Exercise Sheet 3

Search Engines – Inside the black box

*Submit your solutions until **Monday, 14.03.2016, 12h00** by uploading them to ILIAS. Later submissions won't be considered. Every solution should contain the **name(s), email adress(es)** and **registration number(s)** of its (co-)editor(s).*

1 Ranking revisited (22 Points)

Consider graph G with vertices V and edges E:

$V = \{A, B, C, D\}$

$E = \{(A, B), (B, A), (B, C), (C, D), (D, B), (D, A)\}$

Thus, there is a link from A to B, from B to A, from B to C and so on.

1.1 PageRank (8 Points)

The PageRank algorithm iteratively assigns a weight to every vertice in a graph based on its incoming links:

$$PageRank(u) = \frac{(1-d)}{n} + d \sum_{(v,u)\in E} \frac{PageRank(v)}{outdegree(v)}$$

- $n = |V|$ is the number of vertices in the graph
- $v, u \in V$
- $(v, u) \in E$
- d is the damping factor describing arbitrary node visits
- outdegree(v) is the number of edges from v to other nodes

The PageRank is equally distributed before the first iteration:

Iteration 0:

PR(A) = PR(B) = PR(C) = PR(D) = 1/4

Task:

With a damping factor d = 0.8 compute the first 4 iterations and always round to three decimal places. Write down your calculations in detail for the first two iterations.

Do the values converge?

1.2 HITS – Hubs & Authorities (10 Points)

The definitions of hub and authority scores are interdependent. In the following there is an instruction of how the scores are computed including normalization:

First, we update the authority score for each node $v \in V$:

$$a(v) = \sum_{i=1}^{n} h(i)$$

where n is the total number of nodes connected to v and i is a node connected to v (inlink).

After that we compute the normalization factor $norm$:

$$norm = \sqrt{\sum_{\forall v \in V} a(v)^2}$$

Now $norm$ is used to normalize each authority value:

$$a(v) = \frac{a(v)}{norm}, \forall v \in V$$

Then we update the hub scores:

$$h(v) = \sum_{i=1}^{n} a(i)$$

where n is the total number of nodes v connects to and i is a node which v connects to (outlink).

Normalize them, too:

$$norm = \sqrt{\sum_{\forall v \in V} h(v)^2}$$

Mind that we use $h(v)$ and not $a(v)$ this time. The new $norm$ value is used for normalization again:

$$h(v) = \frac{h(v)}{norm}, \forall v \in V$$

If you are interested in another description of the algorithm, the pseudo-code on Wikipedia might help: `http://en.wikipedia.org/wiki/HITS_algorithm#Pseudocode`

In the beginning all hub and authority values equal 1:

$$\forall v \in V : h(v) = 1, a(v) = 1$$

Task:

Compute the first 4 iterations including normalization. Round to three decimal places.

Do the values converge?

1.3 (4 Points)

What are advantages of PageRank over HITS, what advantages does HITS have over PageRank?

You can find the original papers here: (Kleinberg, 1999): `http://www.cs.cornell.edu/home/kleinber/auth.pdf`
(Brin and Page, 1998): `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.2493&rep=rep1&type=pdf`

2 Programming a Simple Web Crawler (25 Points)

This is a programming exercise. You have to submit your code via ILIAS; submissions on paper will be ignored.

In this exercise you will program a simple web crawler with the following abilities:

- fetch an article page from Wikipedia
- find and store the title of the article
- find and store the first 3 links in the article text
- for each of these 3 links repeat the steps above

2.1 Preparation

Download the (compressed) Java file **minicrawler.zip** that contains the general structure of the code from ILIAS. You will need a library called *jsoup* which you can find under the following link: `http://jsoup.org/download`. Make sure that you can import it into your program. If you are using Eclipse, you can follow this description: `https://www.cs.duke.edu/courses/cps004g/fall07/assign/final/shotgun/addlibrary.html`

The following tasks are marked in the code with consecutive comments. So start at *TODO 1* to work on the first problem, then move to *TODO 2* and so on. In the beginning the program does not work because it is incomplete, but it will work soon when you fill out the missing code parts step by step. The comments in the code often provide all the information needed to understand what the task is and sometimes give hints how the solution may look like, so please read them carefully.

