**CS 2302 Data Structures**

**Fall 2019**

**Lab Report #5**

Due: 11/1/2019

Professor: Olac Fuentes

TA: Anindita Nath

**Introduction**

For this lab we were asked to add more variety with word embeddings. We were asked to store word embeddings in both chaining and linear hash tables. For this lab it is key to remember how to access elements from both hash tables. The main objective of this lab is to get a deeper understanding of how chaining hash tables and linear hash tables work. With adding these two tables onto Lab 4, it gives a better understanding of which one is faster to access data from.

**Proposed Solution Design and Implementation**

**Hash Tables with Chaining:**

* Length of the string % n:
* For this operation, I called the method insert in which I called the method h. In method h, I got the length of the word and divided it by how many buckets there are. I then inserted the embedding into that bucket.
* The ascii value (ord(c)) of the first character in the string % n:
* For this operation, I approached it the same way I approached the first bullet. I sent to method h the word and converting the first character into its ascii value (ord(c)) and then inserted the embedding into that bucket.
* Product of the ascii values of the first and last characters in the string % n:
* For this operation, I approached it the same way I approached the first bullet. I sent to method h the word and then multiplied the first character ascii value (ord(c)) by the last character ascii value (ord(c)) and then inserted the embedding into that bucket.
* Sum of the ascii values of the characters in the string % n:
* For this operation, I approached it the same way I approached the first bullet. I sent to method h the word and then added all of the characters ascii value (ord(c)) and then inserted the embedding into that bucket.
* Recursive formulation:
* For this operation, I approached it the same way I approached the first bullet. I sent to method h the word and then checked if the word was empty or not. If the word was not empty, I made a recursive call, in which I returned the rest of the word except the first character. I then inserted the word embedding into that list.
* Function of my choice:
* For this operation, I approached it the same way I approached the first bullet. I wanted to get the absolute value of the last character ascii value (ord(c)) minus the first character ascii value (ord(c)) then insert the embedding into that list.

**Hash Tables with Linear:**

* Length of the string % n:
* For this operation, I called the method insert in which I called the method h. In method h, I got the length of the word and divided it by how many items there are. I then inserted the embedding into that bucket.
* The ascii value (ord(c)) of the first character in the string % n:
* For this operation, I approached it the same way I approached the first bullet. I sent to method h the word and converting the first character into its ascii value (ord(c)) and then inserted the embedding into that item.
* Product of the ascii values of the first and last characters in the string % n:
* For this operation, I approached it the same way I approached the first bullet. I sent to method h the word and then multiplied the first character ascii value (ord(c)) by the last character ascii value (ord(c)) and then inserted the embedding into that item.
* Sum of the ascii values of the characters in the string % n:
* For this operation, I approached it the same way I approached the first bullet. I sent to method h the word and then added all of the characters ascii value (ord(c)) and then inserted the embedding into that item.
* Recursive formulation:
* For this operation, I approached it the same way I approached the first bullet. I sent to method h the word and then checked if the word was empty or not. If the word was not empty, I made a recursive call, in which I returned the rest of the word except the first character. I then inserted the word embedding into that item.
* Function of my choice:
* For this operation, I approached it the same way I approached the first bullet. I wanted to get the absolute value of the last character ascii value (ord(c)) minus the first character ascii value (ord(c)) then insert the embedding into that item.

