**Wiki crawler (https://en.wikipedia.org/wiki/Wikipedia:Getting\_to\_Philosophy)**

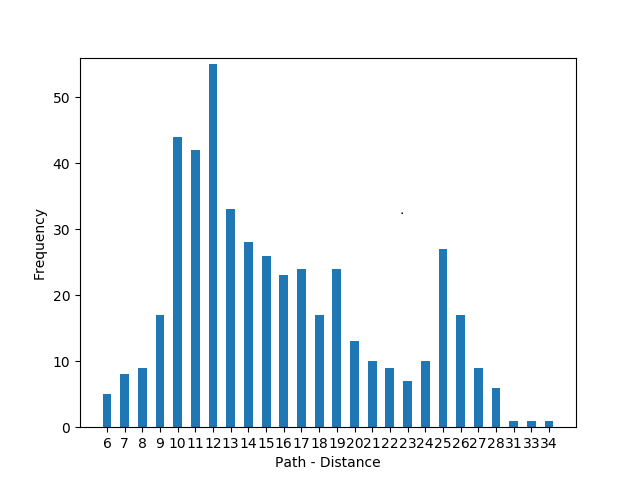
Write a program using Python that performs the following:

Take any random article on Wikipedia (example: http://en.wikipedia.org/wiki/Art) and click on the first link on the main body of the article that is not within parenthesis or italicized; If you repeat this process for each subsequent article you will often end up on the Philosophy page.

* **Code is attached in the forwarded zip file.**

**Questions:**

* What percentage of pages often lead to philosophy?
* **After running 500 iterations, 93.2% of the pages lead to the Philosophy Wikipedia page. There were several infinite loops observed (Congo\_Lion -> Lion) during the program execution which had to be forced terminated by putting in a tolerance level.**
* Using the random article link (found on any wikipedia article in the left sidebar),what is the distribution of path lengths for 500 pages, discarding those paths that never reach the Philosophy page?
* **Distribution of path lengths for 500 pages.**

****

* How can you reduce the number of http requests necessary for 500 random starting pages?
* **We can use a Hash-map to store all the links which have been already looked up. Right now, a lot of links are looked up on multiple occasions. We can then use memoization to reduce the amount of look ups and that way saving significant amount of time. For instance, Ontology to Philosophy is looked up several times, if that information is saved in a Hash-map, we do not have to look it up again.**
* **If the pages are present locally instead of on a server, the number of HTTP requests can be reduced by 100%. Since, all the pages are present locally, it is pretty much looking up the page inside the local server repository which can be achieved without making an HTTP request.**
* **Additionally, if the above two cannot be achieved then mobile version of the Wikipedia pages can be looked. The mobile sites essentially contain the same amount of information but are much less heavy than the Desktop version of the sites.**
* **Another solution would be to make this program multi-threaded. The program, however, will still make the same number of HTTP requests as the original program but the entire process will run faster.**