Information Retrieval

WS 2017 / 2018

Lecture 6, Tuesday November 28th, 2017

(Web applications, Part 1)

Prof. Dr. Hannah Bast
Chair of Algorithms and Data Structures
Department of Computer Science
University of Freiburg

Overview of this lecture

- Organizational
  - Your experiences with ES5 fuzzy prefix search
- Contents
  - How to build a search web application (with live coding)

    Sockets create, accept, receive, send, close

    Hypertext HTTP, Mime types, HTML, CSS
  - ES6: build a web application that displays fuzzy prefix matches for the user query (using your code from ES5)

    For ES6, you will build a simple static application, for ES7 you will make it dynamic; you will be amazed by the result

Experiences with ES5 1/2

- Summary / excerpts
  - Fuzzy search is cool and really useful for searching
  - Relatively time-intensive for many, however, not because of the difficulty, but because of stupid bugs

    Reason 1: many of you still lack basic programming practice

    Reason 2: lack of concentration: make sure that you are well-rested and fresh when you start these exercises
  - Minor changes in the dataset and TIP file on Wednesday

    There was an announcement on the forum, but it seems that many of you do not follow the forum / are not subscribed

    Please subscribe to the forum; we work really hard on the exercise sheets, but mistakes now and then are unavoidable

Experiences with ES5 2/2

- Results
  - Percentage of entity names, for which PED was computed

the	1.71%	very ambiguous query
breib	0.95%	fairly ambiguous query
the BIG lebauski	0.07%	more specific query

  - The best query times are < 50 ms, even for the hardest query
  - Java vs. C++: a lot of variation in your running times

    Some Java codes are faster than some of the C++ codes, but the fastest query times are achieved with C++ code
  - Memory consumption: 4 GB for Java vs. 1.6 Gb for C++

    Java's is wasteful with space, with little user control

Socket Communication 1/5

- Motivation
  - Two programs / processes communicating with each other, possibly (and often) on two different machines
  - For a typical web application:

    Browser asking for (static) web page ... Lecture 6 (today)

    Code in web page asking for (dynamic) contents ... Lecture 7
  - Endpoint of such a communication channel is called **socket**
  - Each socket belongs to a particular machine (host) and has a unique id (port) on that machine

    The same machine can have many communication channels, hence the concept of (many) ports

Socket Communication 2/5

- High-level procedure
  - **Server side:**

    Create a socket and bind it to a given port

    Listen on that port for incoming requests

    Read request, compute result, send result
  - **Client side:**

    Connect to socket on server (need machine name + port)

    OS automatically assigns unique port on client machine

    Send request, wait for result

Socket Communication 3/5

- Implementation, server side
  - All programming languages have standard libraries for convenient socket communication (for server and client)

Python	socket	
Java	java.net.ServerSocket	
C++	boost::asio	(asio = asynchronous IO)

  - We provide code for the server socket communication on the Wiki, in both Java and C++

    Since for ES6 you have to integrate your solution from ES5, Python is not a meaningful option for this ES6

  - Let's now live-code a simple server in Java ...

Socket Communication 4/5

- Implementation, server side, Java
  - Create socket, wait for request, get request, send result

```
ServerSocket  server = new ServerSocket(port);

Socket client = server.accept();
client.setSoTimeout(1000);                    // Read timeout.

auto input  = new BufferedReader(new InputStreamReader(client.getInputStream()));
auto output = new DataOutputStream(client.getOutputStream());

while (...) { ... input.readLine() ... }      // Read string.

output.write("...".getBytes("UTF-8"));        // Write string.
output.write(... response bytes ...);         // Write bytes.
```

In Java, strings are **not** byte arrays ... more in Lecture 7

Socket Communication 5/5

- Implementation, client side
  - For a web application, it suffices to implement the server
  - The web browser plays the role of the client
  - We can also test via simple communication programs, e.g.

    telnet <host> <port>
  - This establishes a communication channel to the given machine and port

    You can also try this when you work on ES6, to check if your basic server loop works

    Alternatively, you can enter http://<host>:<port> in the address bar of your browser and see what happens

Hypertext 1/10

- HTTP = Hypertext Transfer Protocol
  - Used by the browser to communicate with (web) server
  - The typical request looks as follows:

    ```
    GET /search.html HTTP/1.1 ...
    ```

    /search.html = part of URL after the http://<host>:port

  - The typical results is as follows:

    ```
    HTTP/1.1 200 OK
    Content-Length: 137
    Content-Type: text/html

    ... the 137 bytes of the content ...
    ```

    Note: HTTP demands that newlines are encoded as **\r\n**
    but the recommendation is that a single \n is accepted, too

