Lab5 report

**Introduction:**

In this lab we were given a file with a ton of words and numbers. Each word had 50 numbers that represented the word in a certain way. So, the objective of this lab was to read the file, store the word along with the 50 numbers that represent the word in both a binary-search tree and a hash table. We also had to take note of the times for each and compare the two times of the binary-search tree and the hash table. We were also given the task to read a separate file with two columns of just words, search for the words in the binary-search tree or hash table, and then compare the similarities of the two words by using the 50 numbers that represent the word itself.

**Proposed Solution:**

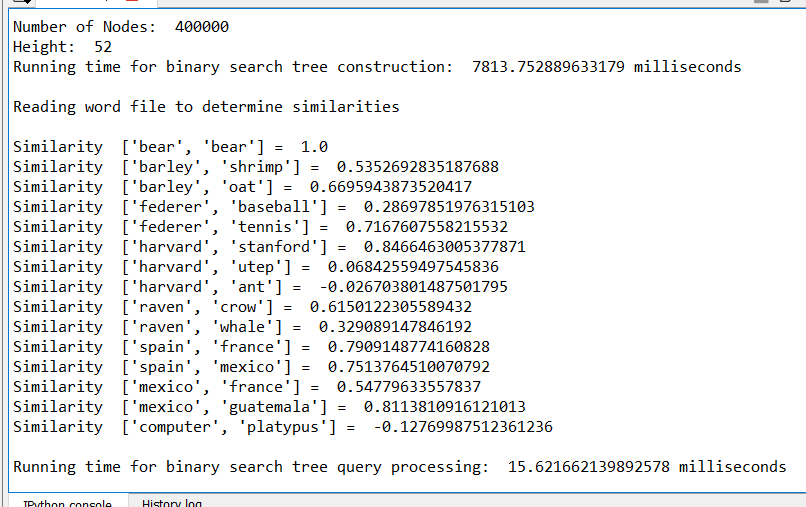
Right at the very beginning, I read the file then store the whole file in a list. Each line in the file represents an element in the initial list I make at the beginning. After making a very large list, I made a method that breaks up each item in the large list into a list containing two elements, the word, then an array of the 50 numbers that represent the word. So, at the beginning I make a huge list with each element being a list itself of length two, containing the word and the 50 numbers. To insert the data in a binary-search tree I use the same insert method that was provided in class, with minor modifications, so I insert the items from the list I created at the beginning of my program. Because each item in the list I created is list of length two, that means that when I am inserting an item in the tree, I am actually inserting a list of length two. When determining if the inserted item goes to the left or right in the tree, I compare the string values of the new item being inserted (which would be the first item in the list) and the present node in the tree. If the new item being inserted is less than the current node, the new item will go to the left, vice versa, the new item goes right.

Creating the hash table was much like the how I did the binary-search tree. I used the items from the list I made at the beginning. To identify the bucket that’s supposed to be used I had to get the ascii value of each word and then divide that by the length of H.item, and once I identified the correct bucket I append the new item to hash table with the address of whatever the bucket value may be. The way the chaining works in this case is when words have the same bucket value, the item still gets appended, the list at that bucket value will continue to grow as long as they have matching bucket values.

When it came to finding the similarities between two words, this also includes for both methods of storing (binary-search tree, and hash table), only the list of 50 numbers that represent each word are being used. The formula was provided, it required that the numerator be all the values in one word be multiplied with all the values in the other word. Then the denominator required that the 50 numbers be squared then multiplied to the other 50 numbers from the other word.

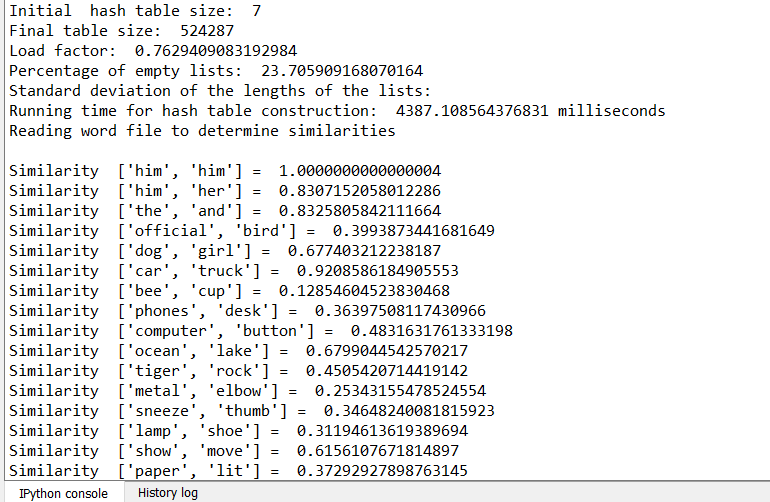
**Experimental Results:**

For the purpose of experimenting, I used different files with different words on them when it came to testing the similarities between two words. This was to make sure that my results were similar to the ones provided, and also to make sure that my methods could find any word on the extremely large file. Below are some screenshots of my results.



This was for the binary-search tree, and the words being used to compare similarities are the same words provided in this lab to make sure my results match those of the ones on the handout.

Below is another screenshot of my results, but this time I used different words to compare similarities.



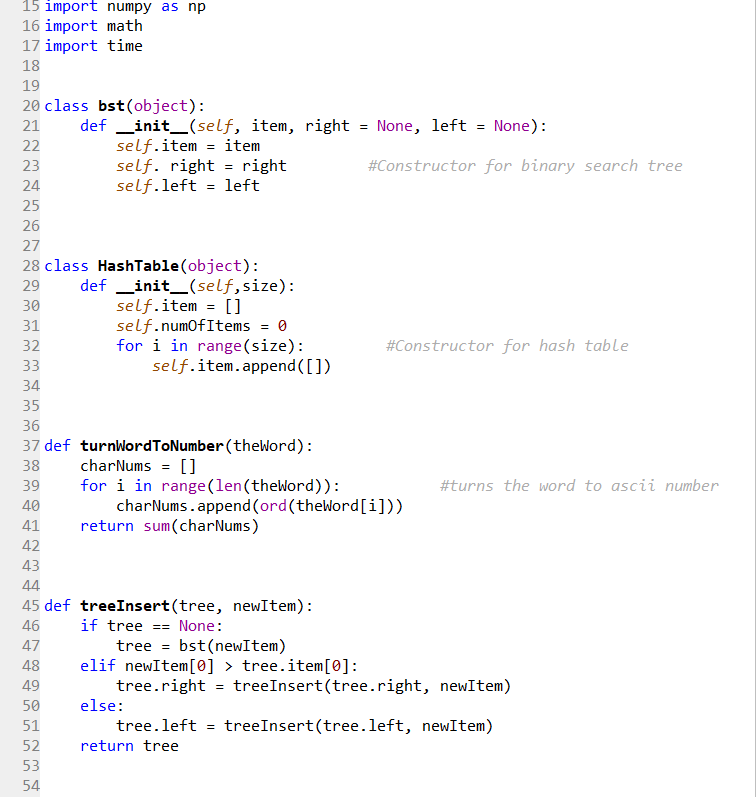
Below is a graph that represents the runtimes of the construction of a binary-search tree and hash table using different input sizes.

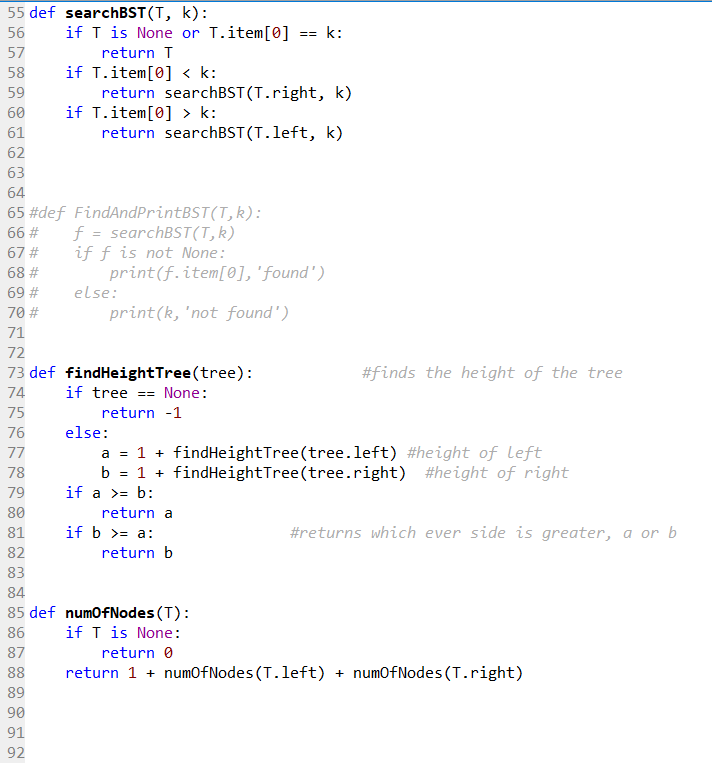
Just by looking at the graph, I can clearly see that the hash table is much faster than the binary search tree. Both the binary-search tree and the hash table appear to have O(logn) run times, because their time is increasing at a very slow pace as the number of inputs increases greatly. (The number of inputs is the number of lines that are being taken from the file, with 400,000 being the whole file)

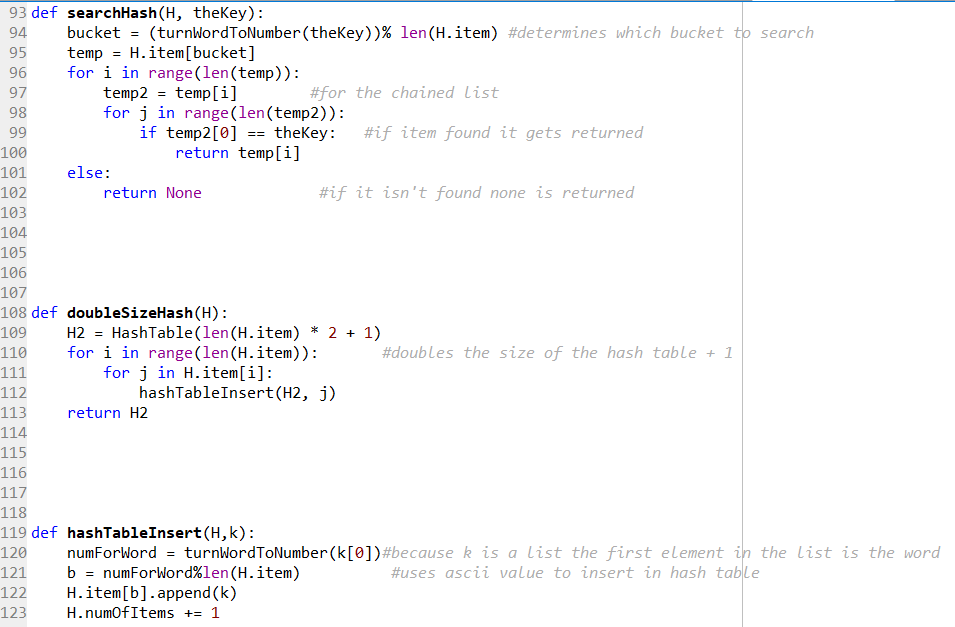
**Conclusion:**

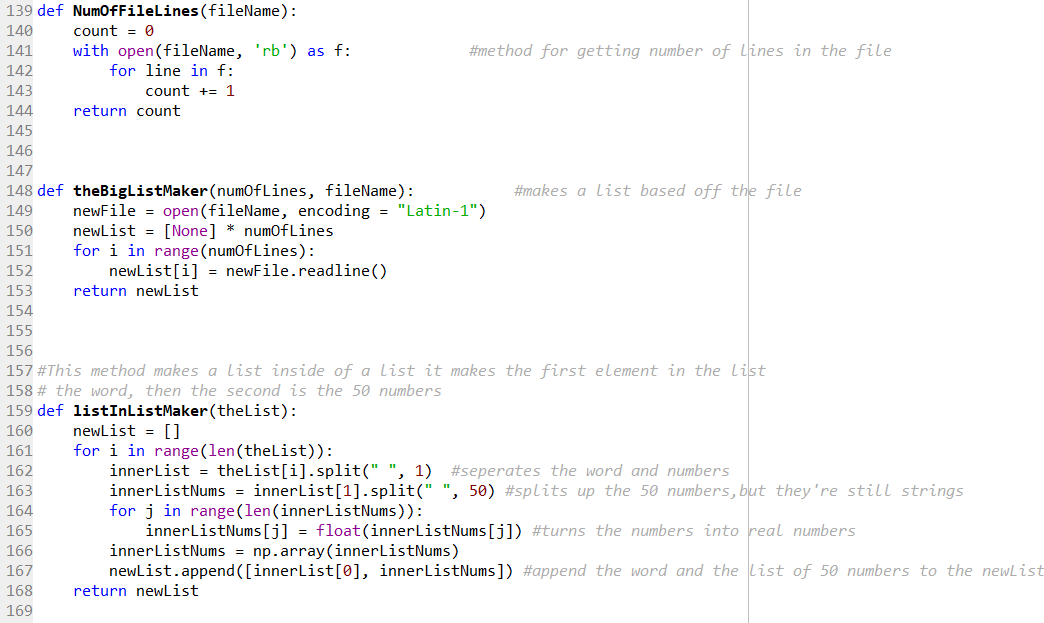
I learned how to read lines from a file and store them both a binary-search tree and a hash table. I also learned how to break up a single string using .split(), that was extremely helpful when it came to separating the words from the numbers and separating the 50 numbers as well. If also learned that hash tables with chaining are a lot faster than binary-search trees. I figured out that you can compare strings without having to change them to numbers in python. To top it all off, this lab really gave me some great practice at moving through lists, trying to find the right word, when it came to searching for it in either the binary-search tree or the hash table.

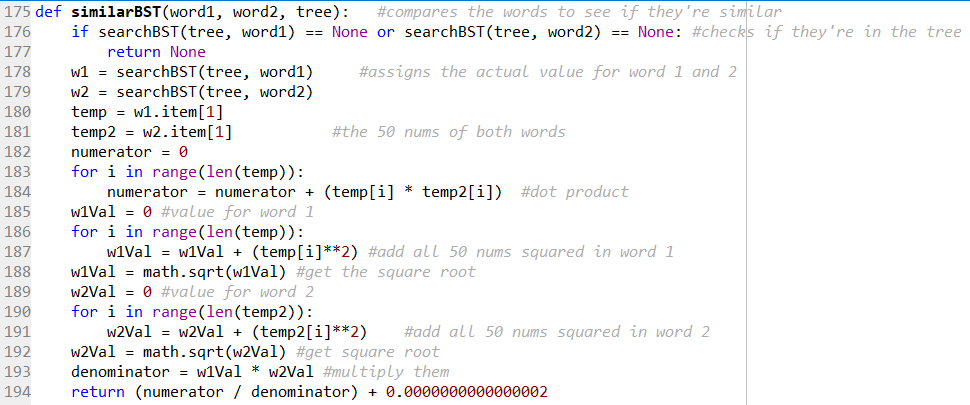
**Appendix Source Code:**

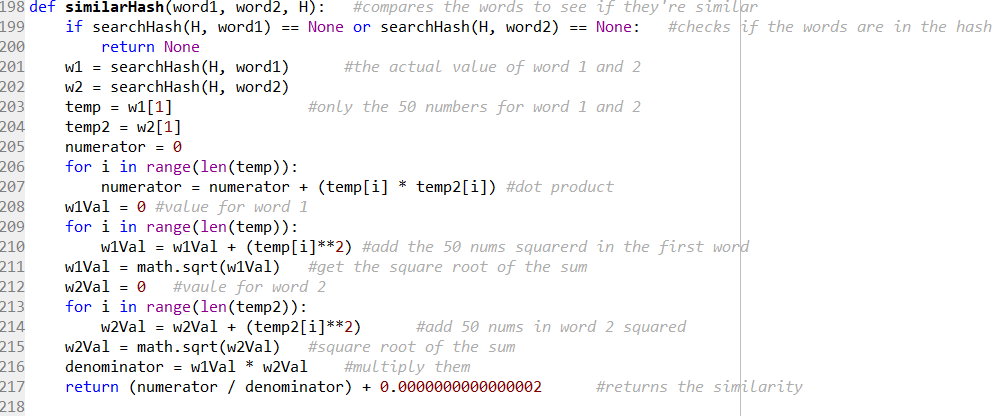


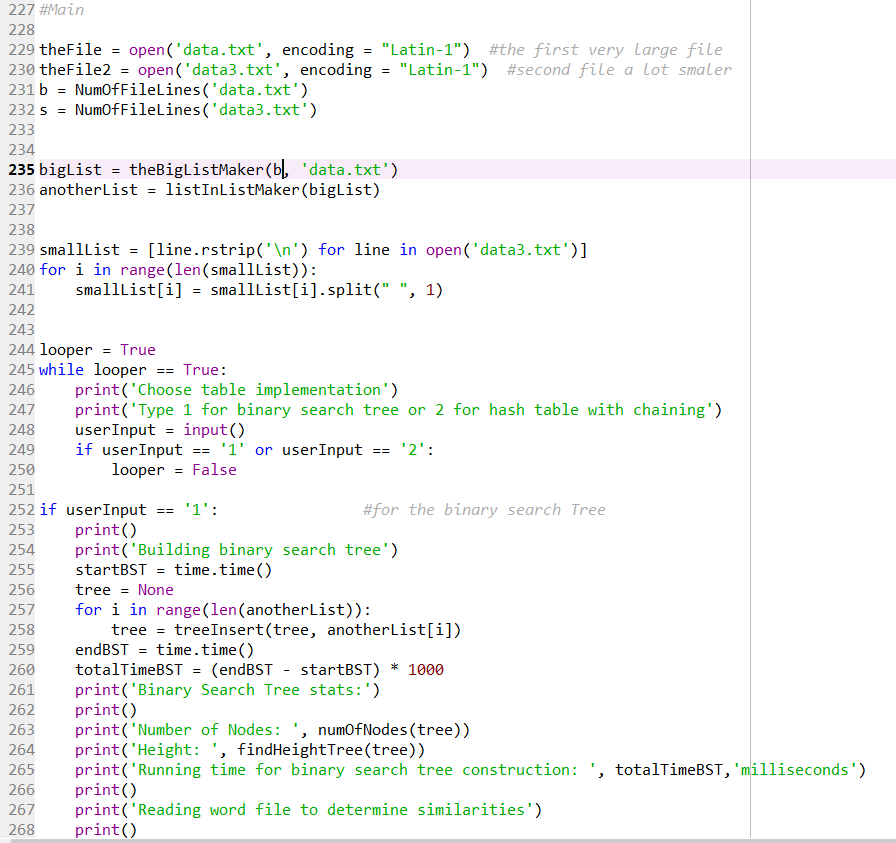


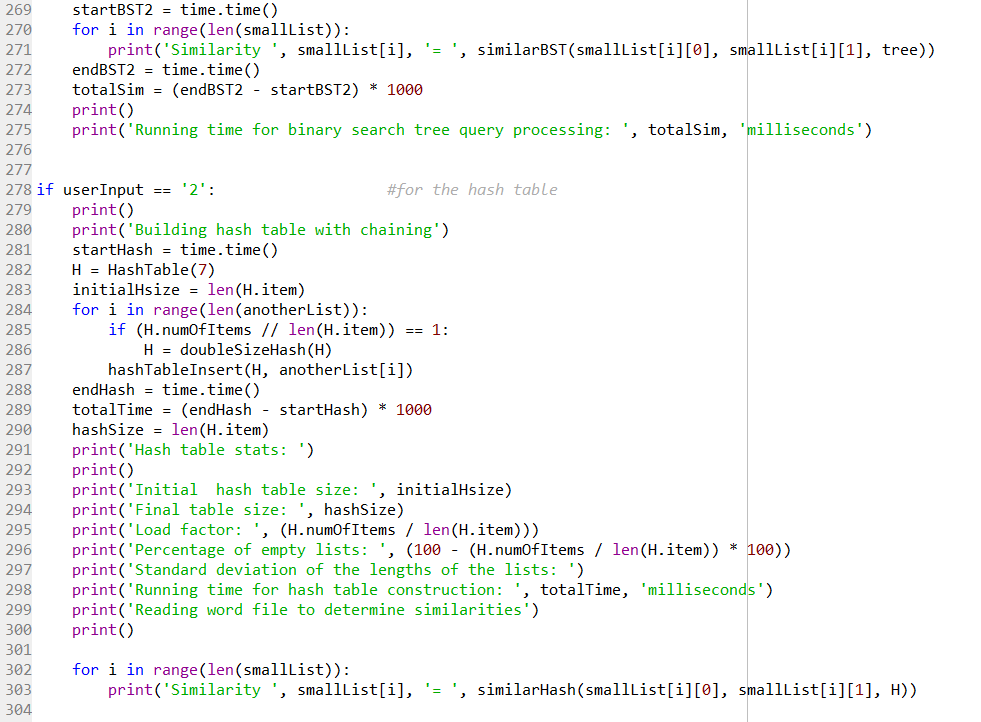












“I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.”

Joey Roe