Introduction to Web Science

Assignment 5

Prof. Dr. Steffen Staab
staab@uni-koblenz.de

René Pickhardt
rpickhardt@uni-koblenz.de

Korok Sengupta
koroksengupta@uni-koblenz.de

Institute of Web Science and Technologies
Department of Computer Science
University of Koblenz-Landau

Submission until: November 30, 2016, 10:00 a.m.
Tutorial on: December 2, 2016, 12:00 p.m.

Please look at the lessons 1) **Dynamic Web Content** & 2) **How big is the Web?**

For all the assignment questions that require you to write code, make sure to include the code in the answer sheet, along with a separate python file. Where screen shots are required, please add them in the answers directly and not as separate files.

Delta Group Members

Oana Dumitrasc
odumitrasc@uni-koblenz.de

Alisa Becker
alisabecker@uni-koblenz.de

Omar Aly
oaly@uni-koblenz.de

1 Creative use of the Hyptertext Transfer Protocol (10 Points)

HTTP is a request response protocol. In that spirit a client opens a TCP socket to the server, makes a request, and the server replies with a response. The server will just listen on its open socket but cannot initiate a conversation with the client on its own.

However you might have seen some interactive websites which notify you as soon as something happens on the server. An example would be Twitter. Without the need for you to refresh the page (and thus triggering a new HTTP request) they let you know that there are new tweets available for you. In this exercise we want you to make sense of that behaviour and try to reproduce it by creative use of the HTTP protocol.

Have a look at `server.py`[1] and `webclient.html` (which we provide). Extend both files in a way that after `webclient.html` is servered to the user the person controlling the server has the chance to make some input at its commandline. This input should then be send to the client and displayed automatically in the browser without requiring a reload. For that the user should not have to interact with the webpage any further.

1.1 webclient.html

```
 1: <html>
 2: <head>
 3:         <title>Abusing the HTTP protocol - Example</title>
 4: </head>
 5: <body>
 6:         <h1>Display data from the Server</h1>
 7:         The following line changes on the servers command line
 8:         input: <br>
 9:         <span id="response" style="color:red">
10:                 This will be replaced by messages from the server
11:         </span>
12: </body>
13: </html>
```

1.2 Hints:

- This exercise is more like a riddle. Try to focuse on how TCP sockets and HTTP work and how you could make use of that to achieve the expected behaviour. Once you have an idea the programming should be straight-forward.
- The Javascript code that you need for this exercise was almost completely shown in one of the videos and is available on Wikiversity.

[1] you could store the code from http://blog.wachowicz.eu/?p=256 in a file called `server.py`

- In that sense we only ask for a ”proof of concept” nothing that would be stable out in the wilde.
  - In particular, don't worry about making the server uses multithreading. It is ok to be blocking for the sake of this exercise.
- Without use of any additional libraries or AJAX framework we have been able to solve this with 19 lines of Javascript and 11 lines of Python code (we provide this information just as a way for you to estimate the complexity of the problem, don't worry about how many lines your solution uses).

Answer:

Server:

We only changed following lines in the `server.py` file in _wait_for_connections(self):

```
1:    elif (request_method == 'POST' and file_requested == "www/webclient.html")
          :
2:              serverInput = input("server input: ")
3:              conn.sendall(bytes(self._gen_headers(200) + serverInput, "UTF-8"
                    ))
4:              conn.close()
5:              continue
6:          else:
7:              print("Unknown HTTP request method:", request_method)
```

Client:

```
1: <!DOCTYPE html>
2: <html>
3: <head>
4:         <title>Abusing the HTTP protocol - Example</title>
5: </head>
6: <body>
7:     <script>
8:         var xmlHttp = new XMLHttpRequest();
9:
10:        xmlHttp.open("POST", "webclient.html", "true");
11:        xmlHttp.onreadystatechange = function(){
12:            if(xmlHttp.readyState == 4)
13:         document.getElementById("response").innerHTML = xmlHttp.responseText
14:        }
15:        xmlHttp.send(null)
16:    </script>
17:        <h1>Display data from the Server</h1>
18:        The following line changes on the servers command line
19:        input: <br>
20:        <span id="response" style="color:red">
21:                This will be replaced by messages from the server
22:        </span>
```

```
23: </body>
24: </html>
```

2 Web Crawler (10 Points)

Your task in this exercise is to "crawl" the `Simple English Wikipedia`. In order to execute this task, we provide you with a mirror of the Simple English Wikipedia at `141.26.208.82`.

You can start crawling from `http://141.26.208.82/articles/g/e/r/Germany.html` and you can use the `urllib` or `doGetRequest` function from the last week's assignment.