Hypertext 2/10

- HTTP = Hypertext Transfer Protocol
  - There are many more request types ... for example:

    POST (instead of GET)

    For longer requests, that are not sent as part of the URL

  - And many more headers ... for example

    HTTP/1.1 404 Not found

    To indicate that the requested resource does not exist

    HTTP/1.1 403 Forbidden

    To indicate that this resource is a no no for you

    For ES6, implement both 404 and 403 (it's easy)

Hypertext 3/10

- Content Types
  - Standard names for the different types of content sent across the internet

    Also called MIME = Multipurpose Internet Mail Extensions
  - Examples

text/plain	plain text
text/html	HTML ... see slides 14 + 15
text/css	CSS ... see slide 16
image/png	PNG image
image/gif	GIF image
application/javascript	JavaScript ... Lecture 7
application/json	JSON ... Lecture 7

Hypertext 4/10

- Browser Development Console
  - Extremely useful for debugging web applications, or in general to understand better what is going on

Chrome	**F12** / Ctrl+Shift+I
Firefox	**F12** / Ctrl+Shift+I
Internet Explorer	**F12**

  - Important sections for us:

Network: requests sent and results received

Elements: elements of the HTML page ... see next slides

Console: output from the JavaScript ... see Lecture 7

Hypertext 5/10

- HTML = Hypertext Markup Language
  - Language for specifying the content of a web page
  - XML-like language, general structure:

```
<html>
  <head>
    ... meta information + includes ...
  </head>
  <body>
    ... contents of the page ...
  </body>
</html>
```

Hypertext 6/10

- HTML
  - Example tags for the <head>...</head> section:

    ```
    <link rel="stylesheet" type="text/css" href="..."/>
    <script src="..."></script>
    ```

    Include style information and code ... see coming slides

  - Example tags for the <body>...</body> section

<h1>...</h1>	Level-1 heading
<p> ... <p>	A paragraph of text
<input> ... </input>	Input field
<div> ... </div>	Arbitrary "logical" section

Hypertext 7/10

- CSS = Cascading Style Sheets
  - Specify style information (layout, font, color, etc) independent from the contents of the page
  - Has its own (simple) syntax ... for example, all level-1 headings in blue and boldface

    h1 { color : blue; font-weight: bold }
  - When several rules apply to same element, the "most specific" rule wins

    Hence the "cascading" ... used a lot for larger web sites

    For ES6, try to make a web page with a reasonably "nice" look and feel, using CSS

Hypertext 8/10

- Forms
  - Purpose: upon user action (typically: click on a button), load a new HTML page with new or updated information
  - Typical HTML code:

```
<form>
   <input type="text" value="xyz" name="abc"/>
   <input type="submit" value="Search"/>
</form>
```

  - The second <input .../> is shown as a (clickable) button
  - When clicking on that button, the browser will append ?abc=xyz to the URL and issue the corr. GET request

With <form action="prefix"> you can modify the URL

Hypertext 9/10

- Template variables
  - For a **SERP** (Search Engine Result Page), one wants the original page with the input augmented by the results
  - This can be realized via template variables as follows:

```
<form>
   <input type="text" value="%QUERY%" name="q"/>
   <input type="submit" value="Search"/>
   %RESULT%
</form>
```

  - The server code can then simply substitute the %...% parts depending on what was typed in the input field(s)

For ES6, replace %QUERY% by the query and %RESULT% by the result (computed via your code from ES5)

Hypertext 10/10

- URL Encoding and Decoding
  - In a URL, only a restricted character set is allowed:

    a-z A-Z 0-9 $ % / - _ . + ! * ... and a few more
  - In particular, the following are forbidden: **space**, ä, ã, â, ...
  - Using forms, the query becomes part of the URL and "forbidden" characters must be encoded differently
  - In particular, a space becomes a + and an ä becomes %C3%A4 or %E4 (why will be explained in Lecture 7)

    For ES6, you must explicitly replace all **+** in the URL by spaces again

    You can ignore URL encoding of other chars like ä for ES6, we will deal with this in Lecture 7 and ES7

References

- Relevant Wikipedia articles (in order of appearance)

http://en.wikipedia.org/wiki/Network_socket

http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

http://en.wikipedia.org/wiki/Internet_media_type

http://en.wikipedia.org/wiki/HTML

http://en.wikipedia.org/wiki/Cascading_Style_Sheets

https://en.wikipedia.org/wiki/Form_(HTML)

https://en.wikipedia.org/wiki/Web_template_system

https://en.wikipedia.org/wiki/Percent-encoding