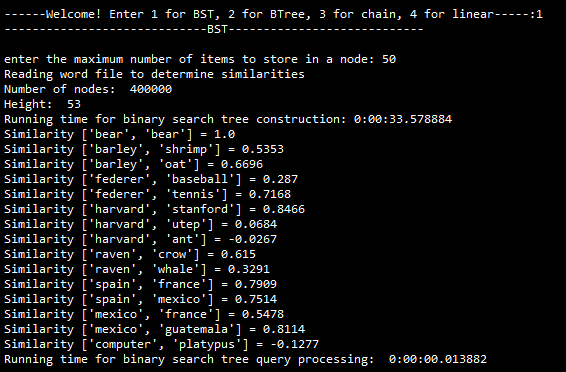
**Experimental Results**

**Hash Tables with Chaining:**

* Length of the string % n:
* For this operation, it was difficult to understand how to access data and how to save a word embedding instead of a number. I wanted to compare the words that were inside the bucket with the word being searched. I then realized it was a good idea and then implemented it.
* The ascii value (ord(c)) of the first character in the string % n:
* For this operation, I approached it the same way that I approached the first bullet of **Hash Tables with Chaining** under **Experimental Results**. But this one I made one thing differently. I wanted to only access the first character of the word being searched and checked if the word embedding was inside of that bucket.
* Product of the ascii values of the first and last characters in the string % n:
* For this operation, I approached it the same way that I approached the first bullet of **Hash Tables with Chaining** under **Experimental Results**. But this one I made one thing differently. I took the first letter of the word and the first letter of the word and multiplied them. This wasn’t so difficult to do since this was very similar to the second bullet of **Experimental Results**.
* Sum of the ascii values of the characters in the string % n:
* For this operation, I approached it the same way that I approached the first bullet of **Hash Tables with Chaining** under **Experimental Results**. But this one I made one thing differently. This was very similar to the second and third bullet of **Experimental Results**. What I did differently was add a for loop in which I accessed every character of the word and summed up all of the ascii values so I could access the bucket it belonged in.
* Recursive formulation:
* For this operation, I approached it the same way that I approached the first bullet of **Hash Tables with Chaining** under **Experimental Results**. But this one I made one thing differently. For this I confused the h that the formula had. I believed it was some variable that I needed. I realized that the recursive formula was already given to us and all I did was put the formula as a recursive call.
* Function of my choice:
* For this operation, I approached it the same way that I approached the first bullet of **Hash Tables with Chaining** under **Experimental Results**. But this one I made one thing differently. I did something similar as the 2nd, 3rd, and 4th bullet. I took the last character of the word (ascii value) and the first character of the word (ascii value) and subtracted them and got the absolute value so it wouldn’t be negative.

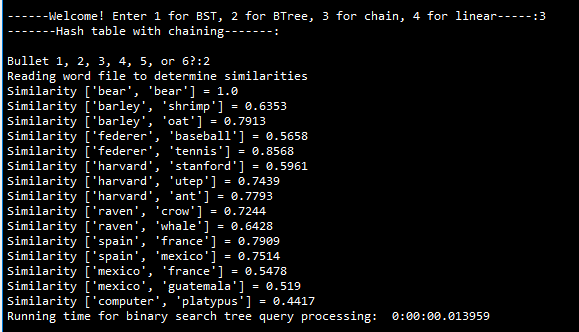
**Hash Tables with Linear:**

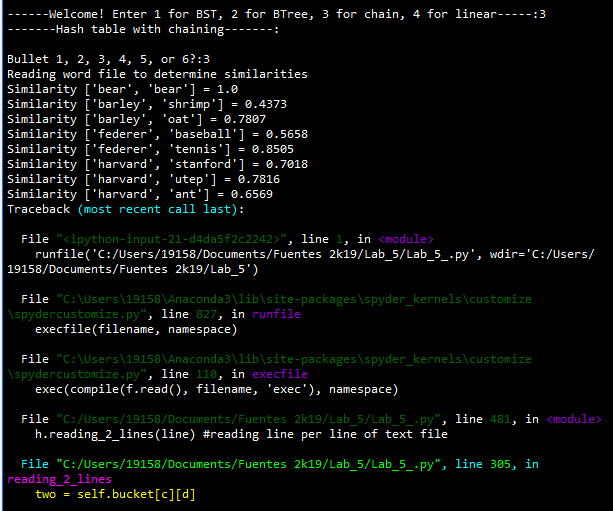
* Length of the string % n:
* For this operation, I did the same thing as I did for Chaining. The only thing different was to access every item in the list and check if the thing I was looking for was there. If the item wasn’t there, I moved the index and checked if the number was there and so on. For the way that I did this was to get the length of the word and checked if the length of the word was inside of the item inside of the table, and if not, it moved indexes.
* The ascii value (ord(c)) of the first character in the string % n:
* For this operation, I approached it the same way I approached bullet 1 under **Hash Tables with Linear**. The only thing different was to get the first character of the string and get its ascii value and checked if the word was inside of the item and if not, it moved indexes.
* Product of the ascii values of the first and last characters in the string % n:
* For this operation, I approached it the same way I approached bullet 1 under **Hash Tables with Linear**. The only thing different was to get the ascii values of the first and last character of the word and multiply them. I then looked for the item and if the item I was looking for wasn’t there, I moved indexes.
* Sum of the ascii values of the characters in the string % n:
* For this operation, I approached it the same way I approached bullet 1 under **Hash Tables with Linear**. The only thing different was to make a for loop in which I traversed the entire word and summed all of the character ascii values. I accessed the item where the word was and if it was not there, I moved indexes.
* Recursive formulation:
* For this operation, I approached it the same way I approached bullet 1 under **Hash Tables with Linear**. The only thing different was to approach it the same way I approached bullet 5 under **Hash Tables with Chaining.** I just used the formula that was given to us and returned the string without the first character and so on.
* Function of my choice:
* For this function, I did the same as I did in **Hash Tables with Chaining.** I just subtracted the ascii value of the last character minus the ascii value of the first character. I then got the absolute value of that and looked for the word. If the word wasn’t there, I moved indexes.

