README.md

Domain Crawler - Domain Mapping

This application is a web crawler which crawls all the pages within a particular domain and produces a sitemap (only internal URLs) for that domain and finds the links between its web pages, and finally stores the whole sitemap of the domain the local hard disk. The application also lists all the broken and dead URLs within that domain and stores them on local hard disk.

How to install the application software requirements

```
$ cd sitemap
```

Choose only one of the below option (Read ***setuptools Note*** below):

- If user wants to install and use the required packages:
  ```
  $ sudo python setup.py install
  ```
- Instead, if user does not want to install the required packages, however, still would like to just use them, then execute:
  ```
  $ sudo python setup.py develop
  ```

This application has been developed and tested on Ubuntu 16.04 LTS OS using Python 3.5.2. For other OS platforms, few instructions might need to be adapted.

setuptools Note: If *setuptools* is not installed on user's system, please execute these two instructions first, as *setuptools* package is the prerequisite for *setup.py* to work:

```
$ sudo apt install python-pip
$ sudo pip install setuptools
```

How to run the application

```
$ cd sitemap
$ export PYTHONPATH=$PWD
$ python generate_sitemap.py -nt 8 -l 4 -t 5 -f monzo_sitemap.txt -d https://monzo.com/
```

Output of application:

```
Sitemap for https://monzo.com/ is written in ./output/monzo_sitemap.txt.
Logs (Broken or dead URLs along with application logs) for domain https://monzo.com/ are available in
./logs directory.
----------------------------------------------------------------------
<<Thank you for using Domain Crawler - Domain Mapping application>>
----------------------------------------------------------------------
```

Help on command-line options and arguments of the application:

```
$ python generate_sitemap.py -h
```

```
usage: generate_sitemap.py [-h] [-nt N] [-l L] [-t T] [-f File_Name] -d Domain

Domain Crawler - Domain Mapping

optional arguments (For an invalid arg value, default value will be picked up):
  -h, --help            show this help message and exit
  -nt N, --nthread N    number of threads (N>=0: default=4)
  -l L, --log L         log level (0<=L<=5: default=2)
  -t T, --timeout T     timeout in seconds (T>=0: default=10)
  -f File_Name, --file File_Name
                        name of the output file (default=output.txt): stored in the output directory

required arguments:
  -d Domain, --domain Domain
                        name of the domain
```

Note 1: The sitemap result and logs get stored in ***./output*** and ***./logs*** directory respectively. The log files contain broken or dead links within the domain along with application logs.

Note 2: Ideally the user should provide an absolute web address for a *Domain*. In case the provided value is different than absolute domain address, the application calculates the absolute domain address on its own using provided values. Example: If a user provides *Domain* = 'https://monzo.com/about/', then the application will still build the sitemap for 'https://monzo.com/' only.

Note 3: By default, the application enforces default values for optional arguments. However, the application supports flexibility to change default values as per user's requisites. Please visit Custom Default Configuration Settings subsection for changing default configuration settings.

How to execute unit tests for the application

```
$ cd sitemap
$ python -m unittest discover
```

How to produce code coverage report of the application

[*Only once*:] If *Coverage* package is not installed on user's system, please install this package (version 4.5.1 with C extension) in the local system by executing the following command:

```
$ pip install -U coverage
```

```
$ cd sitemap
$ sh eval_code_coverage.sh
```

Note: The produced code coverage report will get generated inside ***./documentation/coverage_html_report/***. The above executed shell script automatically opens the home HTML page (***./documentation/coverage_html_report/index.html***) of application code coverage report.

Custom Default Configuration Settings (For Advanced Users)

A user can adjust the default configuration of this tool by modifying **dflt_cfg.py**. By default, each field is assigned a value. For any invalid/absent optional command line argument values, default values stored in *dflt_cfg.py* will automatically get enforced for the application. This section details the explanation of each such field.

- **NUM_THREADS**: Specifies the number of threads that crawls the domain concurrently.
  - Expected value: Non-negative integer
  - Default value: *4*
- **OUTPUT_PATH**: Specifies the file location where generated sitemap for a domain will get written.
  - Expected value: A valid absolute/relative system path to a file in a string format. In case of the relative path, it should be relative to application home directory (./sitemap) only.
  - Default value: *"./output/output.txt"*
- **TIMEOUT**: Specifies the waiting time (seconds) before the application terminates in case of an unresponsive domain.
  - Expected value: Non-negative integer
  - Default value: *10*
- **SYSTEM_PROXY**: Using a dictionary (e.g; proxies below), it specifies mapping protocol or protocol and host to the URL of the proxy to be used on each *urlopen* request.

  If the user system is behind a proxy, assign a dictionary (e.g; using proxies argument) mapping protocol or protocol and host to the URL of the proxy to be used on each urlopen request using SYSTEM_PROXY configuration.

  If the user system is not behind a proxy or user prefers urllib to auto-detect the proxies from the environment variables, then please set it to None. Normally that's a good thing, but there are occasions when it may not be helpful (Reference: Footnote 5).

  To disable autodetected proxy, please pass an empty dictionary. For example, in order to test scripts with a localhost server, a user might need to prevent urllib from using the proxy and thus requires {}.

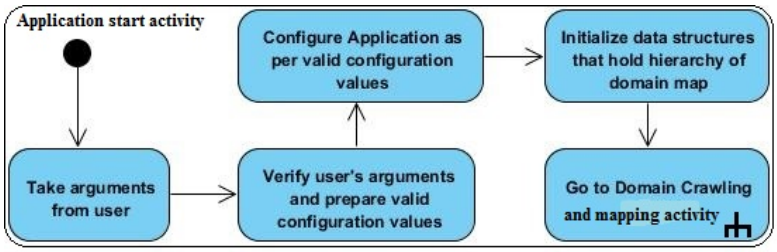
- Expected value: dictionary / None
- Default value: *None*

For detailed information, please visit urllib Proxies Details.

