Compiling instructions:

I honestly tried to get this working without the folder structure but my input files and output files are based on the folder structure so I decided to keep it. It is also a 3 step compiling process so I think this will be easy for you to do!

Compiling Steps:

1. Unzip hw2
2. Copy and paste ap89 documents into hw2/src/indexer/ap89_collection
3. Run indexer.java

Description of outputted txt files:

Over1000.txt - the terms with frequencies of over 1000. There is a count at the bottom
Only1.txt - the terms that are only found once, there is a count at the bottom
Stat.txt - the Total # of Docs, # of Words, and # of unique words
Results.txt - My results for the query searching, including the count of results per term, and then the top 5 documents (docID and title) per term.
heap/zip files - these files I used to generate the charts found below. I have my program set to automatically generate these as well so you can see how i got here.

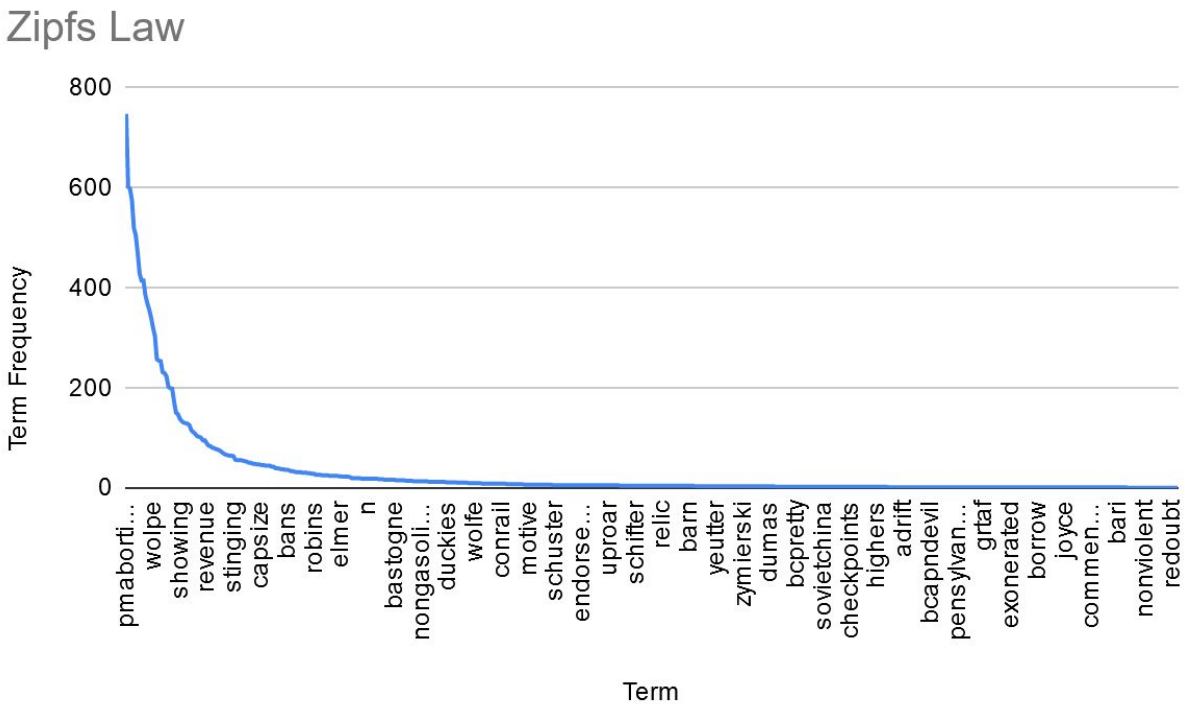
Description of Design:

Since I was building off my first assignment, my approach remained roughly the same. I used a tokenizer to break down the words and remove stopwords. Then I produced a HashMap of tuples, where each term is the key and the value is a list of tuples. I also made a docMap to store my own docIDs along with the actual Doc number. This was useful in many ways, especially in my search and gettitle function. I chose to not save each title as I went, but rather based on a DocID i made a function that outputs the title of that document. This helped me avoid saving every title.

