Coding assignment 1

Web information extraction and retrieval

Jošt Gombač, Marko Krajinović, Rok Petrovčič Vižintin

1 Implementation

1.1 Database schema

We have extended the default database schema to better suit the needs of our crawler. We added the table **visited_ip** to store the IPs and the timestamps of when they were last accessed. We extended the **page** table with a *depth* column, which is the number of links traversed before arriving at a given page and *checksum* column for duplicate detection. We extended the **data_type** table with values *PPTX* and *Other*. We extended the **page_type** table with values *FRONTIER*, *CRAWLING*, *SKIP*, *ERROR* and *TIMEOUT*.

1.2 Program flow

The program starts by checking if any pages were left as *CRAWLING* from previous sessions. This means that they were not crawled to completion, so they are re-added to the frontier. Then, the seed pages get added to the frontier with a *depth* of 0. After that, crawling begins. The amount of workers is passed as a command line argument to the crawler. If left unspecified, it defaults to 6 workers.

Crawling begins by selecting the first page at a depth of 0 from the frontier, which doesn't have the type code *CRAWLING*. Each thread preferably selects currently non-crawling domain to to reduce waiting times. If no *free* domain pages are found, we select any page at current depth or increase the depth search if necessary. If a page is found successfully, it's type code is set to *CRAWLING* and it's *robots.txt* is checked. If crawl delay is not specified, it defaults to 5. The page's IP is looked up and checked in the **visited_ip** table, and the crawler waits for the necessary amount of time before crawling the page's content. If the can't be fetched due to an error, the page is skipped and given the type code *SKIP*.

Before crawling the content, a HEAD request is sent. If it times out after 5 seconds, the crawling is stopped and the page's type code is set to *TIMEOUT*. If the response's status code is over 300, the type code gets set to *ERROR*. Otherwise, the content type is checked. If it's not text/html, the page's content type gets updated to *BINARY*, but the page is not crawled further. If the response is not an error, a GET request is sent. The HTML content is checked for canonical links. If those are found, the current page gets marked as *DUPLICATE* and is removed from the frontier. Otherwise, the content, images, links and checksum are all saved in the database. All discovered links are marked with the current depth and stored in the frontier if not already present and comply with *robots.txt* rules. Checksums are compared to those in the database to check for duplicate content. Crawling continues until the program is cancelled, or the frontier is empty.

2 Problems

We have encountered several issues during development. Different threads were not respecting IP access delays, as they were not getting written correctly to the database. This was solved by using threading locks. Most other issues were also threading related. Threads dying due to bad error handling, and also memory leaks due to browser instances not getting closed properly. Those were solved by improving the error handling by using more specific catch clauses.

3 Results and visualizations

3.1 Results

Table 1: Data collected after approximately 30 hours of crawling

	Seed sites	Other sites
Pages total	34289	68213
Pages crawled	24540	25629
HTML pages	12781	11824
Duplicates	1003	1123
Errors	865	642
DOCX	487	260
PDF	4634	4630
PPT	4	24
DOC	614	541
Other binary documents	2658	2430
Images	60145	181384
Avg. num. images per page	4.59	16.7

3.2 Visualization

In order to better understand the way the crawled domains are connected, we created a visualization. The histogram shows which domains are linked from which other domains.

Figure 1: Histogram of domains by number of inbound links from other domains