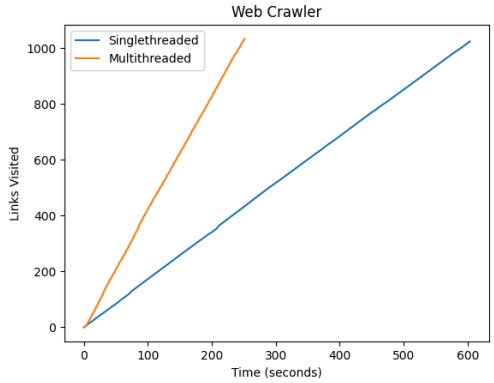
The design on my web crawler is relatively simple and built entirely in Python. My main method creates an instance of a custom web crawler class and sets the seed URL to be “cc.gatech.edu”. I have two modes for my web crawler, a single threaded mode and a multithreaded mode. When I first created the crawler, I wrote it to be single threaded as a proof of concept, but after testing and confirming the functionality, I altered it to be multithreaded in order to speed it up. In order to keep track of the links to visit, I use a deque for O(1) appending and popping from front. I also keep track of links that have already been visited in a set so that I do not add redundant links into the deque. The main loop for the crawler is that it takes the URL from the front of the queue and performs an HTTP GET request to get the HTML. Then it parses the HTML using BeautifulSoup and gets the text from all children of the div with role=”main”. I did this to prevent the crawler from finding keywords on the page that would be on almost every page like the “college of computing” on the headers and the navigation bar. I use a Counter object to get a count of every word in the body. I use these keywords that were just extracted and add them to an index I keep in memory and later wrote to a file called “index.txt”. This is just a dictionary with the key being the keyword and the value being a list of tuples containing the URL which used it as well as the frequency of the keyword on that page. I also filter the keywords and exclude certain words such as “and”, “to” and other very common words that are universal to English sentences and do not have relevance to the topic of the page. After that, I iterate through all <a> tags in the page and add all “href”s to the deque that are in the domain specified by the user. The pros of this design is that it was relatively quick to spin up, it is relatively system agnostic, and is simple. The cons of this design is that the speed is bottlenecked by the speed of the hardware and all data structures are stored in memory so crawling a large number of sites will likely crash the program.

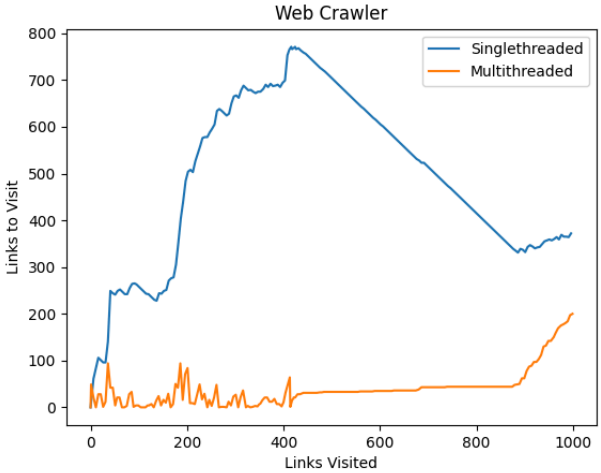
The following are screenshots of the crawler running in the terminal:

```
Lucas@DESKTOP-5VDQ8CC MINGW64 ~/Documents/College/Internet/web-crawler (threading)
$ python main.py
Starting the multithreaded web crawler with seed ('https://cc.gatech.edu/', '', 'https://cc.gatech.edu/')
'NoneType' object has no attribute 'get_text'
('https://cc.gatech.edu/', 'https://cc.gatech.edu/', 'https://cc.gatech.edu/cas')
thread: 100
successes: 94
thread: 200
successes: 191
thread: 300
successes: 291
Some characters could not be decoded, and were replaced with REPLACEMENT CHARACTER.
'NoneType' object has no attribute 'get_text'
('https://cc.gatech.edu/', 'https://cc.gatech.edu/degree-programs/phd-computer-science', 'https://cc.gatech.edu/sites/default/files/documents/2022/202105%20Computer%20Science%20PhD%20Handbook.docx')
thread: 400
successes: 386
thread: 500
successes: 465
thread: 600
successes: 564
thread: 700
successes: 665
thread: 800
successes: 764
thread: 900
successes: 864
thread: 1000
successes: 964
successes: 1003
to visit: 208
visited: 1035
--- 252.0883150100708 seconds ---
Starting the web crawler with seed ('https://cc.gatech.edu/', '', 'https://cc.gatech.edu/')
'NoneType' object has no attribute 'get_text'
('https://cc.gatech.edu/', 'https://cc.gatech.edu/', 'https://cc.gatech.edu/cas')
successes: 101
to visit: 257
visited: 101
time elapsed: 59.68607139587402
unique links to visit: 256
```