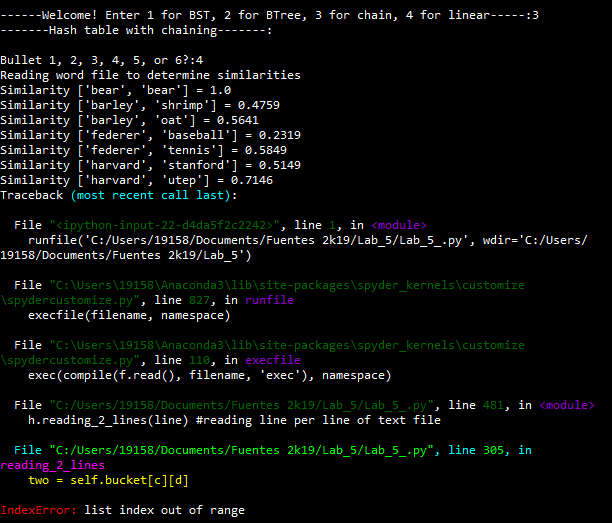


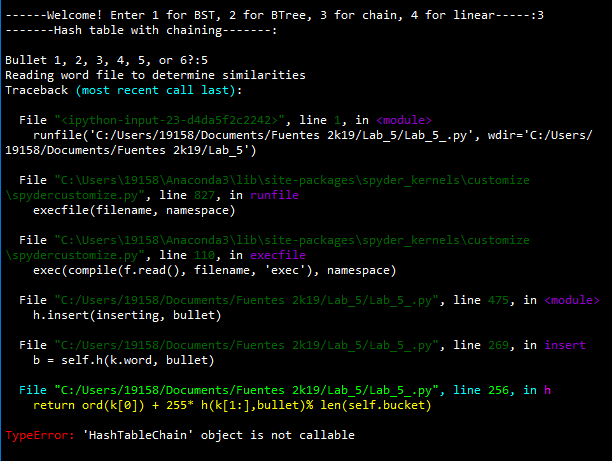


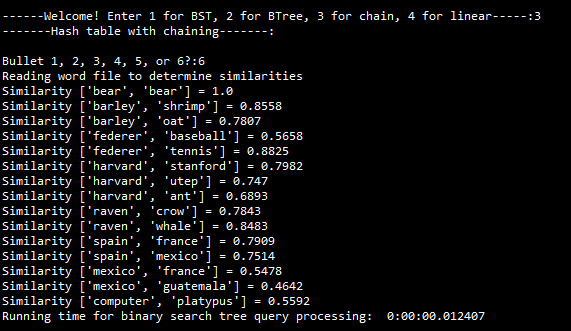


