

Information Retrieval

WS 2015 / 2016

Lecture 6, Tuesday November 24th, 2013
(How to build a web application)

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

Sockets creation and communication

Hypertext HTTP, Mime types, HTML, CSS

JavaScript DOM, AJAX, JSON, jQuery

- **Exercise Sheet 6: build a web app that displays fuzzy prefix matches (ES5) as you type your query**

Experiences with ES5 1/2

$x = bccc$
 $y = aabccc$

$\sigma_i = 0$

$$\text{PED}(x, y) = 3$$

■ Summary / excerpts

- Again, interesting exercise which many of you liked
- Some had problems understanding the algorithm
Partially, because explanation at the end of last lecture were brief, because we ran out of time, sorry for that !
- Confused, because change in code from lecture needed
- Confused, because #PED in Wiki Table same for everyone
- Confused, because normalization on sheet / in code differ
- First $|x| + 1$ columns suffice for PED computation ... **NO !**
- With intensity: <https://youtu.be/FiQnH450hPM>

Experiences with ES5 2/2

■ Results

- Improvement of q-gram based algorithm over baseline

The H \approx 3 times faster (ambiguous query)

Terinator $>$ 10 times faster (typical query)

Figct CL $>$ 2000 times faster (typical query)

- For Python: all queries unbearably slow with baseline, but feasible and often fast with q-gram based algorithm
- For Java and C++: similar situation, but baseline still bearable for a few 100K records

Search web application

■ Main components

- Server that delivers the web pages
- The contents of the web pages
- The code that runs as part of the web pages and communicates with the server that answers queries

■ Implementation

- Many technologies behind this, each quite complex
- But the basic principle behind each is easy to understand

In the following, brief motivation + example for each

Along with that we will code a toy web application **live**

■ 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 pages
 - Code in web page asking for (dynamic) contents
- 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 give 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 **all** three languages

Let's write the server code in Python together

Socket Communication 4/5

■ Implementation, server side, Python

- Create socket, bind to port, and listen

AF = Address Family

INET = Internet (IPv4)

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((socket.gethostname(), port))
server.listen(5)
```

5 = allow to accept that many requests at once

- Wait for request

```
(client, address) = server.accept()
```

- Read request and send result

8192 = read at most this many bytes

```
request= client.recv(8192).decode("ascii")
client.send(result.encode("ascii"))
client.close()
```

- Implementation, client side

- For a web application, 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>`

Establishes a communication channel to the given machine and port

- 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: 653
```

```
Content-Type: text/html
```

```
... the 653 bytes of the content ...
```

Note: HTTP demands that newlines are encoded as `\r\n`

- 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

For ES6, just implement enough to make the browser happy

■ 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 15 + 16

text/css

CSS ... see slide 17

application/javascript

JavaScript ... see slides 19 – 26

application/json

JSON ... see slide 25

application/pdf

PDF

■ 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 today and ES6:

Network: requests sent and results received

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

Console: output from the JavaScript ... see slides 18 – 26

- 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>
```

■ 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

■ 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, make some non-trivial changes to the CSS from the lecture, for a more pleasing appearance

■ Motivation

- A language that runs as part of a web page

Can do (almost) arbitrary computation

Can do (almost) arbitrary communication

Can dynamically changing the contents of the web page in response to user actions

Nowadays, there is hardly a web page anymore without JavaScript in it

■ Language features

- An object-oriented script language, with a syntax similar to Java, hence the name

Speed similar to Python, when interpreted line by line

Modern browsers perform just-in-time (JIT) compilation, in order to achieve speeds similar to Java

- Variables are untyped

```
var x = 1;           // Scalar value.  
var s = "doof";     // String.  
var a1 = [1, "doof", "doof"]; // Array (mixed types).  
var a2 = { "yes" : 5, "no" : 3 } // Associative array.
```

- DOM = Document Object Model

- Well-defined scheme for how to address elements in a web page, in particular by JavaScript code
- For example: get the contents of an element with a particular id on the web page

In the HTML:

```
<div id="result">NO RESULT YET</div>
```

In the JavaScript:

```
document.getElementById("result").innerHTML = "42";
```

- AJAX = Asynchronous JavaScript and XML
 - Old name for communication between JavaScript in browser and some server elsewhere ... typical code:

```
xhr = new XMLHttpRequest();  
xhr.onreadystatechange = function() {  
    if (xhr.readyState == 4 && xhr.status == 200) {  
        response = xhr.responseText;  
        ... process the response ...  
    }  
    xhr.open("GET", "<url>", true);  
    xhr.send();  
}
```

Much simpler with libraries like jQuery ... next slides

■ jQuery

- jQuery is a JavaScript library with convenient functions for all the common stuff ... include via

```
<script src="http://code.jquery.com/..."></script>
```

- Usage examples

```
$(document).ready(function() { ... })
```

Execute included code when HTML has fully loaded

```
$("#heading").html("Different text")
```

Change contents of element with id "heading"

■ jQuery

- Offers a much cleaner separation between static elements (HTML) and dynamic code (JavaScript)
- For example: do something after each keypress

Raw JavaScript:

HTML: `<input id="query" onkeypress="myFct()"/>`

JavaScript: `myFct() { /* ... code here ... */ }`

With jQuery:

HTML: `<input id="query">`

JavaScript: `$("#query").keypress(function() { ... })`

- jQuery, communication with server

- For example: launch GET request and do something with the result:

```
url = "http://" + host + ":" + port + "?q=" + query;
$.get(url, function(result) {
    console.log("Server replied: " + result);
    $("#result").html(result);
})
```

Note: writing to the console is quite useful for debugging

- JSON = JavaScript Object Notation
 - The result from a computation is often a complex object, e.g. an array or associative array
 - If sent as a mere string, we need code to parse that string on the JavaScript side
 - **JSON** is content in the form of ready-to-use JavaScript code ... for example:

```
{ "numVowels" : 5, "numConsonants" : 13 }
```

■ jQueryUI

- Extension of jQuery for more complex UI elements

```
<script src="https://code.jquery.com/ui/..."></script>
```

- For example, autocomplete from fixed set of strings

- HTML: `<input id="query">`

- JavaScript:

```
$("#query").autocomplete({  
    source: [ ... array of strings from  
              which to autocomplete ... ]  
});
```

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

<http://www.w3schools.com/js>

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

[http://en.wikipedia.org/wiki/Ajax_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming))

<http://jquery.com/> <http://jqueryui.com/>