Given below is the strategy that you might adopt to complete this assignment:

1. Download `http://141.26.208.82/articles/g/e/r/Germany.html` and store the page on your file system.
2. Open the file in python and extract the local links. (Links within the same domain.)
3. Store the file to your file system.
4. Follow all the links and repeat steps 1 to 3.
5. Repeat step 4 until you have downloaded and saved all pages.

2.1 Hints:

- Before you start this exercise, please have a look at Exercise 3.
- Make really sure your crawler doesn't follow external urls to domains other than `http://141.26.208.82`. In that case you would start crawling the entire web
- Expect the crawler to run about 60 Minutes if you start it from the university network. From home your runtime will most certainly be even longer.
- It might be useful for you to have some output on the crawlers commandline depicting which URL is currently being fetched and how many URLs have been fetched so far and how many are currently on the queue.
- You can (but don't have to) make use of breadth-first search.
- It probably makes sense to take over the full paths from the pages of the Simple English Wikipedia and use the same folder structure when you save the html documents.
- You can (but you don't have to) speed up the crawler significantly if you use multithreading. However you should not use more than 10 threads in order for our mirror of Simple English Wikipedia to stay alive.

Answer:

```
"""
Importing libraries section
"""
import socket
from urllib.parse import urlparse
from urllib.parse import urljoin
import time
import re
import queue
import os
import csv

"""
Defining class section
"""
class crawler:
    def extractHostAndPath(url):
        result = list()
        # parsing url using urlparse library
        parsedURL = urlparse(url)
        result.append(parsedURL.path)
        result.append(parsedURL.netloc)
        return result
    def doCrawel(url):
        urlQueue = queue.Queue()
        urlQueue.put(url)
        counter = 1
        badUrlsCounter = 0
        dictOfCrawlableLinks = {url:""}
        while not urlQueue.empty():
            try:
                # create TCP socket
                sckt = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            except:
                print("Socket Creation Failed")

            underProcessingUrl = urlQueue.get()
            if counter % 500 == 0:
                print("Currently processing url: " + underProcessingUrl)

            hostAndPath = crawler.extractHostAndPath(underProcessingUrl)
            host = hostAndPath[1]
            filePath = hostAndPath[0]
            # creating server address using the server name from
            # parseURL.netloc parameter and setting the port to
            serverAdd = (host, 80)
            # creating get request using the path extracted from the given
                URL
            getRequest = "GET " + filePath + " HTTP/1.0\r\nHost: " + host + "
```

```
                \r\n\r\n"
49:             try:
50:                 # connecting to the server
51:                 sckt.connect(serverAdd)
52:             except socket.error:
53:                 print("Connection to the server failed")
54:                 badUrlsCounter += 1
55:                 if dictOfCrawlableLinks[underProcessingUrl]:
56:                     dictOfCrawlableLinks[underProcessingUrl] += "\nBad URL"
57:             try:
58:                 # sending request to the server
59:                 sckt.sendall(bytes(getRequest, "UTF-8"))
60:                 # storing received data in a variable
61:                 extractedHtml = crawler.recvFullResponse(sckt)
62:                 extractedBody = crawler.getHeaderAndBody(extractedHtml,
                        filePath)
63:                 if extractedBody == -1:
64:                     badUrlsCounter += 1
65:                     if dictOfCrawlableLinks[underProcessingUrl]:
66:                         dictOfCrawlableLinks[underProcessingUrl] += "\nBad
                            URL"
67:                     pass
68:                 else:
69:                     links = crawler.getAllUrls(extractedBody)
70:                     fullLinks = crawler.absLinkToFullLinks(links,
                        underProcessingUrl)
71:                     dictOfMarkedLinks = crawler.markExternalLinks(fullLinks,
                        host)
72:                     for link in dictOfMarkedLinks:
73:                         if not (link in dictOfCrawlableLinks):
74:                             dictOfCrawlableLinks[link] = dictOfMarkedLinks[
                                link]
75:                     for crawlableLink in dictOfCrawlableLinks:
76:                         if dictOfCrawlableLinks[crawlableLink] == "":
77:                             urlQueue.put(crawlableLink)
78:                             dictOfCrawlableLinks[crawlableLink] = "enqueued"
79:                 counter +=1
80:                 if dictOfCrawlableLinks[underProcessingUrl]:
81:                     dictOfCrawlableLinks[underProcessingUrl] += "\
                        nSuccessfully processed URL"
82:             except:
83:                 print("Sending Message Failed")
84:                 badUrlsCounter += 1
85:                 if dictOfCrawlableLinks[underProcessingUrl]:
86:                     dictOfCrawlableLinks[underProcessingUrl] += "\nBad URL"
87:             # closing socket
88:             sckt.close()
89:
90:             if counter % 500 == 0:
```

```
91:                 print("Number of successfully processed URLs till now: "+ str
                        (counter))
92:                 print("Number of bad URLs till now: "+ str(badUrlsCounter))
93:                 print("Current URLs Queue length: "+ str(urlQueue.qsize()))
94:
95:             logFile = open("downloadedPages/logfile.txt", "a")
96:             logLine = "Number of successfully processed URLs till now: "+ str
                    (counter) + "\r"
97:             logLine += "Number of bad URLs till now: "+ str(badUrlsCounter) +
                     "\r"
98:             logLine += "Current URLs Queue length: "+ str(urlQueue.qsize()) +
                     "\r\n"
99:             logFile.write(logLine)
100:
101:        with open('downloadedPages/dict.csv', 'w', newline='') as csv_file:
102:            writer = csv.writer(csv_file)
103:            for key, value in dictOfCrawlableLinks.items():
104:                writer.writerow([key, value])
105:
106:    def markExternalLinks(listOfLinks, mainHost):
107:        linksDictionary = {listOfLinks[i]: "" for i in range(0, len(
                listOfLinks))}
