Assignment 2: Web Crawler

Due Feb 17 by 11:59pm **Points** 30 **Submitting** a file upload **File Types** zip
Available Jan 27 at 12am - Feb 24 at 11:59pm 29 days

This assignment was locked Feb 24 at 11:59pm.

This assignment is to be done in groups of up to 3 students. You can use text processing code that you or any teammate wrote for the previous assignment. You cannot use crawler code written by non-group-member classmates. Use code found over the Internet at your own peril -- it may not do exactly what the assignment requests. If you do end up using code you find on the Internet, you must disclose the origin of the code. **As stated in the course policy document, concealing the origin of a piece of code is plagiarism**. Use Piazza for general questions whose answers can benefit you and everyone.

In this project, you are going to implement the core of a Web crawler, and then you are going to crawl the **following URLs (to be considered as domains for the purposes of this assignment) and paths:**

- *.ics.uci.edu/*
- *.cs.uci.edu/*
- *.informatics.uci.edu/*
- *.stat.uci.edu/*
- today.uci.edu/department/information_computer_sciences/*

As a concrete deliverable of this project, besides the code itself, you must submit a report containing answers to the following questions:

1. How many unique pages did you find? Uniqueness is established by the URL, but discarding the fragment part. So, for example, *http://www.ics.uci.edu#aaa* and *http://www.ics.uci.edu#bbb* are the same URL.
2. What is the longest page in terms of number of words? (HTML markup doesn't count as words)
3. What are the 50 most common words in the entire set of pages? (**Ignore English stop words**, which can be found, for example, **here** **(https://www.ranks.nl/stopwords)**) Submit the list of common words ordered by frequency.
4. How many subdomains did you find in the ics.uci.edu domain? Submit the list of subdomains ordered alphabetically and the number of unique pages detected in each subdomain. The content of this list should be lines containing *URL*, *number,* for example:
   http://vision.ics.uci.edu, 10 (not the actual number here)

What to submit: a zip file containing your modified crawler code and the report.

Grader meetings: this project requires a meeting of all members of your group with one of the TAs/Readers, where all of you will be asked questions about your crawler -- your code and the operation of the crawler. These meetings will occur a few days after the submission deadline. Instructions will be sent at the time.

Specifications

To get started, fork or get the crawler code from: https://github.com/Mondego/spacetime-crawler4py (https://github.com/Mondego/spacetime-crawler4py)

Read the instructions in the README.md file up to, and including, the section "Execution". This is enough to implement the simple crawler for this project. In short, this is the minimum amount of work that you need to do:

1. Install the dependencies
2. Set the USERAGENT variable in Config.ini so that it contains all students' IDs of the group members. If you fail to do this properly, your crawler will **not** exist in Prof. Lopes server's log, which will put your grade for this project at risk.
3. (**This is the meat of the crawler**) Implement the scraper function in scraper.py. The scraper function receives a URL and corresponding Web response (for example, the first one will be "http://www.ics.uci.edu" and the Web response will contain the page itself). Your task is to parse the Web response, extract enough information from the page (if it's a valid page) so to be able to answer the questions for the report, and finally, return the list of URLs "scrapped" from that page. Some important notes:
    1. **Make sure to return only URLs that are within the domains and paths mentioned above**! (see **is_valid** function in scraper.py -- you need to change it)
    2. Make sure to defragment the URLs, i.e. remove the fragment part.
    3. You can use whatever libraries make your life easier to parse things. Optional dependencies you might want to look at: lxml (nudge, nudge, wink, wink!)
    4. Optionally, in the scraper function, you can also save the URL and the web page on your local disk.
4. Run the crawler from your laptop/desktop or from an **ICS openlab machine.** Note that this will take several hours, possibly several days! It may even never end! Note that you need to be inside the campus network, or you won't be able to crawl. If your computer is outside UCI, use the VPN.
5. **Monitor what your crawler is doing**. If you see it trapped in a Web trap, or malfunctioning in any way, stop it, fix the problem in the code, and restart it. Sometimes, you may need to restart from scratch. In that case, delete the frontier file (frontier.shelve), or move it to a backup location, before restarting the crawler.

Crawler Behavior Requirements

In this project. we are looking for **text** in Web pages so that we can search it later on. The following is a list of what a "correct crawl" entails in this context:

- Honor the politeness delay for each site
- Crawl all pages with high textual information content
- Detect and avoid infinite traps
- Detect and avoid sets of similar pages with no information
- Detect and avoid dead URLs that return a 200 status but no data
- Detect and avoid crawling very large files, especially if they have low information value

For most of these requirements, the only way you can detect these problems is by first monitoring where your crawler is going, and then adjusting its behavior in order to stay away from problematic pages.

Test Period and Deployment Period

Due to the nature of this project, the time allocated to it is divided into two parts:

Test: until 12 Feb, 23h59. During this time, your crawler can make all sorts of mistakes -- try to crawl outside allowed domains, be impolite, etc. No penalties while you are figuring things out!

