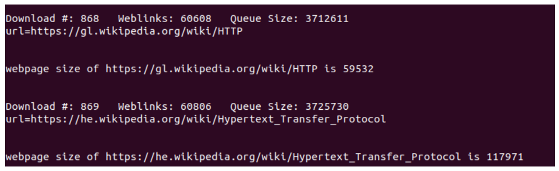
MIR Webquery

Ruben van der Waal
S1559451

March 2018

Assignment 1

For this assignment we were asked to implement a form of duplicate detection. Different levels of difficulty were available.
I opted to implement the low-medium difficulty option and organize the links in a binary tree. The binary tree uses the MD5 hashed url for faster look ups. If the url is not present in the tree it is added to the queue. The program now easily performs 300+ downloads, as opposed to last assignment. See Figure **??**.



```
Download #: 868   Weblinks: 60608   Queue Size: 3712611
url=https://gl.wikipedia.org/wiki/HTTP


webpage size of https://gl.wikipedia.org/wiki/HTTP is 59532


Download #: 869   Weblinks: 60806   Queue Size: 3725730
url=https://he.wikipedia.org/wiki/Hypertext_Transfer_Protocol


webpage size of https://he.wikipedia.org/wiki/Hypertext_Transfer_Protocol is 117971
```

Figure 1: Output of program assignment 1

Here the webcrawler starts at “http://www.tweakers.net” and successfully downloads 900 web pages. A memory issue still exists where the program runs out of memory.

Assignment 2

The second part of the assignment requires us to make several reverse indexes. A link index, web index, title index and page index. The link index is again done by hashing the url‘s with MD5. All indexes except for the page index mostly work as intended. The page index contains a lot of dirty indices. I was unable to correctly parse only the text sections of the page. The other issue with the indices is that they keyword files can contain url‘s more than once. This gives some issues later down the line with the ranking. Lastly I also implemented a repository for saving visited webpages. All normal keywords are converted to lower case automatically.

The next step is to implement a relevance ranking algorithm. I kept to the suggested amounts of score for each connection. An issue is that because url's occur twice in the files they can receive a double score from each keyword. The relevance ranking list is implemented as a linked list. This is definitely not the fastest implementation but a simple one. After all rankings are added to the list it is sorted with a bubblesort algorithm.
The last part of this assignment was to run the entire program on a mongoose server and format the results of the searches. The homepage can be see in Figure 2 and the result page in Figure 3. The program is dubbed: "Capibara Search".

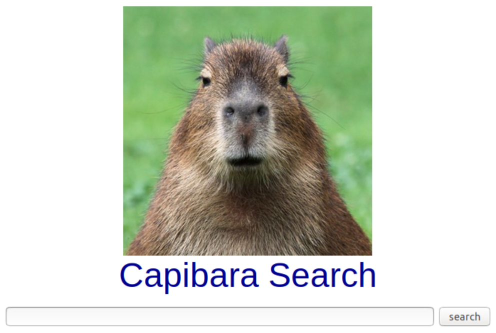
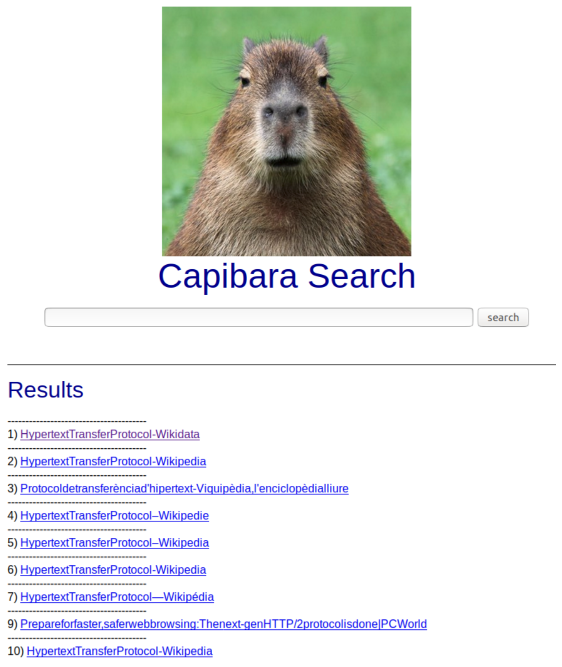


Figure 2: Homepage

On the resultpage you can see the result of the search "protocol", for each result the title of the page is displayed. Due to time constrains no short summary of the page content is given. All searches are also automatically converted to lowercase.



Results

1) HypertextTransferProtocol-Wikidata

2) HypertextTransferProtocol-Wikipedia

3) Protocoldetransferènciad'hipertext-Viquipèdia,l'enciclopèdialliure

4) HypertextTransferProtocol–Wikipedie

5) HypertextTransferProtocol–Wikipedia

6) HypertextTransferProtocol-Wikipedia

7) HypertextTransferProtocol—Wikipédia

9) Prepareforfaster,saferwebbrowsing:Thenext-genHTTP/2protocolisdone|PCWorld

10) HypertextTransferProtocol-Wikipedia

Figure 3: Resultpage

Assignment 2b

Assignment 2b was not implemented.