108:        for link in linksDictionary:
109:            if urlparse(link).netloc != mainHost:
110:                linksDictionary[link] = "External Link"
111:
112:        return linksDictionary
113:
114:    def absLinkToFullLinks(dictOfLinks, mainUrl):
115:        for i in range(0, len(dictOfLinks)):
116:            if dictOfLinks[i][0:4] != "http":
117:                newLink = urljoin(mainUrl, dictOfLinks[i])
118:                dictOfLinks[i] = newLink
119:        return dictOfLinks
120:
121:    def getAllUrls(htmlPage):
122:        urls = re.findall(r'href=[\'"]?([^\'" >]+)', str(htmlPage))
123:        return urls
124:
125:    # creating function to keep socket open till we receive all data
126:    # or connection timeout(default 3 seconds)
127:    def recvFullResponse(socket, timeout=3.0):
128:        # using non blocking sockets
129:        socket.setblocking(0)
130:        # variables to hold our partial and final completed data
131:        # received from the server
132:        allData = list()
133:        partialData = ''
134:        # begin counting time for timeout
```

```
135:         startTime = time.time()
136:         # variable to determine how much data we want to receive
137:         bytesSize = 1024
138:         while True:
139:             # we don't need to wait if we didn't receive any further data
140:             # than what we already received
141:             if allData and time.time() - startTime > timeout:
142:                 break
143:             # wait double timeout if we didn't get any data at all
144:                 # then break with a readable message to the user
145:             elif time.time() - startTime > timeout * 2:
146:                 return "Time out and No Data Recieved"
147:             else:
148:                 # try catch block
149:                 # trying to receive data if nothing received we just do
150:                 # another loop till the server responded with some data
151:                 try:
152:                     # receiving data
153:                     partialData = socket.recv(bytesSize)
154:                     # check if the get request result status is 200 OK
155:                     # the bytesSize variable is used to not recheck
156:                     # for each and every next part of the data
157:                     if bytesSize == 1024 and (str(partialData)).find("200 OK"
                            ) < 0:
158:                         return ''.join(partialData)
159:                     else:
160:                         bytesSize = 4096
161:                         if partialData:
162:                             # append new data to already received data
163:                             allData.append(partialData)
164:                             # for sure reset start time if we get data
165:                             # to be sure we didn't get timeout before
166:                             # data receiving completion
167:                             startTime = time.time()
168:                             # wait for a while before doing another loop
169:                         else:
170:                             time.sleep(0.1)
171:                 except:
172:                     pass
173:                 # use join with empty byte to return data
174:                 # in a one byte sequence, for later extracting header
                        purposes
175:         return b''.join(allData)
176:
177:     # method to create a dictionary for header content from header string
178:     def createHeaderDict(header):
179:         headerDict = dict()
180:         headerLines = header.split("\r\n")
181:         for i in range(1, len(headerLines)):
```

```
182:             keyValue = headerLines[i].split(": ")
183:             headerDict[keyValue[0]] = keyValue[1]
184:         return headerDict
185:     # function to separate header from body & print header to console
186:     # also save body in the appropriate file format according to the
187:     # content type value of the header
188:     def getHeaderAndBody(receivedData, fPath):
189:         if (str(receivedData)).find("200 OK") >= 0:
190:             try:
191:
192:                 # find the starting part of body
193:                 startOfBody = receivedData.find(b"\r\n\r\n")
194:                 # creating the header part
195:                 header = receivedData[:startOfBody]
196:                 # creating the body part
197:                 # the constant 4 is just to escape the \r\n\r\n characters
198:                 body = receivedData[(startOfBody + 4):]
199:                 # creating the header dictionary
200:                 # to know the content type
201:                 headerDictionary = crawler.createHeaderDict(header.decode("
                        utf-8"))
202:                 # extracting the content type of the body
203:                 contentType = headerDictionary["Content-Type"]
204:                 # getting the image name by splitting on the url path
205:                 splittedPath = fPath.split("/")
206:                 fileName = splittedPath[-1]
207:                 directoryPath = '/'.join(splittedPath[0:-1])
208:                 # if the HTTP status is 200 OK we process header & body
209:                 # else we just print out the header
210:                 if not os.path.exists("downloadedPages"):
211:                     os.makedirs("downloadedPages")
212:                 if not os.path.exists("downloadedPages"+directoryPath):
213:                     os.makedirs("downloadedPages"+directoryPath)
214:                 #if the file name doesn't have extension we add it
215:                 #using the content type value
216:                 content = (((contentType.split(";"))[0]).split("/"))[1]
217:                 fileExtension = "" if fileName.find(".") >= 0 else content
218:                 # writing to the body file
219:                 # which will be created if it doesn't exist
220:                 # using 'w' to write string to the file
221:                 bodyFile = open("downloadedPages"+directoryPath+"/"+fileName
                        +"."+ fileExtension, 'wb')
222:                 bodyFile.write(body)
223:                 return body
224:             except:
225:                 return -1
226:         else:
227:             return -1
228:
```

```
229: """
230: Main Section
231: """
232: startTime = time.time()
233: crawler.doCrawel("http://141.26.208.82/articles/g/e/r/Germany.html")
234: endTime = time.time()
235:
236: totalTime = endTime - startTime
237: print("Crawling time: " + str(totalTime))
```