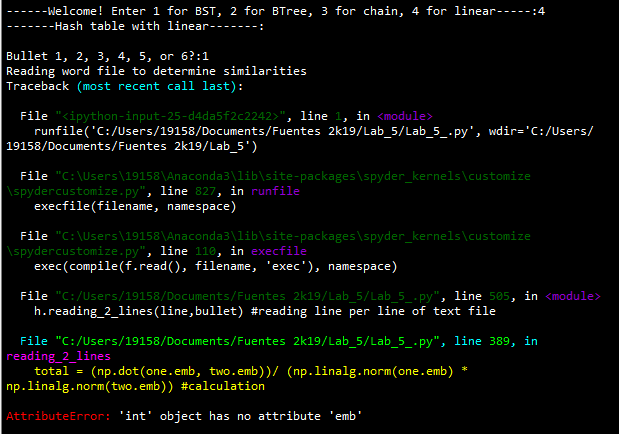












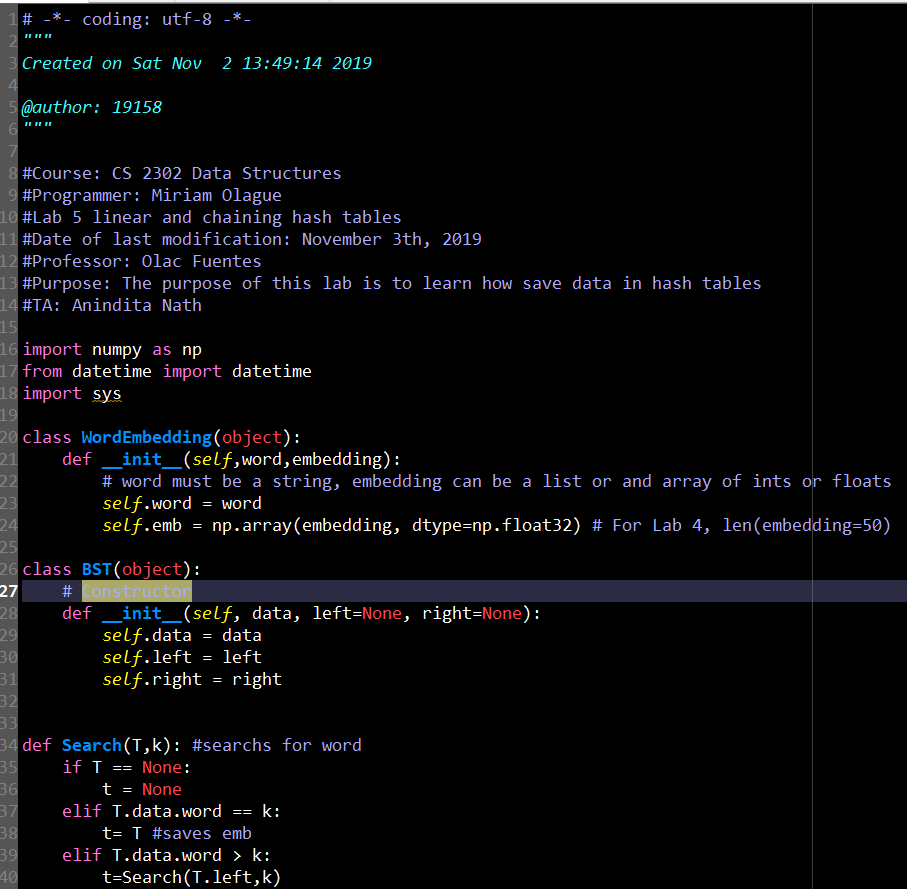
This same error appears for the rest of the method I made for LP.

