Project 2 – The Crawler
Due 1/27

Implementing your Project

Step 1 Getting the project

See Assignment Description Page

Step 2 Installing the dependencies

Use python3 for this project. You will need to install cbor package (# python -m pip install cbor) to use the project skeleton. Also, you may want to install third-party libraries like lxml later.

Step 3 Writing the required classes, functions, and parameters

1. **Before doing the next steps, take your time to go through main.py, frontier.py, corpus.py and crawler.py files to get an overall idea about how the project skeleton works.**

- **frontier.py**: This file acts as a representation of a frontier. It has method to add a URL to the frontier, get the next URL and check if the frontier has any more URLs. Additionally, it has methods to save the current state of the frontier and load existing state
- **crawler.py**: This file is responsible for scraping URLs from the next available link in frontier and adding the scraped links back to the frontier
- **corpus.py:** This file is responsible for handling corpus related functionalities like mapping a URL to its local file name and fetching a file meta-data and content from corpus. In order to make it possible to work on a crawler without accessing the ICS network, this file accesses a static corpus and maps given URLs to local file names that contain the content of that URL.
- **main.py:** This file glues everything together and is the starting point of the program. It instantiates the frontier and the crawler and starts the crawling process. It also registers a shutdown hook to save the current frontier state in case of an error or receiving of a shutdown signal.

2. Define **functionextract_next_links(**crawler.py)
This function extracts links from the content of a fetched webpage.

> **Input**: url_data which is a dictionary containing the content and required meta-data for a downloaded webpage. Following is the description for each key in the dictionary:
>
> url: the requested url to be downloaded
> content: the content of the downloaded url in binary format. None if url does not exist in the corpus
> size: the size of the downloaded content in bytes. 0 if url does not exist in the corpus
>
> content_type: Content-Type from the response http headers. None if the url does not exist in the corpus or content-type wasn't provided
>
> http_code: the response http status code. 404 if the url does not exist in the corpus
> is_redirected: a boolean indicating if redirection has happened to get the final response
>
> final_url: the final url after all of the redirections. None if there was no redirection.

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 fetched. The frontier takes care of ignoring duplicates.

3. Define **functionis_valid**(crawler.py)
This function returns True or False based on whether a URL is valid and must be fetched 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. This is a great place to filter out crawler traps. Duplicated urls will be taken care of by frontier. You don't need to check for duplication in this method

Filter out crawler traps (e.g. the ICS calendar, dynamic URL's, etc.), Additionally crawler traps include history based trap detection where based on your practice runs you will determine if there are sites that you have crawled that are traps, continuously repeating sub-directories and very long URLs. You will need to do some research online but you will provide information on the type of trap detection you implemented and why you implemented it that way.(DO NOT HARD CODE URLS YOU THINK ARE TRAPS, ie regex urls, YOU SHOULD USE LOGIC TO FILTER THEM OUT)

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

You need to first download the static corpus. Extract the compressed file somewhere in your disk.

To run the project, simply run:
python3 main.py [CORPUS_DIR]
Replace [CORPUS_DIR] with the address of the corpus on disk, e.g.
/home/user/corpus/spacetime_crawler_data

Please note that getting out of the program because of an error or shutdown signal (e.g. through CTRL+C) will save the current state of the frontier in the `frontier_state` directory. To clean the current state and start from the beginning, simply delete that directory.

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 valid out links (of all pages given to your crawler). Out Links are the number of links that are present on a particular webpage.
3. List of downloaded URLs and identified traps.
4. What is the longest page in terms of number of words? (HTML markup doesn't count as words)
5. What are the 50 most common words in the entire set of pages? (Ignore English stop words, which can be found, (https://www.ranks.nl/stopwords)

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 delete the `frontier_state` directory. Resetting the frontier clears all of your progress.

c) Python 3.6+ is required.

d) Optional dependencies you might want to use: lxml

e) 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.

Evaluation Criteria

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

a) Do 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?

e) Are you collecting stats in your code to produce items 1,2 and 3 of the analytics?

Project 2 FAQS:

1. Do I have to implement/modify any function outside the two functions?

-- No

2. Can I write one or more helper methods?

-- Yes.

4. Are all dynamic URLs trap?

-- Not necessarily. For example,
https://www.ics.uci.edu/community/news/view_news?id=1473 is not a trap.

5. Is every URL which contains the word 'calendar' a trap?

-- No. For example, "https://www.reg.uci.edu/calendars/quarterly/2018-2019/quarterly18-19.html" is not a trap.

6. So how do I know which URL is a trap?
-- Do your own research from internet and class materials. Typically, if a URL growths in length everytime you crawl/is super duper long/gets you stuck in a loop chances are its a trap.

7. What is the threshold of a 'super duper long' URL?

-- Use an arbitrary but intuitive threshold.

8. I'm getting UnicodeEncodeError. What should I do?

-- In Python 3.X you don't have to .encode() explicitly. But in Python 2.7.x you do. For more. For this assignment, use Python 3.6+.

9. I can see is_valid is already implemented. What else am I supposed to do?
-- A dumb (but working) implementation of is_valid is being provided. Your task is to make it smarter by augmenting it by trap detection.