NYU Tandon School of Engineering   September 11, 2025
Computer Science and Engineering
CS 6913, Fall 2025

Assignment #1 (due September 26)

In this assignment, you are asked to write a (very primitive) web crawler in Python that attempts to crawl web pages in a BFS-like manner, but with priority given to crawling pages from diverse domains, and with emphasis on efficient crawling. The purpose of this assignment is to learn about crawling, to start programming in Python, and to learn a bit about the various structures and features found in web pages and how to handle/parse them. You should work on this homework individually, not in groups. There will be a demo with the TAs where each student has to show their crawler and answer some basic questions about their design. The project **must** be done in Python.

More precisely, your crawler should allow the user to input a query, and start crawling from the ten or more result pages returned by a major search engine on this query. Your crawler should follow a roughly BFS policy, but give priority to pages from domains from which fewer pages have been crawled, and also look at how many other domains from the superdomains have already been crawled. As an example, you could define the priority for a page as $1/\lg(p)$ where p is the number of pages already crawled from the same domain, and then add another term modeling the novelty of the domain within its superdomains. But the details are up to you. Your crawler should also run fast, at least 10 pages per second but hopefully faster. To do this, you should consider using multi-threading for parts or all of your application. You should crawl each URL at most once.

Your program should output a log file with a list of all visited URLs, in the order they are visited, together with information such as the size of each page in bytes, the time of the access, the return code, its page and domain priority scores, and its depth in the crawl – meaning, its distance from the seed pages in the crawled part of the web graph.

There are a couple of tricky issues that come up in this assignment. Following is a list of hints and comments on the assignment. More help on this will be provided in the next few days. But please get started right away, and ask the TAs when you run into problems!

Downloading Pages: Python has a module called `urllib` that contains functions for downloading web pages. Check it out to find the right function for downloading a web page from a given URL. Please do not use powerful crawling libraries such as `scrapy` for this assignment, but stay on the basic Python level when it comes to retrieving each individual URL.

Parsing: For each web page that you encounter, you will need to parse the file in order to find links from this page to other pages. Python provides some convenient functions for these types of problems in modules called `htmllib` and `xmllib`, which are explained on the Python web site at `www.python.org`. Note that in addition to "normal" hyperlinks, a page may also contain hyperlinks as part of image maps (i.e., by clicking on an image you get to the linked page) or within javascript or flash; you can either ignore these links, or you can try to parse stuff such as javascript. (If you miss some pages, it is no big deal.)

Ambiguity of URLs: Note that URLs, as encountered as hyperlinks in pages, are "ambiguous" in several ways. If a URL ends with `index.htm`, `index.html`, `index.jsp`, or `main.html`, etc., then we can usually omit this last part, but this is not always the case. The same content my be served from different hosts. Also, hyperlinks can be relative – if you find a link to `../people/joesmith.htm` on a page, then this link is relative to the current page, with `..` denoting going one level up in the directory structure. Such relative links need to be properly resolved. Finally, check out the meaning of the `<base>` tag in HTML. Check out the Python module `urlparse` and the function `urljoin` to deal with some of these issues.

URL Forwarding: You will also see that some URLs forward (change) to other URLs, sometimes several times in a row. Check how to best handle this issue as needed.

Different Types of Files: Apart from HTML files, your crawler may encounter links to many other types, including images, pdf, audio files, XML, etc. Focus on files with mime type `text/html`, and avoid downloading and parsing other file types. Try to use the information supplied by file endings (e.g., `.html`, `.asp`, or `.jpg`) to generate a blacklist of endings that result in a link being discarded, put also ask for the MIME type of a file after you download it, for those cases where an ending is not in your blacklist.

Checking for Earlier Visits: You need to check if a page has already been visited. For this assignment, you may use the dictionary structure provided by Python and use the normalized URL (after any forwarding) as key.

Be Considerate when Testing: At first, your crawler will probably be very buggy and thus misbehave often. So do not use it for large crawls until you have found most of the bugs, and also periodically vary the seed pages so you do not constantly access the same web site. As you vary the seeds, and thus visit new sites, you will at first constantly run into new bugs and challenges that you can try to resolve – this is the point of the homework. But you will probably not be able to overcome all problems – so do as much as you can. If you can reliably download say ten thousand pages in half an hour that will be OK, though larger and faster crawling may give you extra points.

Also, implement the *Robot Exclusion Protocol*, to avoid going into areas that are off-limits. Make reasonable decisions about how to deal with CGI scripts. (For example, you could decide to not crawl any URLs with the string "cgi" in it.) Also, detect 404 and 403 return codes and discard such pages.

Exceptions: Make sure that your program does not break if the server at the other end fails to respond. Use the `try` command and exceptions in Python whenever you request a page. Set suitable timeouts for request to make sure your program does not hang for minutes or forever. If necessary, also make sure your crawler does not get stuck on links to password-protected pages.

To summarize, your task is to build a basic but fast web crawler in Python that tries to crawl pages based on a BFS-type traversal with preference for new domains. You may use libraries for tasks such as HTML parsing, downloading a file located at a URL, and Robot Exclusion, but of course you should not simply download and reuse a complete Python crawler or even a powerful crawling library. You should maintain your own data structures for the crawling queue. More details about what to submit are provided later.