CSCI 136 Assignment – Web Crawler

March 22nd, 2023

Due Date: Tuesday, 3/28/2023 11:59PM

Points: 30

Topic: Regular Expressions, HTML-Processing, Web Crawling

NOTE: There is potential in this lab assignment to cause network or server issues knowingly or unknowingly. Denial of Service Attacks are conducted by bombarding networks and/or servers with excessive traffic to the point where their operation degrades or fails. Be very careful with the code you write for this lab and your use of it. You could cause harm, including having your network port shut down due to suspicious/malicious activity.

In this assignment, you will use regular expressions, recursion, web requests, and text file output. Your code will implement a web crawler that searches web pages for links to other web pages and then explores those links recursively.

After following all instructions below, please submit your Python script (`Crawler.py`) in Moodle. Upload this as an individual file – do not zip it. Be sure the script file has a header section that includes the file name, your name, credits, a description of the file, and other useful information. Make sure classes, methods and functions are thoroughly commented.

- `Crawler.py`

Grading:

This assignment is worth 30 points. You will be graded according to the following criteria:

Grade Item	Points
Program Compiles and Runs	4
Header Comments	2
Regular expression works	4
crawl() Function works	3
crawl() Function is recursive	4
url pages opened and read correctly	4
Links stored in dictionary	3
Count of link encounters is correct	3
Results written to file	3
Total	**30**

Web Crawler

For this assignment you will write a program that takes two command line arguments. The first argument is the full url of the page where you want your web crawler to start. For example, if you wanted to start on the Montana Tech website you would enter https://mtech.edu/. The second argument is how many links deep you want to explore. While testing your program, we likely will only go two levels deep. Three levels could result in a large number of links and would take a long time to run. There is not a lot of code to this assignment, but it will need to be correct.

Example Call:

The following should visit the main Montana Tech web page, find all links contained within, visit those links individually, and find all links contained in them. Using a dictionary, it keeps track of the links and how many times each link is referenced. It then outputs the links and counts from the dictionary to a text file named `links.txt`.

```
>python Crawler.py https://mtech.edu/ 2
```

Example Output to links.txt:

```
2 https://www.mtech.edu/
8 https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.2.1/css/fontawesome.min.css
…
10 https://outlook.com/mtech.edu
1 https://aadcdn.msauth.net
1 https://login.live.com/Me.htm?v=3
…
8 https://my.mtech.edu/
1 http://mymtech.mtech.edu/
1 http://orediggerweb.mtech.edu/
…
```

Your program should have two parts:

Main Program

The main program (`Crawler.py`) will read the command line arguments and pass them to the recursive `crawl()` function. Once the function completes, your main program should write the contents of the dictionary to a file named `links.txt`. Each link should be output on its own line when written to the file, and the count should be listed immediately prior to the link and on the same line.

Recursive crawl() Function

The `crawl()` function will take three arguments: the recursion level, the current link, and a dictionary to store links in. `crawl()` will:

- Open the web page specified by the link.
- Read the HTML contents of that page into a string.
- Find all the links specified on that page and for each one, recursively call itself with the next lower recursion depth number, the link and the dictionary.
- The dictionary is a single dictionary used to store links and a count of how many times each link has been encountered.
- The dictionary should be indexed by links and should store the count associated with individual links.

There are two base cases to control/stop the recursion:

- If the recursion level is 0, update the dictionary and then stop/return.
- If the link has been encountered before, increment the count in the dictionary for that link and return. Do not explore it again.

You will use a regular expression to find the links within each page you read in. Links are specified by the attribute `href="`*`link`*`",` where *`link`* is the link to another page. The quotes around the link are part of the pattern. We have to be careful in using regular expressions because they use a "greedy" approach. Example, if we use `.*` to match 0 or more characters, it will match everything until the end of the text.

You are welcome to experiment and try to create a regular expression for this purpose. Or you can use the following, which matches the desired pattern:

```
href=\"[^\"]*\"
```

This regular expression matches the pattern `href="`*`link`*`"` by matching `href=` followed by a double-quote, subsequent non-double-quote characters (the link), and the closing double quote. If we simply tried to match characters between two double quotes, the regular expression would be processed in a greedy manner and would read until the last quote in the text.

You will need to extract the link portion from each match and use it accordingly.

Helpful Hints:

Do I need to follow the given specifications? Yes. You must implement methods as described. You may add additional methods if they are helpful to your code.

How do I get the source html for a given link? Here is an example that retrieves the html for the main Montana Tech web page:

```
import urllib.request
link = "https://mtech.edu/"
page = urllib.request.urlopen(link)
html = str(page.read())
print(html)
```

A request can fail and throw an exception for a variety of reasons including that the link specifies an unknown or unreachable server. You should wrap your attempt to retrieve the html for a given link within a `try-except` block to gracefully handle a failure rather than crash your program. If a link fails, catch the exception and move on.

Should I separately parse or split the HTML string before search for the regular expression? You should not have to do this. `re.findall()` will find all occurrences of the pattern within the full string. You can iterate through the matches returned by this method, and each match will correspond to a link.

Should I follow every link that is identified by the regular expression pattern? There are a variety of "link types" that can be included in an href expression in HTML. Many of these are outside the scope of what we are trying to accomplish in this lab. For this lab, you should only use links that start with http or https. You could modify the regular expression pattern to identify only those starting with http/https or you could check those that match the broader pattern for containment of http/https. Ignore all other links.

My program runs for a long time and doesn't produce any output. How do I debug it? Include print statements to indicate values of variables, calls to methods, etc. Make sure these print statements are removed, commented out or otherwise suppressed in your final submission. Test with small values (0, 1, 2) for depth and with known links/websites.

Submit Your Work

After following all instructions above, please submit your Python script (`Crawler.py`) in Moodle. Upload this as an individual file – do not zip it. Be sure the script file has a header section that includes the file name, your name, credits, a description of the file, and other useful information. Make sure your classes, methods and functions are thoroughly commented.

- `Crawler.py`

EXTENSIONS

You will not submit the answers to these extensions as part of your assignment submission. You should work through and understand them to learn more and practice. You may be held responsible for the content within, so you really should work through them. With that said, this are outside the scope of what will be submitted for this lab. Do not change your original submittal to address these extensions.

1. For this lab, we assumed that links are specified within double quotes. Single quotes are allowed for this purpose as well. How could you change the regular expression you used to match links specified with single or double quotes?

2. As mentioned earlier, there are a variety of link types, and you were asked to only use links that start with http or https. Another type of link references pages residing on the same website without specifying the site name or http/https. This type of link can be a path relative to the current page's path on the server or an absolute path off the root of the site. How would you navigate to and explore such links? Hint: you would need to extract and use portions of the containing link along with the contained link.

3. JavaScript and client-side manipulation of the DOM (document object model) for a given page as displayed within a browser allows for dynamic creation and manipulation of links that otherwise do not appear within `<A href="…">…</A>` tags in the HTML retrieved directly for a given page. How could you identify and explore these links via Python scripts like the one you created for this lab?

4. The Beautiful Soup Python package is a robust, well-documented package for parsing HTML and XML. It can be used to accomplish tasks like what you did in this lab and much more. Investigate how Beautiful Soup could have been used to do this lab. Would it make the task easier? Investigate other things the Beautiful Soup can do.