IEPS: Crawler

Avtorja: Julijan Jug, Jaka Jenko

I. Introduction

For the first project at the subject Web Information Extraction and Retrieval, we had to implement a web crawler that would visit the pages on some of the gov.si domains and gather information. Gathered information consisted of links between pages, HTML content or type of binary file and images.

II. Crawler

A. Implementation

We can divide our implementation of the crawler into four main parts. Firstly we need to initialize it, then we visit a few pages and gather information from them, from that information we create new pages and check them for duplicates and in the and we save the information into the database.

At the first step of **initialisation** we can do one of two things. Begin crawling from the beginning with the empty database or continue running crawler from the created database. If we choose to run the crawler from the beginning we firstly create sites from the provided seeds and an additional site called "OUTSIDE" where to we will save the pages that show outside of the provided sites. At that time we also check the robot.txt files on sites and save the data from them. After that, we get from sites the first pages to visit and we add them to the frontier. On the other hand, if we want to continue running the crawler from where we finished, we get the whole database. We load sites, frontier, and history, where the frontier sites are saved with html_data as "X;...", X representing the depth of the page in the frontier, so we know at what depth we are.

When the frontier is initialized we continue with **visiting web pages and gathering information from them**. The pages are visited in breadth-first search fashion. To set how many web pages we visit at once we set the number of threads. Then we assign a page to each thread and do the following things:

- We get the request data from the page
- With the help of Selenium web-driver, we render the page
- After that, we again get the request data from the page (if we compare it with the first request we can see if we were redirected)
- Then if the data is HTML we save the content of the pages and we find links and images on the page. We gather links from "href" elements end from Java-scripts "onClick" elements. The links are after that cleaned with the help of url_normalize and w3url.canonicalize_url library and we add the "/" sign in the and if needed. If the page is BINARY we just save the data type.
- Then the pages are created and checked if they belong to one of the pages from seeds and if the sites robot file allows visiting that site. At that, the images are also linked to the pages.
- As said, in the beginning, we check if there was a redirect and we also save the information which page was redirected to which url.

When we gathered all the data, links and images from multiple web sites we check for **duplicates** and manage all of the **redirects**. We check for duplicates in the frontier and in the already visited pages, where if we find that we have already had two pages with the same url we join their data and their link from/to page. We have also checked if the content of the new visited page is similar to any other already visited page with the help of the SQL similarity function. But it seems that we have set the threshold too high and we didn't find any duplicates.

In the end, we **save all of the new sites in the database**, where we also check their depth and discard the sites with the depth that is higher than the maximum set depth.

Our crawler has three possible parameters we can set (not including seed sites). Those parameters are, the number of threads (which sets how many pages are crawled at once), the timeout in seconds (the time we wait for the page to load, we have this parameter because none of the sites include crawling delay in their robot.txt file) and the max depth parameter(which limits the depth to which the crawler can go).

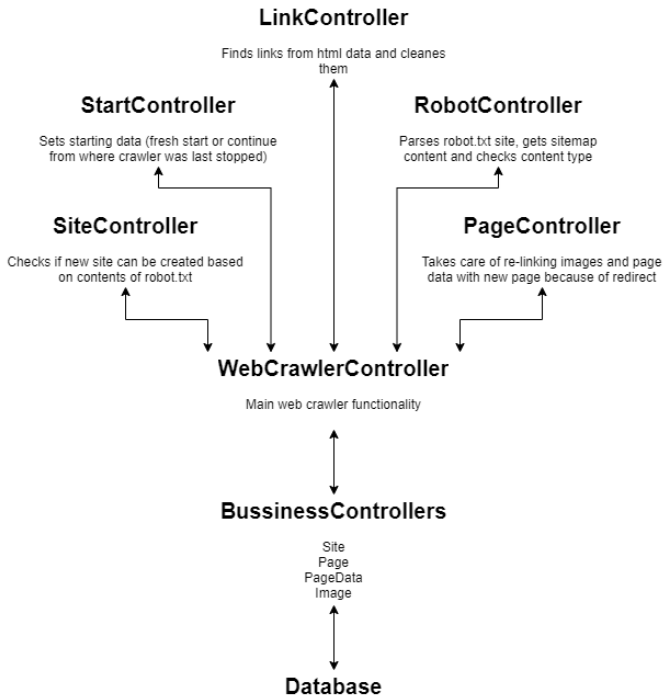

Figure 1: Crawler architecture

In figure 1 we can see the crawler architecture. We decided to split the crawler into three main parts. The main is

WebCrawlerController which takes care of connecting all of the parts of the crawler and does most of the work (visiting pages, checking for duplicates, ...). The WebCrawlerController is then being helped by another controller which takes care of smaller things, like parsing links or checking if the page can be visited. In the end, BussinessControllers take care with connecting to database and writing and reading from it.