3 Web Crawl Statistics (10 Points)

If you have successfully completed the first exercise of this assignment, then please provide the following details. You may have to tweak your code in the above exercise for some of the results.

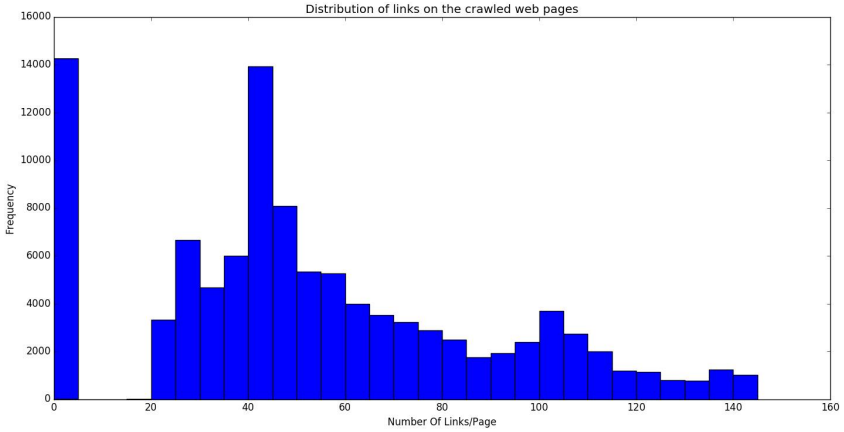
3.1 Phase I

1. Total Number of *webpages* you found.
2. Total number of links that you encountered in the complete process of crawling.
3. Average and median number of links per web page.
4. Create a *histogram* showing the distribution of links on the crawled web pages. You can use a bin size of 5 and scale the axis from 0-150.

Answer Phase I:

```
 1: import os
 2: import re
 3: import numpy
 4: import matplotlib.pyplot as plt
 5:
 6: def getAllUrls(file):
 7:     urls = re.findall(r'href=[\'"]?([^\'" >]+)', str(file))
 8:     return len(urls)
 9:
10: numOfPages = 0
11: resDict = dict()
12: for root, dirs, files in os.walk(".\simple"):
13:     for name in files:
14:         numOfPages += 1
15:         fileToRead = open(os.path.join(root, name), "rb")
16:         resDict[name] = getAllUrls(fileToRead.read())
17:
18: print("Total Number Of Pages: ", numOfPages)
19: totalNumLinks =  sum(resDict.values())
20: print("Total Number Of Links: ", totalNumLinks)
21: avgNumLinksPerPage = numpy.mean(list(resDict.values()))
22: print("Average Number Of Links/Page: ", avgNumLinksPerPage)
23: medianOfNumLinksPerPage = numpy.median(numpy.array(list(resDict.values())))
24: print("Median Number Of Links/Page: ", medianOfNumLinksPerPage)
25:
26:
27: plt.hist(list(resDict.values()), bins=range(0, 150, 5))
```

```
28: plt.title("Distribution of links on the crawled web pages")
29: plt.xlabel("Number Of Links/Page")
30: plt.ylabel("Frequency")
31: plt.show()
```



