Mehran Ali Banka (mab10251@nyu.edu)

WEB CRAWLER IMPLEMENTATION:

SUMMARY:

This Project was to implement a capable web crawler in Python. In this document, I have listed the details of the implementation. Multiple factors were considered, such as – fresh leads are prioritized, making sure the crawler doesn't spend too much time in a domain, rate limiting is ensured, non-polite URLs are blacklisted, sensitive URLs are not crawled, etc. Due to the limited computing resources I had, I believe I could not run this crawler to its best potential. I have given a parameters file (Parameters.py) with multiple parameters to control the power and flexibility of this crawler. These parameters can be set based on the computing resources and other considerations.

LOGIC OVERVIEW:

Queue:

The code at a high level does a BFS with a priority queue and random sampling. For the priority queue, I give the best priority to the nodes that are either

1) New geography – weight: 10*(100 - *geographic_representation*)
2) New language – weight: 5*(100 - *lang_representation*)
3) New domain – weight: 2.5*(100 - *domain_representation*)
4) Log of Size of the content in the URL – weight: *log(url_response_size)*

I used https://ipinfo.io to get the geography of a URL. As the weight of these would not remain same through the lifecycle of the code, I have decreased the weight as we go along. For example, when we visit a Chinese page for the first time, it should have more priority than when we visit it the second time. This is the reason for the multiplication by 100 – ***representation_percentage***

Selecting Child Nodes:

As the number of URLs per node can be huge, and with rate limiting and thread sleeps enforced, the crawler can spend too much time collecting, validating and scoring the child nodes, rather than sample more pages. This is strictly a limitation of computing resources and time. But there is also the problem that considering all pages from a URL might lead to localization in the graph. So, based on the paper() suggester in the course, I have randomly sampled the child nodes of a URL – I used random.sample() for this. And this can also be controlled with the parameter ***max_child_per_page.***

Validating URLs:

In order to decide whether to put a URL in the queue, various rules are enforced. Don't visit a URL if:

1) It is already visited
2) It has a length of greater than 100 characters
3) It is a query URL or most likely a garbage URL (Special characters like '%')
4) It is not supported (***supported_crawl_types***) or it is unresponsive
5) It does not allow crawling (Robot exclusion)
6) It is a sensitive URL like .gov (***urls_to_avoid***)
7) Its domain has been reached too many times (***max_pages_per_domain***)

Various other checks are done on the fly, in the code, to limit the number of GET requests just at the validation stage. These include checking for password protection, or if a URL's domain has been banned by the code because it visited the domain earlier and didn't get a friendly response (like 429)

Rate Limiting:

Since I ran this code with limited computing resources, I have added sleeps in various parts of the code to simulate a simple rate limiting mechanism. I did not want to get flagged for DDoS attacks. This is again controlled by parameter ***max_requests_per_second*** (N) and the sleep is of the order of 1/N.

Seeding:

For this code, I wrote a separate module to do seeding (seed_loader.py). I wanted to have multiple strategies for seeding, for example – using a static pre-loaded list or querying a web-engine etc. I could not spend much time on this, so currently it seeds only from a pre-loaded list. Some parameters have been given to control seeding as well as extending seed_loader.py in the future. These are:

max_pages_per_seed: Controls max pages sampled that started at this seed.

max_number_of_seeds: Take only the top-N seeds from the list of lists.

seed_strategy: Which module to run in seed_loader.py. Currently, only supports one(default)

Logging:

I have written a simple logger class by extending the logging module in Python. By default it logs both the pages visited as well as pages sampled. Pages visited can add a lot of noise, so we can control whether to log them or not (***log_all_explored_files***). The logger mentions size, language and country of URL it samples as well, but not for the ones that it just visits. No errors have been logged in the file at this moment. At the end of the code, the logger puts in the summary of the crawl – the frequency of each language, of each geography etc, for analysis purposes.

Other parameters in Parameters.py are self-explanatory and don't control the logic much.

RESULTS:

The logger did surprisingly well to visit many languages. The performance was not great because of rate limiting, sleeps and low computational power. Sometimes, the logger would get stuck for a while in trappy domains but it generally comes out. I had to kill the process a few times too. More enhancements are needed. Please check the summary stats at the end of the log file I have given to get and idea. Note the stats only cover the pages that were sampled, so they are not based on a lot of data.

Pages Visited: 5055, Pages Sampled : 820.

Unique Languages: 47, Unique Countries: 1 , Unique Domains: 93.

Crawling Runtime: 5436.111655 seconds (1.5 hours)

Language	No of pages sampled	Percentage
Spanish	12	1.463
Chinese	2	0.243
Polish	8	0.975

Raw Language Statistics from the log file end:

```
Language Details:
----------------------
Language: en, Count: 357
Language: de, Count: 41
Language: so, Count: 5
Language: sl, Count: 5
Language: sv, Count: 38
Language: ca, Count: 6
Language: da, Count: 2
Language: sk, Count: 2
Language: ja, Count: 34
Language: ko, Count: 7
Language: sw, Count: 38
Language: fr, Count: 3
Language: tl, Count: 1
Language: af, Count: 4
Language: es, Count: 12
Language: cy, Count: 1
Language: hr, Count: 35
Language: ro, Count: 5
Language: fi, Count: 9
Language: lt, Count: 3
Language: pl, Count: 8
Language: nl, Count: 3
Language: it, Count: 4
Language: NA, Count: 3
Language: id, Count: 7
Language: tr, Count: 5
Language: uk, Count: 34
Language: he, Count: 30
Language: zh-cn, Count: 2
Language: lv, Count: 30
Language: cs, Count: 1
Language: ar, Count: 1
Language: pt, Count: 2
Language: no, Count: 2
Language: el, Count: 26
Language: et, Count: 4
Language: bg, Count: 3
Language: hu, Count: 1
Language: ru, Count: 7
Language: mk, Count: 30
Language: te, Count: 1
Language: ur, Count: 3
Language: vi, Count: 1
Language: hi, Count: 2
Language: ta, Count: 2
Language: fa, Count: 1
Language: ne, Count: 1
```