Project 2 – The Crawler

Due Date: 5/3/2019 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.

Implementing your Project

Step 1 Getting the project

 # git clone https://github.uci.edu/mustafai/IR_INF141_CS121.git

Step 2 Installing the dependencies

Use python3 for this project. You shouldn't need any extra package to be installed to use the project skeleton. However, 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. In order to make it possible to work on a crawler without accessing the ICS network, this file accesses a static corpus located in the WEBPAGES_RAW directory 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 function fetch_url** (crawler.py)

This method, using the given URL, should find the corresponding file in the corpus and return a dictionary containing the URL, content of the file in binary format and the content size in bytes. To find out how you can find the corresponding file in the corpus take a look at Corpus class (corpus.py)

> **Input:** the URL to be fetched
> **Output**: a dictionary containing the URL, content and the size of the content. If the URL does not exist in the corpus, a dictionary with content set to None and size set to 0 can be returned.

3. **Define function extract_next_links** (crawler.py)

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

> **Input:** url_data which is a dictionary with the `url`, `content` and `size` keys. The `content` value is the content of the URL's corresponding file in the corpus and the `size` is the size of the content in bytes.
> **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.

4. **Define function is_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.) You will need to do some research online or apply concepts regarding crawler traps covered in class.

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

To run the project, simply run:

python3 main.py

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.

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.

Submitting Your Assignment

Your submission should be a single zip file containing the spacetime code, including your additions to crawler.py, as well as the text file with the analytics mentioned above. Please do not include WEBPAGES_RAW in your submission. 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 5/6. During that meeting, be ready for the following:

1. You will 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.
3. You will be deducted points for not showing up to your F2F or having more than one scheduled F2F time for your group.

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?
   b) How well you are able to answer questions during the F2F.

Project 2 FAQS:

1. Do I have to implement/modify any function outside the three 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.

10. Are all of my group members required to schedule a slot for F2F?

-- No. Just one of you have to schedule a slot for the entire group.