items 1 and 2 above. Take care that the host name and port are obtained automatically (from the URL) by your JavaScript, and not hard-coded anywhere.

6. When you select one of the cities from the list, something related to that city should happen in your web app. It's up to you.

7. Make sure that your web app looks good and feels nice.

Important notice: if you find an error in the TIP file (which nobody has found before), you will be awarded a pack of Bahlsen ABC for building a tasty inverted index at home.

Add your code to a new sub-directory *sheet-06* of your folder in the course SVN, and commit it. Make sure that *compile*, *test*, and *checkstyle* run through without errors on Jenkins. Confirm that by now your hatred against checkstyle has turned into unconditional love. Also commit the usual *experiences.txt* with your much appreciated brief and concise feedback.

Which objective function do life forms in our universe optimize?