Project 1: **Web Crawler**
Web Science
50 points

Create a web crawler in Python called crawler.py that downloads web pages and searches for URLs to other pages to crawl.

Requirements

The crawler is started using optional command-line arguments and a seed URL:

$ crawler.py [args] URL

1. -n *total*

   Specify the total number of pages to download. Once the crawler has downloaded *total* number of pages or the frontier is empty, the crawler should terminate.

2. -r

   Turn on recursive retrieving, which follows links in discovered web pages. Without -r, only the given URL is downloaded, then the crawler terminates.

3. -w *seconds*

   Wait the given number of seconds between HTTP requests. If no -w option is specified, the default wait should be 2 seconds.

4. -h

   Display help information. Please display what the above options do, and output the programmer(s) name.

In addition to supporting the arguments above, the following requirements should also be implemented:

1. The crawler should only crawl pages having the same hostname as the seed URL. Example:

   ```
   $ crawler.py -r http://cs.harding.edu/
   ```

   means only URLs with the hostname cs.harding.edu should be downloaded. So www.harding.edu, taz.harding.edu, www.google.com, URLs should not be downloaded.

2. The crawler should save each successfully retrieved page in a directory called “pages” and use a filename that is the MD5 hash of the page’s URL.

   ```
   import hashlib
   filename = 'pages/'  + hashlib.md5(url.encode()).hexdigest() + '.html'
   ```

The crawler should create the pages directory in the current directory if the pages directory does not already exist.

The page's URL for creating the MD5 hash should be retrieved using response.geturl() instead of the URL used to request the page with urllib.request.urlopen(url). The two URLs will be different if the requested URL redirects to a different URL. Example:

http://cs.harding.edu/fmccown redirects to http://cs.harding.edu/fmccown/

The saved page should contain the following content:

URL
Response headers
(Blank line)
Response body

Example:

```
http://cs.harding.edu/seminar/seminar_sched.html
Date: Wed, 28 Aug 2019 16:38:25 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/5.4.16 mod_apreq2-20090110/2.8.0
mod_perl/2.0.10 Perl/v5.16.3
Last-Modified: Thu, 22 Aug 2019 20:36:18 GMT
ETag: "e15-590baa44ec42b"
Accept-Ranges: bytes
Content-Length: 3605
Connection: close
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html>
<head>
   <title>Tentative Seminar Schedule</title>
   <style>
Etc…
```

3. The crawler should use the Beautiful Soup library to discover links to other pages. Run the following pip command to install Beautiful Soup:

```
pip install beautifulsoup4
```

The example below creates a list of links from the downloaded web page:

```
from bs4 import BeautifulSoup
resp = urllib.request.urlopen('http://python.org/')
html = str(resp.read(), encoding='utf8')
soup = BeautifulSoup(html)
links = soup('a')    # list of all links
```

4. Pages should be crawled in the order in which the links are discovered by the Beautiful Soup library. This is important so I can more easily test your crawler with mine.

5. Only pages that have the text/html MIME type should be saved and examined for links. All other MIME types should be ignored.

6. Your crawler should output the URL being crawled, followed by a status prefaced with --. The status should be a "Saved" message for pages successfully saved, "Skipping" message for non-HTML content, or "Could not access" message for 404, 500, or other error responses. A "Limit" message should be output if the crawler stops because the -n limit has been reached. Example:

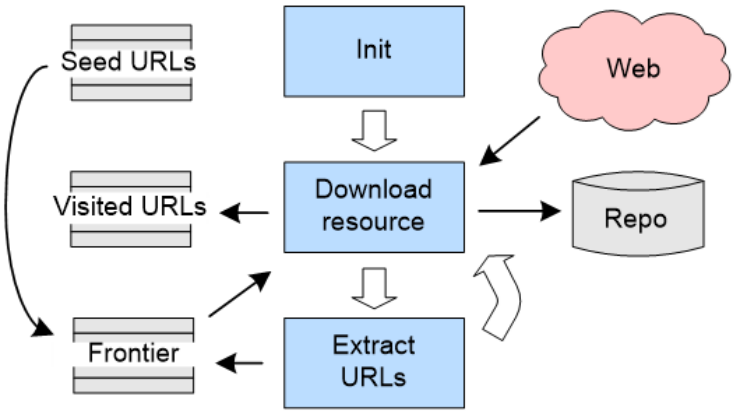
```
crawler.py -r -n 5 http://cs.harding.edu/
Crawling: http://cs.harding.edu/
-- Saved to pages/8695073990b9599d9c8db3d305223b10.html
Crawling: https://cs.harding.edu/easel/cgi-bin/login
-- Saved to pages/e2681a74393964d0cf9fd35eb374b727.html
Crawling: http://cs.harding.edu/room_request.pdf
-- Skipping application/pdf content
Crawling: http://cs.harding.edu/seminar/seminar_attendance.html
-- Saved to pages/4efd0b9425e9306e40269fdc5980e69e.html
Crawling: http://cs.harding.edu/seminar/archive.html
-- Saved to pages/2acd175d5d0da5471e5ba8bb32df4aee.html
Crawling: http://cs.harding.edu/seminar/seminar_sched.html
-- Saved to pages/e9fd96fc3a977d66d915e71909daca74.html
Limit 5 reached
```

7. The crawler should set the HTTP request header User-Agent to "WebSci Crawler".

How a Crawler Works

Primary data structures:

1. Frontier
   a. Links that have not yet been visited
   b. Implement as a list to simulate a queue
2. Visited
   a. Links that have been visited
   b. Implement as a set to quickly check for inclusion
3. Discovered
   a. Links that have been extracted from downloaded HTML
   b. Implement as a list to keep the order



Crawler pseudocode:

```
Place seed urls in Frontier
For each url in Frontier
    Add url to Visited
    Download the url
    If the downloaded content is HTML then
        Clear Discovered
        For each link in the page:
            If the link is not in Discovered, Visited, or Frontier then
                Add link to Discovered
        Add links in Discovered to Frontier
    Pause
```

Bonus

Three bonus points are available for making the crawler respect the robots exclusion protocol (robots.txt). Do this by looking for a robots.txt file for any new domain names encountered, and keep the list of excluded URLs in memory. Before a URL is to be added to the frontier, make sure it doesn't match any of the excluded URLs. For each crawl, you should only read robots.txt once for each domain.

Submit

Submit your crawler.py to Canvas. If you work in pairs, you must use pair programming (both individuals work together on the *entire* program together), and only one person needs to submit the solution.

Use good coding practices, including good variable names and functions where appropriate. Put comments in your code citing the URL of any code you reused from the web.