Project 2 – The Spacetime Crawler

Due Date: 2/8/2018 11:59PM

This assignment is to be done in groups of up to 3. You can use text-processing code that you or any classmate **in your team** 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 on Piazza for general questions whose answers can benefit you and everyone.

Your crawler is standalone but shares data with the rest of the crawlers in the course. Each crawler has their own frontier in a server in ICS, and can manage this frontier. The frontier does a lot of the heavy work. Your crawler will be given 1 URL at a time and should proceed to download and process it.

Implementing your Project

Step 1 Getting the project

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

Step 2 Installing the dependencies

Make sure you do not have conflicting libraries by issuing the command.
python -m spacetime -v

You should see the following output"
Spacetime Version is 2.0
Rtypes Version is 2.0

If the outputs do not match, or if it returns an error "unrecognized argument: version", please uninstall the old spacetime, and rtypes by issuing the commands.
python -m pip uninstall spacetime
python -m pip uninstall rtypes

Get the latest repository of spacetime-crawler to get the latest version of spacetime and rtypes. Both packages are included with the assignment.

Step 3 Writing the required classes, 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. **Write out the details of your teammates and you in the team.txt file:** The details are a comma separated list of your UCInetID, and student number. Each team member is written in a new line. EG:
   panteater,12345678
   peterant,87654321
2. **Run generate_crawler_application.py:** It will generate files in two folders: applications/search and datamodel/search with the crawler code in them customized by the details in team.txt. If the details

are wrong, or need to be changed, please regenerate using this script after correcting the team.txt file. Remember, lines starting with the # symbol are ignored (Act as comments).

3. **Define function extract_next_links** (applications/search/crawler_frame.py)

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

   Save the details that have been downloaded in your own machine.

   Input: raw_content_obj
   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 ignoring duplicates.

   Note: raw_content_obj is an object of Type *"`UrlResponse`"* declared at the beginning of datamodel/search/server_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.

4. **Define function is_valid** (applications/search/crawler_frame.py)

   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 (e.g. the ICS calendar, dynamic URL's, etc.) You will need to do some research online or apply concepts regarding crawler traps covered in class.
   2. Double-check that the URL is valid and in absolute form.

   Returning False on a URL does not let that URL to enter your frontier. Some part of the function has already been implemented. It is your job to figure out how to add to the existing logic in order to avoid crawler traps and ensuring that only valid links are sent to the frontier.

Step 4 Running the crawler

Execute the following commands

To run the crawler.

```
python applications/search/crawler.py -a amazon.ics.uci.edu -p 9400
```

To check frontier status.

```
python applications/search/check_frontier.py -a amazon.ics.uci.edu -p 9400
```

To reset frontier progress.

```
python applications/search/reset_frontier.py -a amazon.ics.uci.edu -p 9400
```

To clean only those urls from frontier that no longer satisfy your is_valid function.

```
python applications/search/delete_invalids_from_frontier.py
-a amazon.ics.uci.edu -p 9400
```

Note that resetting the frontier may take a while depending on how big the frontier is.

Step 5 Monitoring the crawler

Crawlers do not automatically filter for bad urls. You must build the is_valid function as explained in Step 3, point 4. However, to help you along the way, we have implemented a way for you to see the bad links sent to the server. They can be seen at

http://amazon.ics.uci.edu:9400/<Your_crawler_id>/invalid

This list is never deleted, however, every restart of the crawler is logged as well. The epoch timestamp when each bad url was received is written at the start. You can use the epoch converter to convert it to human readable time.

This is a basic check, and does not filter all crawler traps. If there are more crawler traps that you detect, please feel free to block them in your is_valid function.

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 the subdomains that it visited, and count how many different URLs it has processed from each of those subdomains.
2. Find the page with the most out links (of all pages given to your crawler). Out Links are the number of links that are present on a particular webpage.
3. Any additional things you may find interesting to keep track of, such as, invalid links, crawler traps encountered, and so on. But this is not mandatory.

Analytics should be saved as file(s). These analytics need to be submitted along with your project code.

Notes

a) In order for your crawler to work, make sure you implement the "extract_next_links" function. This function is not implemented and it's the starting point of your assignment. Without extracting links from the first page obtained from the server, the crawler cannot progress.
b) You can start and stop your crawler multiple times and progress will be saved. To reset the progress, please use the reset_frontier method provided. Resetting the frontier clears all of your progress. Try and avoid resetting the frontier if you've crawled the right number of webpages.
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.
e) Optional dependencies you might want to use: lxml
f) Please do not use direct concatenation for making urls absolute in "extract_next_links". Use a pre built library function like the ones found in lxml. There are many cases that cannot be handled by a direct concatenation and can cause at the best case the crawler to send invalid links. At the worst, it can cause the crawler to enter a crawler trap.
g) You can stop your crawler when you have crawled 3000 links

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 15-20 minutes during the week starting on 02/11. During that meeting, be ready for the following:

1. You may be asked to run your crawler on your own machine so please carry a laptop.
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 extract the links correctly and in absolute form?
   b) Does your crawler validate the URLs that is given and that is sends back to the frontier?
   c) How does your program handle bad URLs?
   d) Does your crawler avoid traps?

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