Project 2 – The Spacetime Crawler

Due Date: 05/10

This assignment is to be done in groups of 3. You can use text processing code that you or any classmate 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 the Discussion Board for general questions whose answers can benefit you and everyone.

Your crawler is part of a collaborative, distribute crawling infrastructure. The central frontier of this infrastructure is in one of Prof. Lopes' servers. The frontier does a lot of the heavy work. Your crawler will be given 5 URLs at a time, and should proceed to download and process them.

Two Project Phases

Because this is a collaborative, distributed crawler, a lot of things can go wrong with one student's crawler that will affect some other student, also between your crawler and our central frontier. As such, we will do the project in two phases:

- Phase 1: TESTING. From now until May 2nd, 8:00pm. You can do as many mistakes as needed, as you find your way through the code and the tasks you need to do. **On May 2nd, end of the day, we will reset the central component to forget about everything you sent to it.**
- Phase 2: PRODUCTION. From May 3rd 08:00am until May 10 11:59pm. During this second phase I expect your code to be ready to run without a lot of problems, so we will do the real, final crawl.

Implementing your Project

Step 1 Getting the project

git clone https://github.com/Mondego/spacetime-crawler

Step 2 Installing the dependencies

pip install spacetime (use admin mode if needed. If admin mode is not possible, install using the --user flag)

Step 3 Writing the required functions and parameters

You must set this correctly to get credit for the project. If we can't trace your crawler in our logs, it's equivalent to you not doing the project.

1. **UserAgentString**: (applications/search/crawler_frame.py, L34)

   The user agent string should be of the form "IR S17 UnderGrad student_id1, student_id2, …" EG: "IR S17 UnderGrad 12345678, 23156523, 12343545"

2. **app_id** (applications/search/crawler_frame.py, L31)

   The app_id is a string containing a list of the team student ids demarcated by underscores EG: "12345678_23156523_12343545"

3. **extract_next_links** (applications/search/crawler_frame.py, L83)

   This function extracts links from the content of a downloaded webpage.

   Input: [raw_content_obj1, raw_content_obj2, ….]
   Output: list of URLs in string form. Each URL should be in **absolute form**. It is not required to remove duplicates that have already been downloaded. The frontier takes care of that.

   Note: raw_content_obj1, raw_content_url2 are objects of Type *"`UrlResponse`"* declared at L27-41 datamodel/search/datamodel.py
   Each object contains information that can be used to make an informed decision for link extraction. The detailed description of each field is given below:

   - url: The source link given by the frontier.
   - content: The raw_content that was downloaded by the crawler client using the given URL.
   - error_message: The HTTP error message/ custom error message *in case of failure*.
   - headers: The HTTP response headers returned on download.
   - http_code: The HTTP response code sent by the server on download.
   - is_redirected: Boolean that declares if there were redirects when downloading this URL. (True if the URL was redirected, False if not)
   - final_url: In case of redirects (i.e. is_redirected = True), the final URL where the resource was found. If there were no redirects, this URL is left as None.

   We understand that you may receive certain URLs that you may think should not be crawled, in such situation we give you the option of reporting it to our servers as well. Objects can be updated for the following fields in extract_next_links function:

   - bad_url: if you want to vote to ban this URL from entering the frontier, mark this as True. If sufficient crawlers mark URLs as bad, patterns corresponding to the URLs are banned.

4. **is_valid** (applications/search/crawler_frame.py, L97)

   This function returns True or False based on whether a URL is valid and must be downloaded or not.

   Input: URL is the URL of a web page in string form

   Output: True if URL is valid, False if the URL otherwise. Robot rules and duplication rules are checked separately and *should not be checked here*. This is a place to

   1. filter out crawler traps (the ICS calendar)
   2. double check that the URL is valid and in absolute form.

   Returning False on bad urls can contribute to banning such patterns.

Step 4 Running the crawler

Execute the command

```
python application/search/crawler.py -a amazon.ics.uci.edu –p 9050
```

Analytics

Along with crawling the web pages, your crawler should keep track of some information, and write it out to a file at the end. Specifically, it should:

1. Keep track of all the subdomains that it visited, and count how many different URLs it has processed from each of those subdomains
2. Count how many invalid links it received from the frontier, if any
3. Find the page with the most out links (of all pages given to your crawler)
4. Any additional things you may find interesting to keep track

At the end, it should write this information out to a file.

Notes

a) Your crawler will stop crawling when it reaches 3,000 successful URL downloads. These URLs are written to a file created on the same folder you run the crawler, and it's called **successful_URLs.txt**. If you want to do more crawling than this, simply stop the crawler, remove, or move, that file out of that folder and start the crawler again.
b) You can start and stop your crawler multiple times without having to remove the successful_URLs.txt file. The number of URLs processed will accumulate between sessions.
c) You must run the crawler inside the campus network (or VPN). You can use either your laptop or one of the openlab machines.
d) Python 2.7 is required

Submitting Your Assignment

Your submission should be a single zip file containing the spacetime code, including your additions to crawler_frame.py, as well as the text file with the analytics mentioned above. Only one member per team should submit the file on canvas.

Grading Process

You will meet with the grader for about 10-15 minutes during the week starting on 5/11. During that meeting, be ready for the following:

1. Show your program running on your own computer
2. Answer questions about your program (as submitted) and its behavior during the meeting

All the members of the group need to be present for F2F Demo. You should **not** continue to work on your program once you submit it to Canvas, as the TA's will be looking at your submitted code.

Evaluation Criteria

Your assignment will be graded on the following criteria's.

1. Correctness (50%)
   a) Did you successfully download at least 3,000 pages?
   b) Did you extract the links correctly?
   c) Does your crawler validate the URLs that is given and that is sends back to the frontier?
   d) How does your program handle bad URLs?
   e) Does your crawler avoid traps?

2. Understanding (50%)
   a) Do you demonstrate understanding of your code?