Deployment: from Feb 13 until Feb 17, 23h59. This is the real crawl. During this time, your crawler is expected to behave correctly. **Even if you finish your project earlier, you must operate your crawler during this time period.**

Extra credit: (+10 points)

Make the crawler multithreaded, so to crawl faster. However, your multithreaded crawler must obey the politeness rule: **two or more requests to the same domain, possibly from separate threads, must have a delay of 500ms**. This is more tricky than it seems!

In order to do the extra credit, you should read the "Architecture" section of the README.md file. Basically, to make a multithreaded crawler you will need to:

1. Reimplement the Frontier so that it's thread-safe and so that it makes politeness per domain easy to manage
2. Reimplement the Worker thread so that it's politeness-safe
3. Set the THREADCOUNT variable in Config.ini to whatever number of threads you want

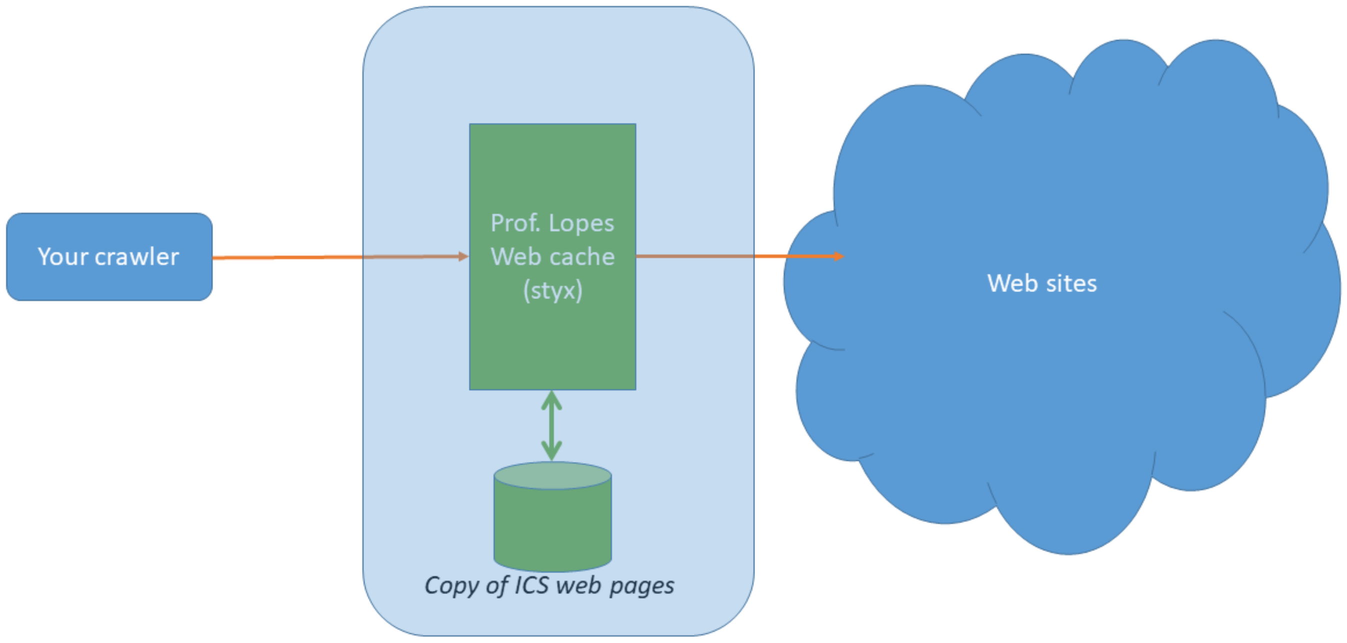
Grading criteria

1. Are the analytics in your report within the expected range? (10%)
2. Did your crawler operate correctly? -- we'll check our logs (50%)
    1. Does it exist in Prof. Lopes' Web cache server logs? (**if it's not in the ICS logs, it didn't happen: you will get 0**)
    2. Was it polite? (penalties for impolite crawlers)
    3. Did you crawl ALL domains and paths mentioned in the spec? (penalties for missing domains and paths)
    4. Did it crawl ONLY the domains and paths mentioned in the spec? (penalties for attempts to crawl outside)
    5. Did it avoid traps? (penalties for falling in traps)
    6. Did it avoid sets of pages with low information value? (penalties for crawling useless families of pages)
3. Are you able to answer the questions about your code and the operation of your crawler? (40%)

Technical Details

In order not to disrupt the ICS network, your crawlers use a Web cache that is specifically designed for this project, and that runs on one of Prof. Lopes servers. The following picture illustrates the architecture of this project:

Architecture of Web Crawling Project



If you use the crawler code properly, the cache is largely invisible to you. But you should be aware that it is there. At certain points, you may receive errors that are specifically sent by the cache server to your crawler, when your crawler is doing something it shouldn't -- they are in the 600 range of the status code. DO NOT ATTEMPT TO BYPASS THE CACHE! -- that may seriously disrupt the ICS network.