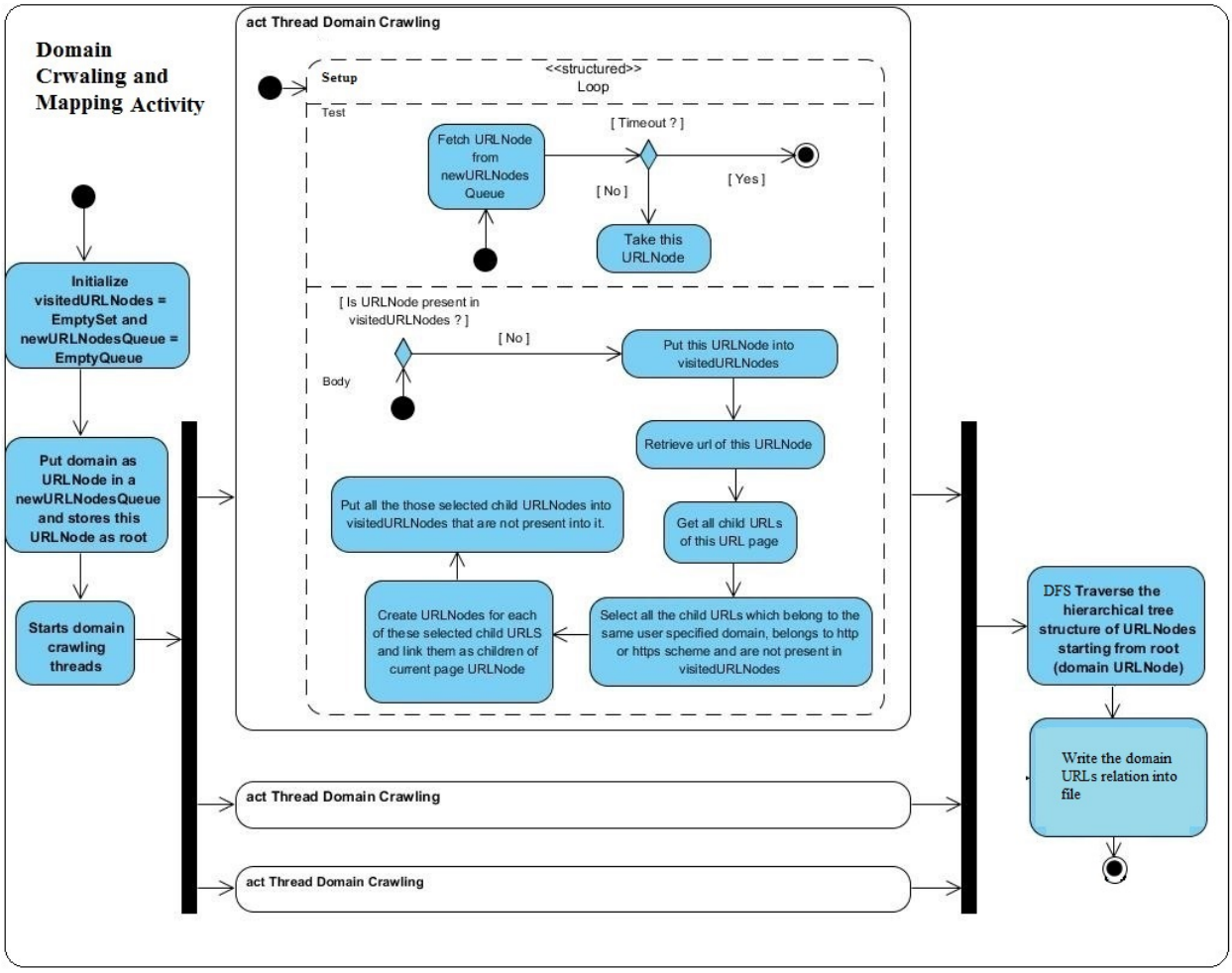
```
proxies = {
          'http': 'http://10.10.1.10:3128',
          'https': 'http://10.10.1.10:1080',
}
```

- **LOG_LEVEL**: Specifies the log level of the application.
  - Expected value: 0 <= LOG_LEVEL <= 5 (0 is the lowest level log severity and 5 is the highest level log severity)
  - Default value: *2*

Algorithm of Domain Crawler - Domain Mapping



Activity Diagram for Application Initialization



Activity Diagram for Domain Crawling and Mapping

Future Possible Extension

Since this application parses all the active URLs in a domain in a hierarchy of interconnecting URL-nodes, it is possible to extend this application to introduce an enhancement that would produce the graphical representation of domain URL inter-connectivities. This application has already addressed the issue of loops between URLs. All the end URL nodes of the inbuilt hierarchy of interconnecting URL-nodes are either dead-end URLs (which do not direct to any other UR) or an URL node which direct to first already visited URL node, thus resolving the issue of loops between URLs.

References:

- 1. https://pymotw.com/3/urllib.parse/#parsing
- 2. Uniform Resource Identifiers (URI): Generic Syntax: https://tools.ietf.org/html/rfc2396.html
- 3. pysitemap 0.5: https://pypi.python.org/pypi/pysitemap/0.5 (We have implemented am entirely different approach than this, as we have built a hierarchy of interconnecting nodes which is capable to support a wide range of future requirements (Please see the Future Possible Extension)section. We also list dead or broken URL links within a domain.)