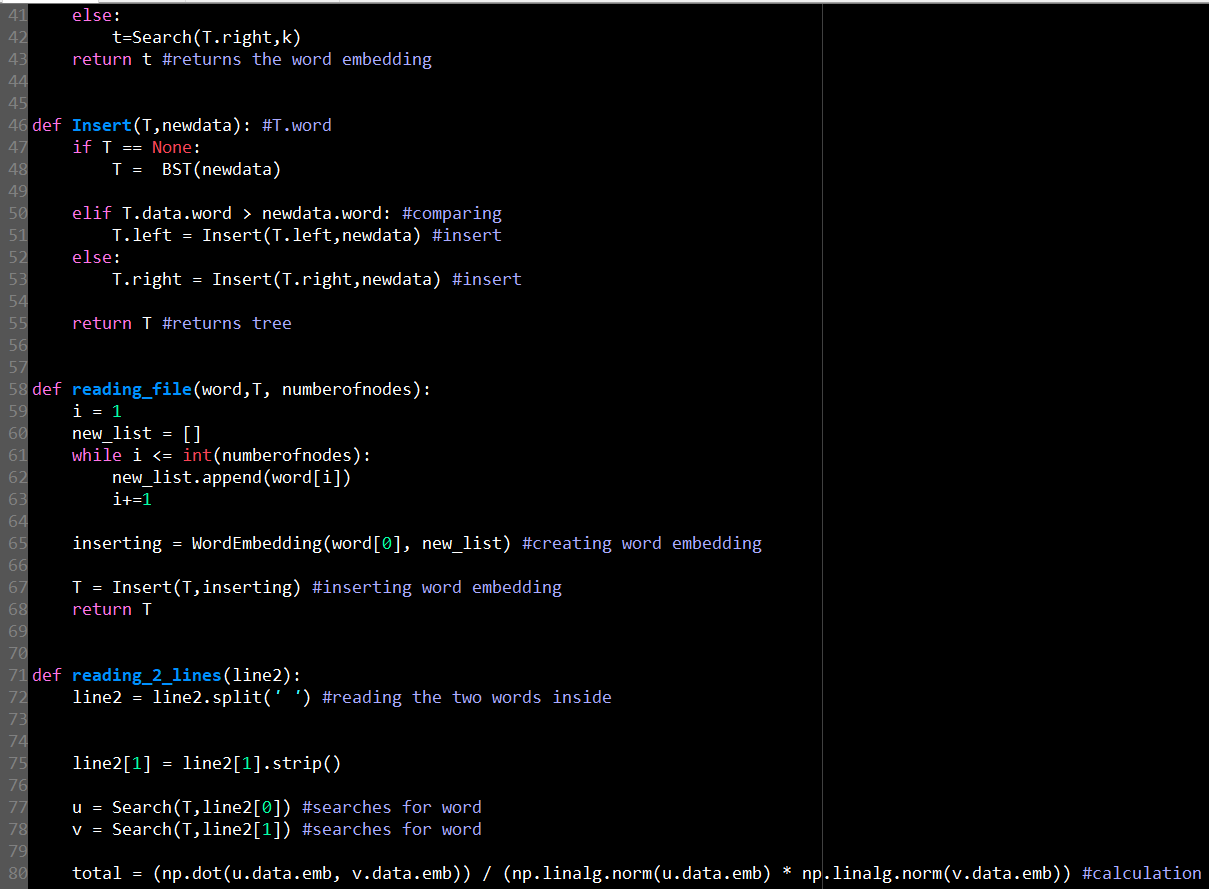
As the results show, I could not test Linear Probing because of issues I had. However, the amount of time that