After getting my indexer working I made several helper functions that are there to answer the questions on the assignment. I created a new output file for each function and wrote the information it produced into those files. This made it really easy to in one program produce all of this information, without it overwhelming the stdout printing. Each function looks at the data in a different way and produces results that can be clearly seen in the corresponding text file.

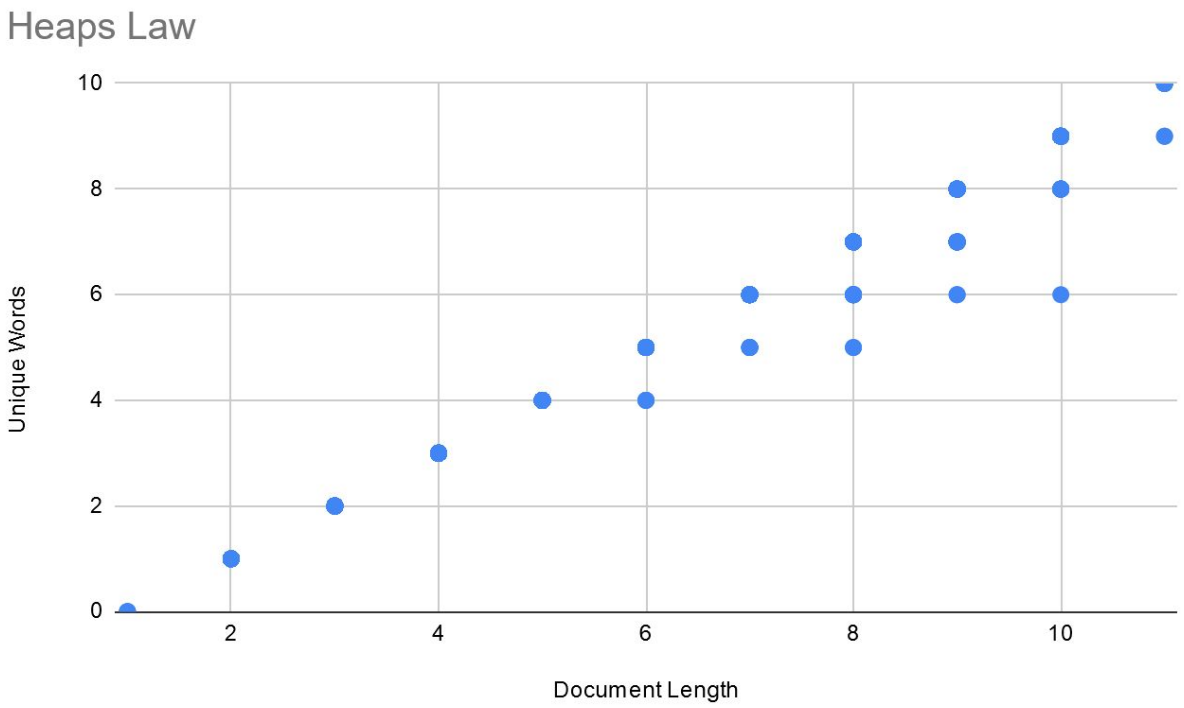
Also if you do not want one part of the analysis to run you simple need to comment out the function that does that part within my main function. By delegating to all these helper functions I think my code is very readable and also it is easy to customize the main function to produce exactly what you want. For example if you do not want to produce the txt files i used to make my Zipf and Heap charts, you can just comment out those function calls in the main. This also made debugging as I went a lot smoother.

Zipfs Law:



Some terms have extremely high frequency, while the majority of terms are found only 1-2 times. This supports Zipfs law because we can clearly see the difference between the most common terms and the less common ones. One way that this might help us is we could save the more frequent terms in memory, and the less frequent terms on disk, so that the most common stuff has the fastest access time.

Heaps Law:



As the document length increases we also see an increase in the # of unique words. We will never pass 1:1 of course, but we can clearly see there is a trend in this data that supports Heaps Law. I did this calculation without stopwords, but that would produce longer documents with fewer unique words I predict.