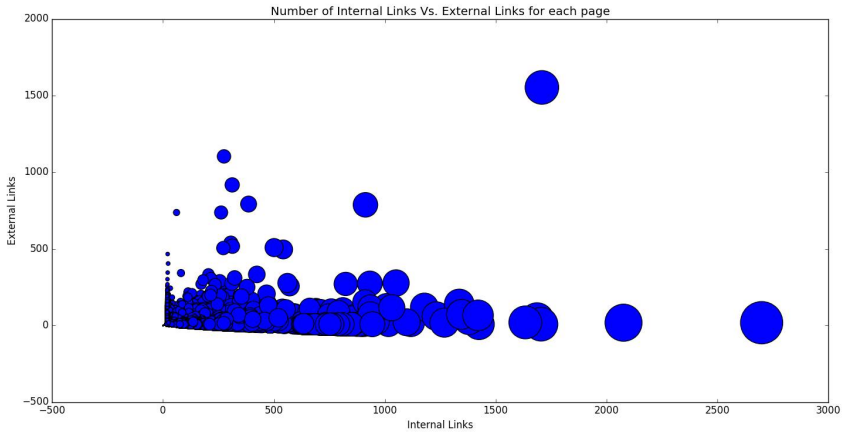
3.2 Phase II

1. For every page that you have downloaded, count the number of internal links and external links.
2. Provide a *scatter plot* with number of internal links on the X axis and number of external links on the Y axis.

Answer Phase II:

```
 1: import os
 2: import re
 3: from urllib.parse import urlparse
 4: import matplotlib.pyplot as plt
 5:
 6: def getAllUrls(file):
 7:     urls = re.findall(r'href=[\'"]?([^\'" >]+)', str(file))
 8:     return urls
 9:
10:
11: extDict = dict()
12: intDict = dict()
13: extlink = []
14: intlink = []
15: for root, dirs, files in os.walk(".\simple"):
```

```
16:     for name in files:
17:         extLinkCount = 0
18:         intLinkCount = 0
19:         fileToRead = open(os.path.join(root, name), "rb")
20:         urllinks = getAllUrls(fileToRead.read())
21:         for url in urllinks:
22:             if  (urlparse(url).netloc.find("simple.wikipedia.org") is -1) and
                     (urlparse(url).netloc.find("141.26.208.82") is -1 ) and not
                    url[0] ==".":
23:                 extlink.append(url)
24:                 extLinkCount += 1
25:             else:
26:                 intlink.append(url)
27:                 intLinkCount += 1
28:
29:         extDict[name] = extLinkCount
30:         intDict[name] = intLinkCount
31:
32: plt.scatter(list(intDict.values()), list(extDict.values()), s=list(intDict.
      values()))
33: plt.title("Number of Internal Links Vs. External Links for each page")
34: plt.xlabel("Internal Links")
35: plt.ylabel("External Links")
36: plt.show()
```



Important Notes

Submission

- Solutions have to be checked into the github repository. Use the directory name `groupname/assignment5/` in your group's repository.
- The name of the group and the names of all participating students must be listed on each submission.
- Solution format: all solutions as *one* PDF document. Programming code has to be submitted as Python code to the github repository. Upload *all* `.py` files of your program! Use `UTF-8` as the file encoding. *Other encodings will not be taken into account!*
- Check that your code compiles without errors.
- Make sure your code is formatted to be easy to read.
  - Make sure you code has consistent indentation.
  - Make sure you comment and document your code adequately in English.
  - Choose consistent and intuitive names for your identifiers.
- Do *not* use any accents, spaces or special characters in your filenames.

Acknowledgment

This latex template was created by Lukas Schmelzeisen for the tutorials of "Web Information Retrieval".

LaTeX

Currently the code can only be build using LuaLaTeX, so make sure you have that installed. If on Overleaf, there's an error, go to settings and change the LaTeXengine to `LuaLaTeX`.