2.2 Program the test cases (10 Points)

First, look at the **main** function of the program and the definitions at its beginning. They are used in the **test_config** function which calls the essential functions that solve the steps mentioned above. You can find the first task in the **load_doc_from_url** function:

a) Loading a document (3 Points)

The java library jsoup can be used to load documents from the web and parse them. Look at the documentation here `http://jsoup.org/cookbook/input/load-document-from-url` and fix the return value of **load_doc_from_url**.

When you have done that the code should compile and the output looks like this[1]:

```
Crawling starts at 'https://en.wikipedia.org/wiki/Information_extraction'
Following links until depth of 2
Gathering 3 links per page
Information extraction - Wikipedia, the free encyclopedia Information
extraction From Wikipedia, the free encyclopedia Jump to: navigation, search
Information extraction (IE) is the task of automatically extracting struc
```

b) Find the article title (2 Points)

Look for the second *TODO* in **test_config** to uncomment the line that calls **get_article_title** and prints the result. *TODO 3* asks you to fix the function and return the correct article title.

HINT: Use jsoup to extract the h1 headline tag of the html source.

When you have done that the output should look like above plus the following:

```
Information extraction
```

c) Find and store the first 3 links in the article text (5 Points)

To complete the tests there is one function left to fix: **get_links** (*TODO 4* and *TODO 5*). The function gathers the first 3 links in the article text and returns them.

That means you cannot just look for the first 3 links on the web page, you have to select the relevant part of the html document and then start looking for the links. Have a look at the "Extracting data" part on `http://jsoup.org` to find out how you can select the relevant section of the document. The comments in the code may help you solve the problem.

HINT: Inspect the html source of `https://en.wikipedia.org/wiki/Information_extraction` and look for identifiers that unambiguously mark the start of the article. Remember that a parsed html document forms a tree and jsoup can select a subtree to work with.

[1] the line breaks may look different

When you have done that the code should compile and the output should look like above plus the following line:

```
[https://en.wikipedia.org/wiki/Natural_language_processing,
https://en.wikipedia.org/wiki/Machine-readable_data,
https://en.wikipedia.org/wiki/Unstructured_data]
```

2.3 Create the crawler (10 Points)

Now that you fixed the basic functions you can start crawling Wikipedia. In the **main** function *TODO 6* tells you to remove the line to allow the execution of the rest of the code. It defines a HashMap that stores the results, creates an initial queue containing the url the crawler should start with, and calls **crawl_wp**. The latter contains only comments at this point, so *TODO 7* is the task to complete the function. Again, the comments can be a guidline to the solution.

Here is an excerpt from the final output:

```
{Relational model=[https://en.wikipedia.org/wiki/Database_model,
https://en.wikipedia.org/wiki/First-order_logic,
https://en.wikipedia.org/wiki/Database], Artificial
intelligence=[https://en.wikipedia.org/wiki/Artificial_intelligence_(disambiguation)
...}
```

2.4 Questions (5 Points)

Assuming we wanted to crawl Wikipedia this way:

- which problems arise with the presented approach?
- How would you solve them? A general description is sufficient, no programming required.

2.5 Store the results using Lucene (10 Bonus Points)

The points of this exercise are bonus points. They do not influence the total amount of achivable points.

On the last exercise sheet you worked with Lucene and learned how to create a document index. The current code uses a HashMap to store the Wikipedia article titles along with links in the respective articles.

- Replace the HashMap with a Lucene Index.
- Create at least 2 test queries and report their result.

References

S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998. ISSN 0169-7552.

J.M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999. ISSN 0004-5411.