**Conclusion**

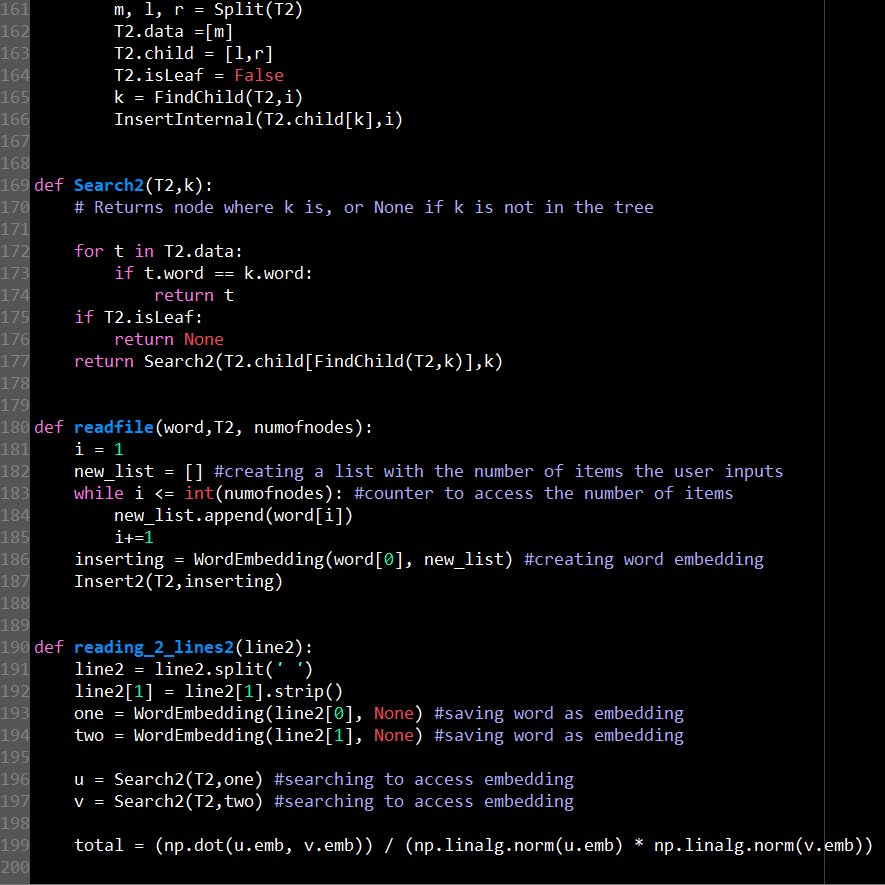
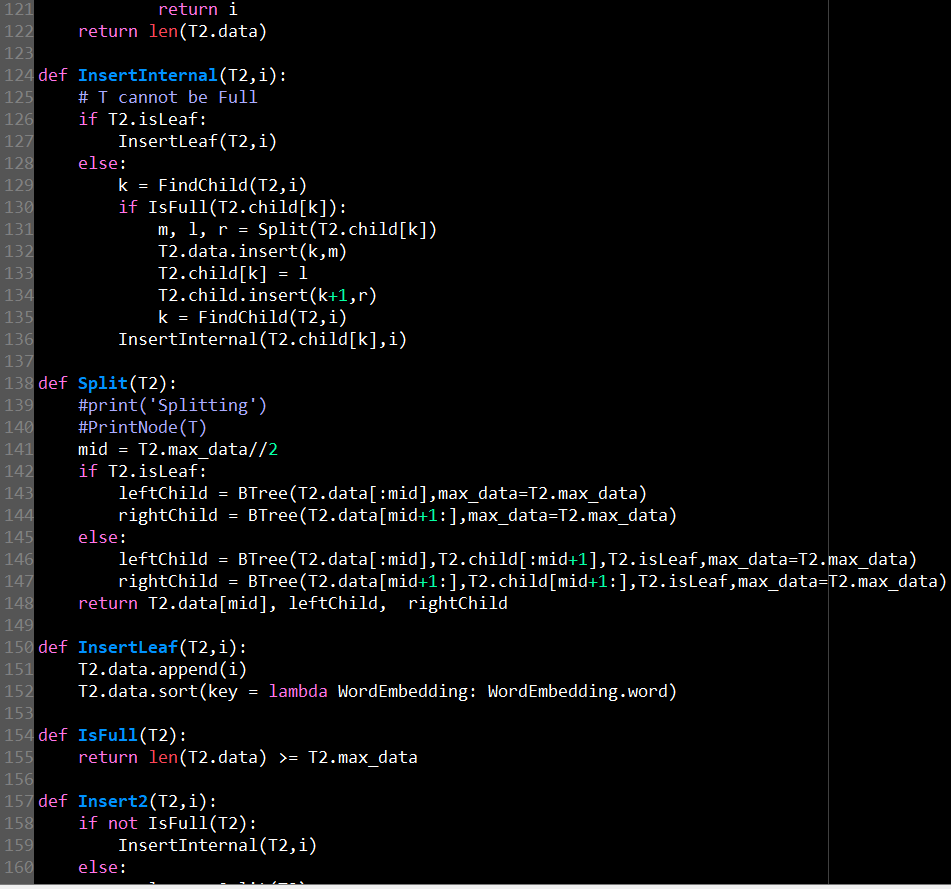
Since the word embeddings represent words, it makes it very efficient to access data of that word very quickly. Binary Search Trees (BST) were easy to understand when compared to B-Trees. I learned a lot about B-Trees and learned how to access data from a BST or a B-Tree. This lab helped me understand what the difference between B-Trees and BSTs. Hash tables were easy to think through what you wanted to do. However, modifying it so that your methods only see the word of the embedding was tricky.

