Ziyad Alrasbi Coursework Report

In this coursework, we were required to implement a hash map and a list word map using linked lists. In my program, I made sure that all the tests provided passed first, then I started to work on my own tests to make sure that my other functions in my program work too. In the end, all the test classes passed without error.

For my design, I chose to keep my code as simplistic as possible to not over complicate things and make me not understand what I am meant to do. In my ListWordMap and HashWordMap implementations, I chose to make my Entry class embedded in both to make it easier for me to work with. This could cause design notation issues however it still functions as it is meant to. In all my classes that I implemented, I made sure to briefly describe what each section of the code does to increase the readability of the code and to help readers understand how I implemented my code.

In my classes, I stuck to the interface provided. However, I also made my own functions to help me implement some sections of it. For example, in my HashMap class, I implemented my own functions to help me implement the other functions. I explained what each of those functions do in my code. I did this because the implementation of HashWordMap was difficult for me to begin with, but when I broke it down and started to understand it better, I was able to complete the implementation.

In my HashWordMap, I created 2 functions for checking if a number is a prime number and for getting the next prime number after it. This helped me implement my resize function, which was used for updating my load factor. It was essentially used for rehashing purposes. In my adding and removing functions, I declared a variable called deleted which helped me a lot in understanding the logic of hash codes and hash tables. I also declared a function called nextNonNullEntryIndex which essentially searches for the next index that isn’t empty which I used in my Iterator and hasNext functions. Implementing the hash codes at first was difficult, however when I understood how the prime numbers worked, I was able to implement them.

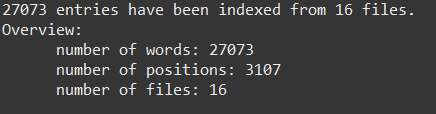
Implementing WordIndex and ListWordMap was straightforward.

Here are some examples of my code output:

For this one, I was not able to make it output as in the specification, however I made it specify how many cases of the word were found in the files:

https://i.gyazo.com/25a5590fad5b67ed2be8c758a3eae6d3.pnghttps://i.gyazo.com/acaf41b37265fa3715ee654f06afb2a5.png

Here is how “overview” is outputted as expected:

https://i.gyazo.com/5eddeb4fb636a344eae38dfe86d6bd44.png

­­­­­­­­

Here is how “remove” is outputted as expected:

https://i.gyazo.com/f7805deabc6277a55a2cce232e2e23d5.pnghttps://i.gyazo.com/e9e56f97548b27990b2f06a792f873e0.png

https://i.gyazo.com/587f865b07c09b04cf98480eab1731c5.pngSearching for a word that doesn’t exist outputs an error:

Here is how “add” outputs:

https://i.gyazo.com/159b6ffc58866ad68525be3f8e6470eb.pnghttps://i.gyazo.com/dd09b40fd1c0df46410a48a8dbe31e7c.png

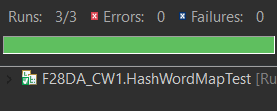
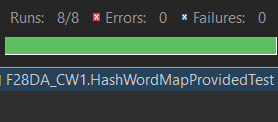
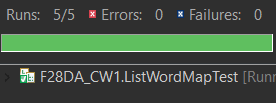
https://i.gyazo.com/5ed79f24d3b7227b39e98f9155109e36.pnghttps://i.gyazo.com/aa209cfcd88f9cefd510b2debc9e79db.png

Although my program runs as expected, there are some limitations to my code. For instance, when searching for a word, it only outputs the count of how many times it has occurred in each file rather than output exactly where it occurred in the files. This would make it harder for people to search for the words in the files because it gives a general idea of how many instances there are but not where the word occurs.

Another small limitation is that the “overview” command will return 0 for the count of words and positions if “addall” isn’t ran right before it. If addall is added before then it will work as it should.

A design notation issue my program could have is that I made many small functions inside each of my classes to help me write my code. This could result in longer running time for my application which would make the program slower. I could have prevented this by using less sub-programs and writing all my code in each function instead, however I struggled to understand it without making my own functions. When I faced issues in understanding how a section should work, I went to online resources such as YouTube to help me understand how it should work. YouTube was extremely helpful in helping me understand how hash maps should work. To clarify, all code written is my own and I only used the online resources to understand the theory.

Another issue I faced was being able to implement the comparison between the hash table and the linked list. I understood the theory behind it, however implementing it did not seem to work for me. What I did understand is that running the hash table would give a result in linear time, so in the worst case scenario for searching and adding, it would take O(N) time. Whereas for the linked list, the worst case scenario would be a quadratic function of O(N^2). So in a graph, a graph for the hash table would be a linear, straight line whereas for the linked list it would be a curve.

Finally, here is some proof of all the tests passing: