CS 2302 - Lab 5

Sofia Gutierrez

Olac Fuentes

**Introduction**

The objective of this lab was to create a program that compared the running times of two data structures, Hash Tables with chaining and Hash Tables with linear probing. These two data structures were given a list of words that are most commonly used in the English language. Each of these words were accompanied by floating point numbers which served as the word embedding. This word embedding represented the word’s vector description or in other words, these numbers would represent the word. After the two data structures were created and were each filled with this list of words and their numbers, a second, separate file, which contained a list of more words, was created by the user. More specifically, this list had two words per line. The point of this was so that each of the two words per line would be compared to one another and their similarities would print. A cosine distance formula was used which would then help produce this similarity output. Again, this would then compare the running times that the outputs took to generate for both data structures.

**Design and Implementation**

This program had three sections which were added to the previous lab (BST and B-tree). The word embedding class would of course still contain the attributes which were ‘word’, and ‘emb’. These were needed in order to modify the basic methods which belong to the classes Hash Tables with chaining and Hash Tables with linear probing (insert method, find method, etc.). Then came the Hash Tables with chaining class which contained its appropriate attributes and methods: ‘insert’, ‘find’, and ‘h’, and later ‘string\_lengthHTC’, ‘asciiHTC’, ‘product\_asciiHTC’, ‘sum\_assciiHTC’, and, ‘recursiveHTC’. The ‘h’ method would return the index that corresponds to the item, in other words it looks up where the item should be placed. Then came ‘insert’ which would insert k in the appropriate bucket using ‘h’ but first looks to see if k, the item, is already in the bucket which, if it was, it would do nothing. Then came ‘find’ which would just recturn the bucket and index. The method ‘string\_lengthHTC’ returned the length of the string % n. ‘asciiHTC’ returned the first character of the string % n. ‘product\_asciiHTC’ returned would return the first and last characters of the string % n. ‘sum\_assciiHTC’ sums all the characters in the string given and returns the sum % n. ‘recursiveHTC’ returns (ord(S[0])+255\*h\_recursive(S[1:],n))%n.

Next was the Hash Table with linear probing class. The methods included ‘hHtlp, ‘insertHtlp, ‘findHtlp, and later ‘length\_stringHTLP’, ‘asciiHTLP, ‘product\_asciiHTLP, ‘sum\_asciiHTLP, and ‘recursiveHTLP’. ‘hHtlp’ would again find the correct placement of where to store k within the hash table. ‘insertHtlp’ would again store the k, the item, in its correct placement using the method ‘hHtlp’. And then finally, ‘findHtlp’ which returned the position of k in the table or -1 if it was not found within the table. The methods ‘length\_stringHTLP’, ‘asciiHTLP, ‘product\_asciiHTLP, ‘sum\_asciiHTLP, and ‘recursiveHTLP’ were the same as those implemented within the Hash Table with chaining class.

And finally, the last section was coded which contained the main method. This main method was split into two sections, one so that the Hash Table with chaining class would operate and the other so that the Hash Table with linear probing class could operate. Within the chaining section the larger file that was given would be opened and each line would be read and stored within the table. Next, the second file, the one created by the user, would be read and the words within this smaller file would be searched within the table all so the similarity of the two words given could be printed for the user. Again, the running time to compute the similarities would be displayed for the user.

Then came the next section which involved the Hash table with linear probing. Again, each line from the larger file would be read and the data list of the tree was filled with a list of the word embedding objects which would be sorted alphabetically with the help of the functions found within the Hash table with linear probing class. Then the second file was read through. Again, this file contained a list of two words. The two words were each stored as objects of the word embedding class to accommodate to the search function found within the linear probing class. The search function would return their location and the cosine distance formula would compute. The similarities were then printed out along with the time it took for these similarities to run.

**Experimental Results**

|  |  |  |
| --- | --- | --- |
| Number of Word Pairs | Hash Table with chaining | Hash Table with linear probing |
| 3 | 15 minutes | 2 hours |
| 5 | 30 minutes | 2 hours+ |
| 10 | 1 hour | 3 hours+ |

The running times for both chaining and linear probing did not compare. The same file was used for both and both data structures were tested with 3 to up to 10 words. The hash table with chaining had a faster time than the hash table with the linear probing as the table above clearly depicts.

|  |  |  |
| --- | --- | --- |
| Number of Word Pairs | Binary Search Tree (BST) | B-Tree (max\_data = 5) |
| 15 | 0.007227897644042969 | 0.007316112518310547 |
| 30 | 0.01764512062072754 | 0.022891759872436523 |
| 50 | 0.022414684295654297 | 0.030154943466186523 |

|  |  |  |
| --- | --- | --- |
| Number of Word Pairs | Binary Search Tree (BST) | B-Tree (max\_data = 3) |
| 15 | 0.0038268566131591797 | 0.0189361572265625 |
| 30 | 0.023446083068847656 | 0.015089035034179688 |
| 50 | 0.02054572105407715 | 0.028614044189453125 |

The figures above display the running times for both data structures used in this program with a different max\_data for B-tree. This shows that both Binary Search Tree and B-Tree have running times that are very similar however, when taking a closer look at the time the BST has a slightly faster running time than the B-tree when the max\_data was set to 5. The B-tree had a slightly better time than the BST when the max\_data was set to 3.

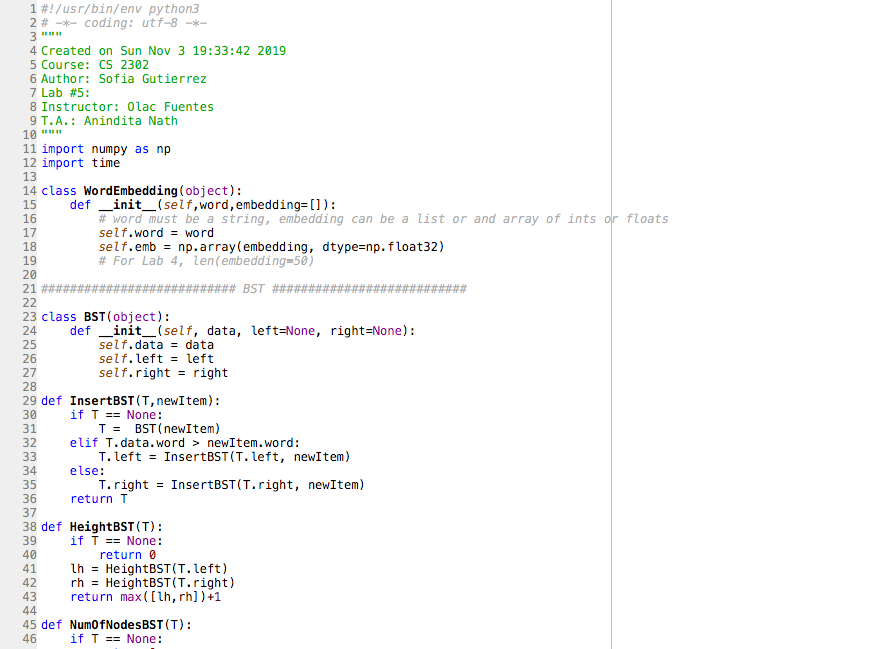
|  |  |  |
| --- | --- | --- |
| Hash functions | Hash Table with chaining | Hash Table with linear probing |
| 1 | 0.04824588613136718 | 0.001788903604317968 |
| 2 | 0.02059386690432781 | 0.002957594750321257 |
| 3 | 0.0533789768044869 | 0.002239530475308859 |
| 4 | 0.0522845739091138 | 0.00407050348504984 |
| 5 | 0.0129488458395485 | 0.00185741750619385 |

As stated earlier both Hash Table classes had functions that were added to them and were expected to be compared. Both classes had similar running times for these functions but the Hash Table with chaining had slightly quicker times than the Hash Table with linear probing.

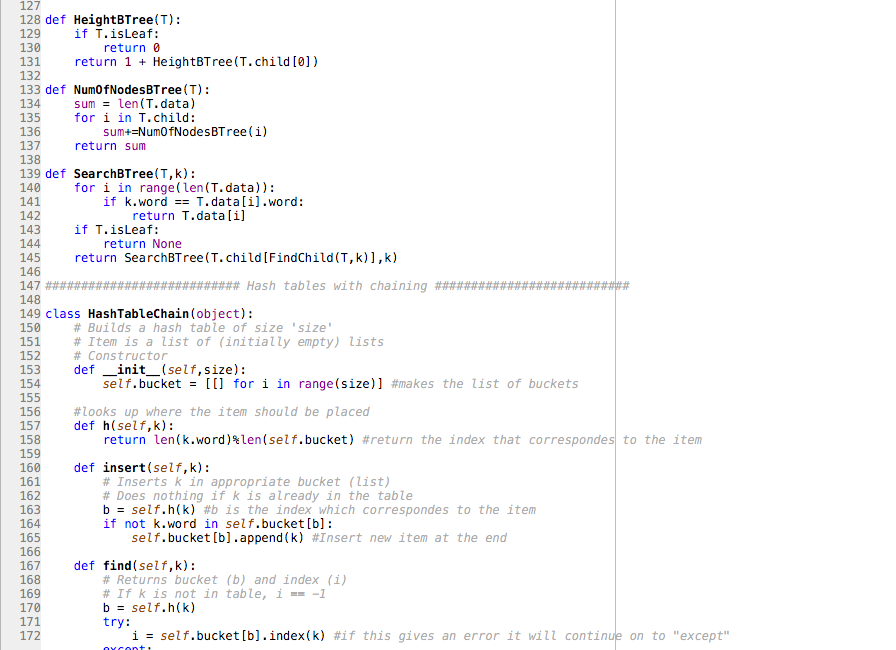
**Conclusion**

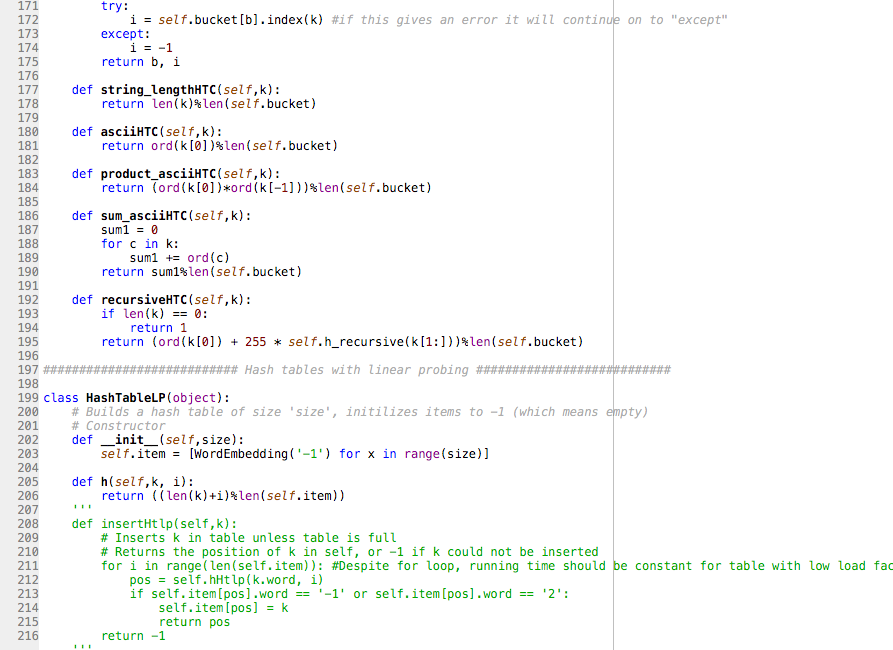
This program successfully completed its task which again was to compare the running times of two data structures, Hash Tables with chaining and Hash Tables with linear probing. In this lab, I became more familiar with the basic functions found within both Hash Tables. I also became more comfortable with creating classes, attributes, and objects. The most trouble I had during this lab was accommodating the insert function for both data structures to the word embedding, but once I became more familiar with objects and their purpose I was able to successfully perform this task.

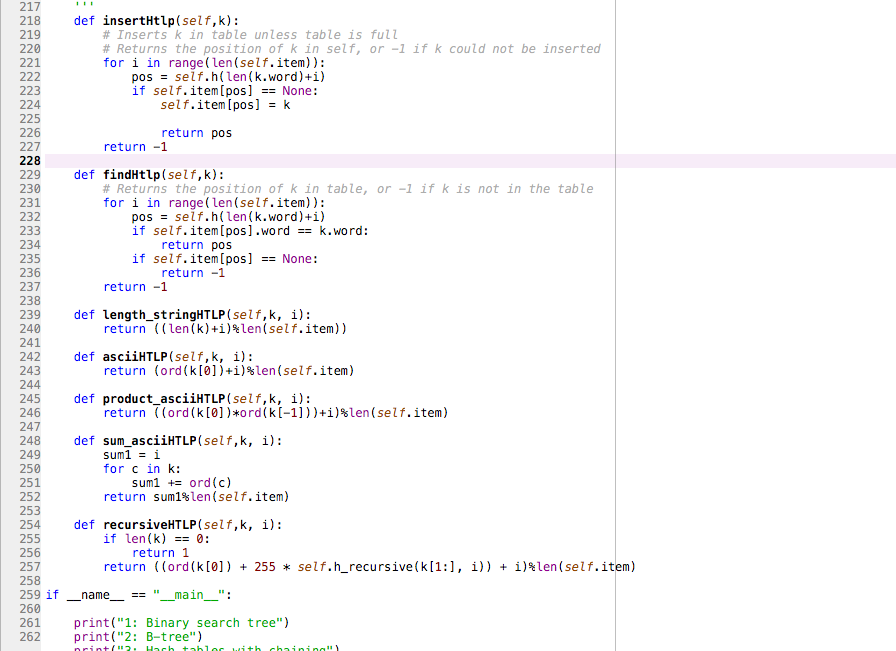
**Appendix**

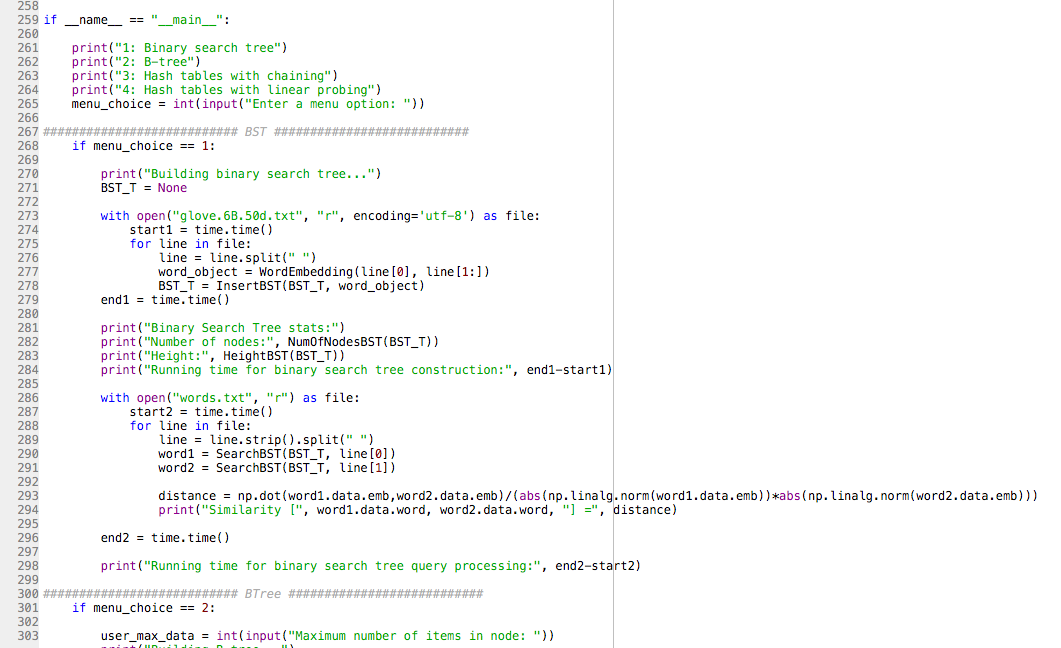


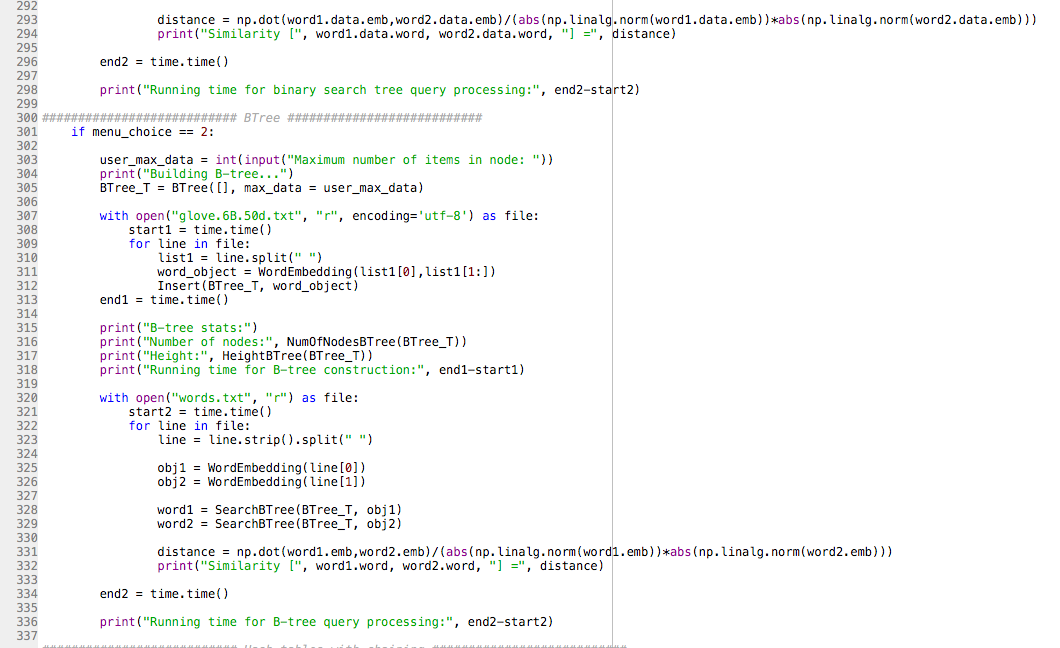




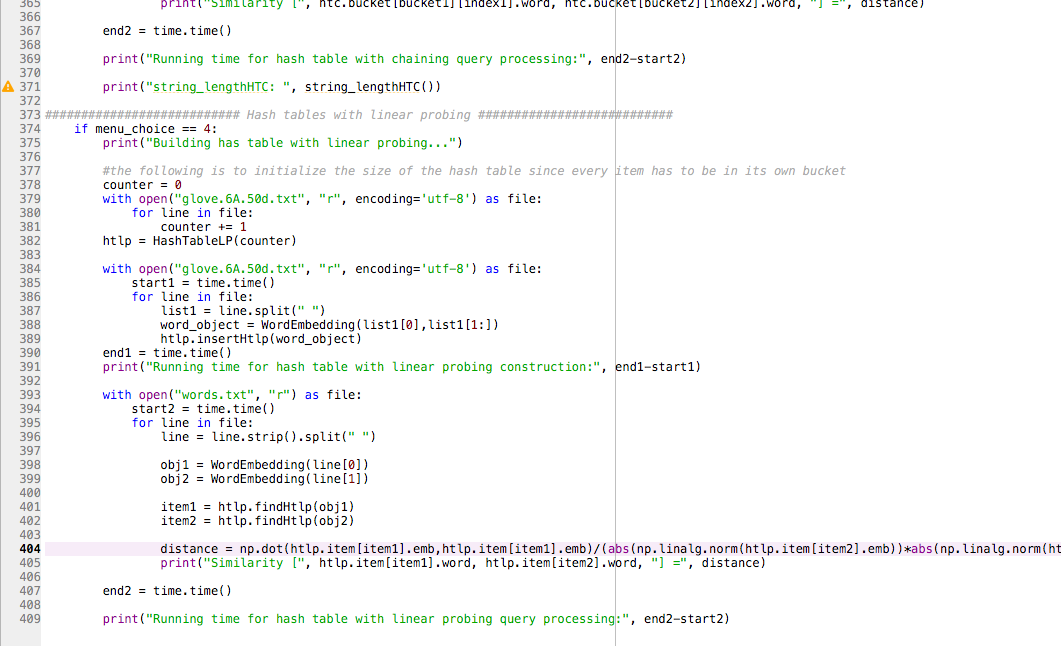












**Academic Honesty Certification**

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.

Sofia Gutierrez November 10, 2019