The main **problems** we encountered were:

- problems with overwriting the links from and to pages when writing to the database,
- problems with the database memory, where we had to up the working memory from 4MB to 64MB for stable operation,
- very slow checking for duplicates when removing duplicates from the frontier and visited pages,
- problems with selenium driver, that caused random crashes and that on macOS it downloaded all the binary files, which filled the hard drive to the max and then crashed the crawler.

B. Statistics

The crawler ran for 55 hours straight using 5 threads and a slow network connection. Our crawler database contains 43.801 visited pages or 83.193 if we also include the pages in the frontier. Table I lists the number of pages in the 4 starting domains and outside domains. "e-uprava.gov.si" contains the most pages, over 40 thousand. The smallest domain is "e-prostor.gov.si" with only 632 pages.

The crawler found 14.832 different binary files. Table II presents the distribution of binary files in the starting domains and table III the distribution of file types. By far the most binary files were found on "gov.si", 33 thousand. And the most popular file type is pdf.

We found 52.322 images on the visited pages that is an average of 1.19 images per page. Table IV contains distribution of image types and table V the distribution of images in different domains. The most images were found on "evem.gov.si", 22.322 and the most popular image format is png. A little surprisingly followed by gif format with 17.148 images.

The database contains a total of 877.517 links. The distribution of links is presented in table VI. The most links are found on page "https://e-uprava.gov.si/drzava-in-druzba/javni-sektor/solstvo.html", a total of 1.156. and the most linked page is "https://e-uprava.gov.si/", with 10.001 links pointing to it.

Domain	Number of pages
e-uprava.gov.si	40.857
gov.si	33.805
e-prostor.gov.si	632
evem.gov.si	6.807
Outside	1.092

Table I: Number of pages in regards to their domains.

Domain	Number of files
e-uprava.gov.si	333
gov.si	33.805
e-prostor.gov.si	385
evem.gov.si	13.802

Table II: Distribution of binary files in domains.

File type	Number of files
pdf	7.087
doc	1.127
docx	999
ppt	8
pptx	38
xls	187
xlsx	389
zip	184
other	4.813

Table III: Distribution of file types.

File type	Number of images
PNG	28.261
JPEG	114
JPG	6.467
SVG	7
GIF	17.148

Table IV: Distribution of image file formats.

Domain	Number of images
e-uprava.gov.si	11.215
gov.si	18.061
e-prostor.gov.si	724
evem.gov.si	22.322

Table V: Distribution of images on domains.

Domain	Number of links
e-uprava.gov.si	559224
gov.si	194550
e-prostor.gov.si	6441
evem.gov.si	117302

Table VI: Distribution of links on domains.

C. Visualization

We construct the graph of all the links, using the NetworkX python library. For visualizing the graph we then used ForceAtlas2, which uses Force atlas 2 algorithm for positioning the nodes. ForceAtlas is a force-directed layout used for network spatialization.

firstly we visualized each domain separately. The most interesting visualization turned out to be "gov.si" in figure 2. We can see the most connected pages in the center and the surrounding ring of pages either in the frontier or binary pages with small in-degrees.

We also visualized the whole database, which is presented in figure 3. We see a similar pattern in this visualization with a dense center and a ring. It is a little surprising that all of the starting domains converged into one single network.

III. Conclusion

For this project, we implemented a web crawler and managed to crawl a large portion of the specified domains. We had some problems regarding the speed of the crawling and detection of duplicates (by setting too high of a threshold), that could be improved and optimized. We also taught that the crawler would crawl a bit faster but in the end, we managed to crawl over almost all of the 50.000 pages. A database error stopped us from reaching that goal at the end.

Figure 2: Gov.si link visualization.

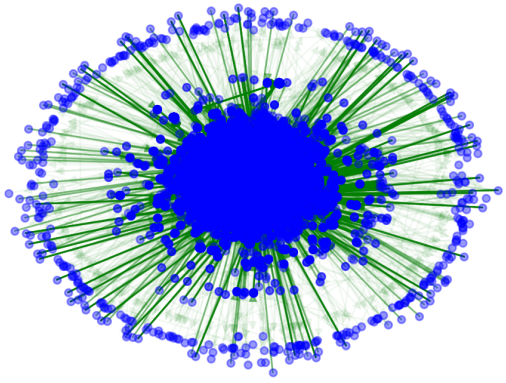

Figure 3: Visualization of the whole database.