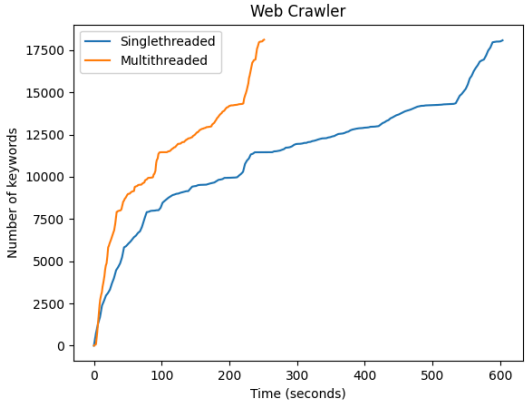
The following are graphs of the crawler’s speed.



Ratio of queue size and links visited



Number of keywords by time



Metrics:

Singlethreaded: 1000 links crawled in 606.0178701877594 seconds.

approximately 98.816 links per minute

10,126.77 minutes to crawl 1 million links at this rate

10,126,770.38 minutes to crawl 1 billion links at this rate

Multithreaded: 1000 links crawled in 252.0883150100708 seconds

approximately 238.194 links per minute

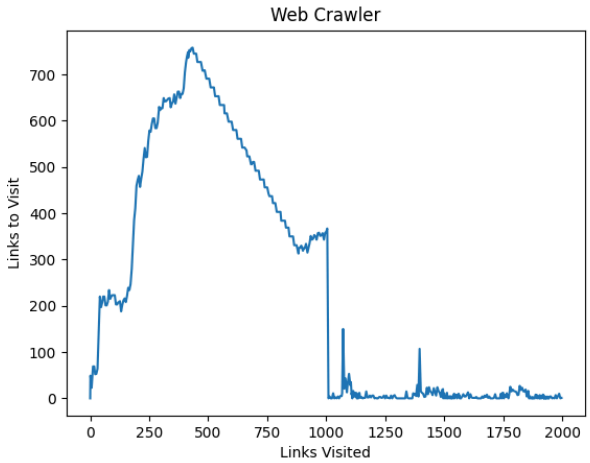
4197.706 minutes to crawl 1 million links at this rate

4,197,705.747 minutes to crawl 1 billion links at this rate

Reflection:

I severely underestimated how many URLs cc.gatech.edu had. I ended up running my crawler for more than 1000 links and was surprised by how long it went. When initially thinking about cc.gatech.edu I did not take into account the number of articles and calendar events that each have their own individual pages as well as every faculty member having a page. Additionally, I found that there were a lot of dead links that gave me 404 errors. I looked at the links that were causing these errors and it was mostly calendar links that were 404ing. If I had more time I would like to possibly improve my design. I found that creating the index and looking for keywords took a lot of time for my crawler to do. An idea I had was creating an AWS Lambda function that my crawler could use that could handle the processing of the keyword extraction and store it in a DynamoDB instance. Additionally, if I took it a step further I could have a machine on the cloud that kept track of the queue and did this, I could run the crawler on multiple machines at once that communicated with the machine in the cloud to get the URLs it needs to crawl.

Also I ran the crawler one last time for as long as I could and got the following:



This is crawling ~2000 links in 1036.38 seconds ~17.27 minutes