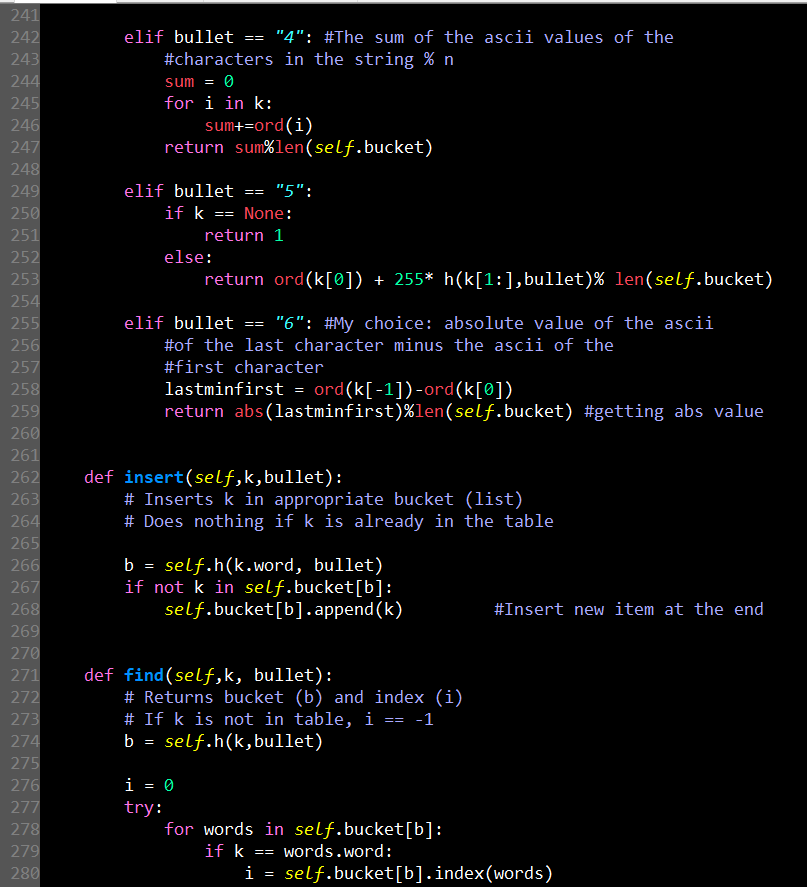
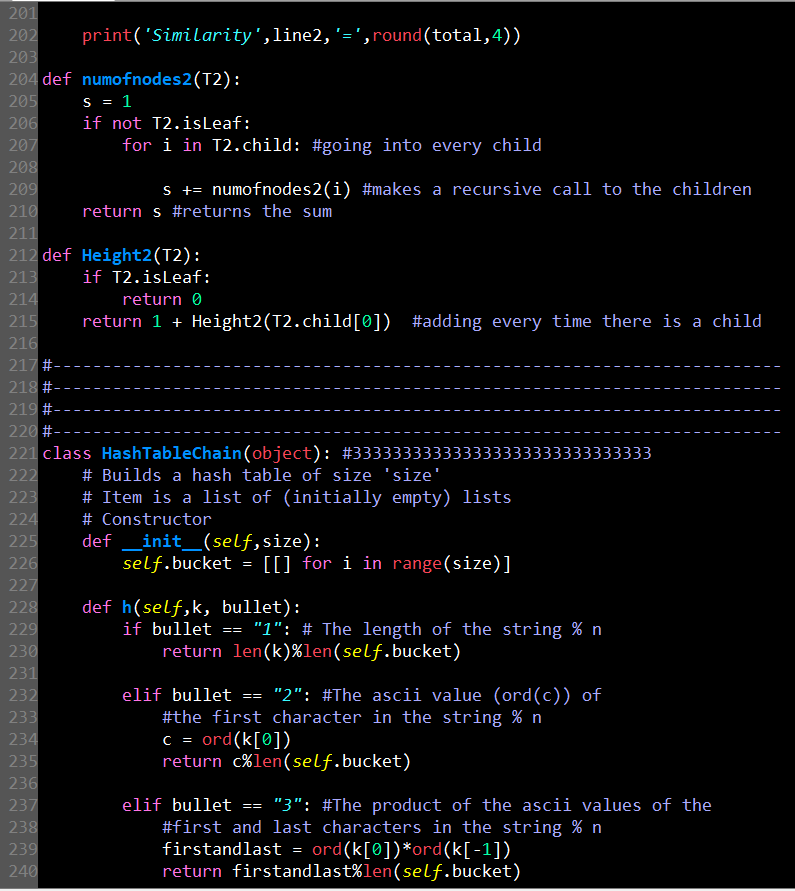
**Appendix**

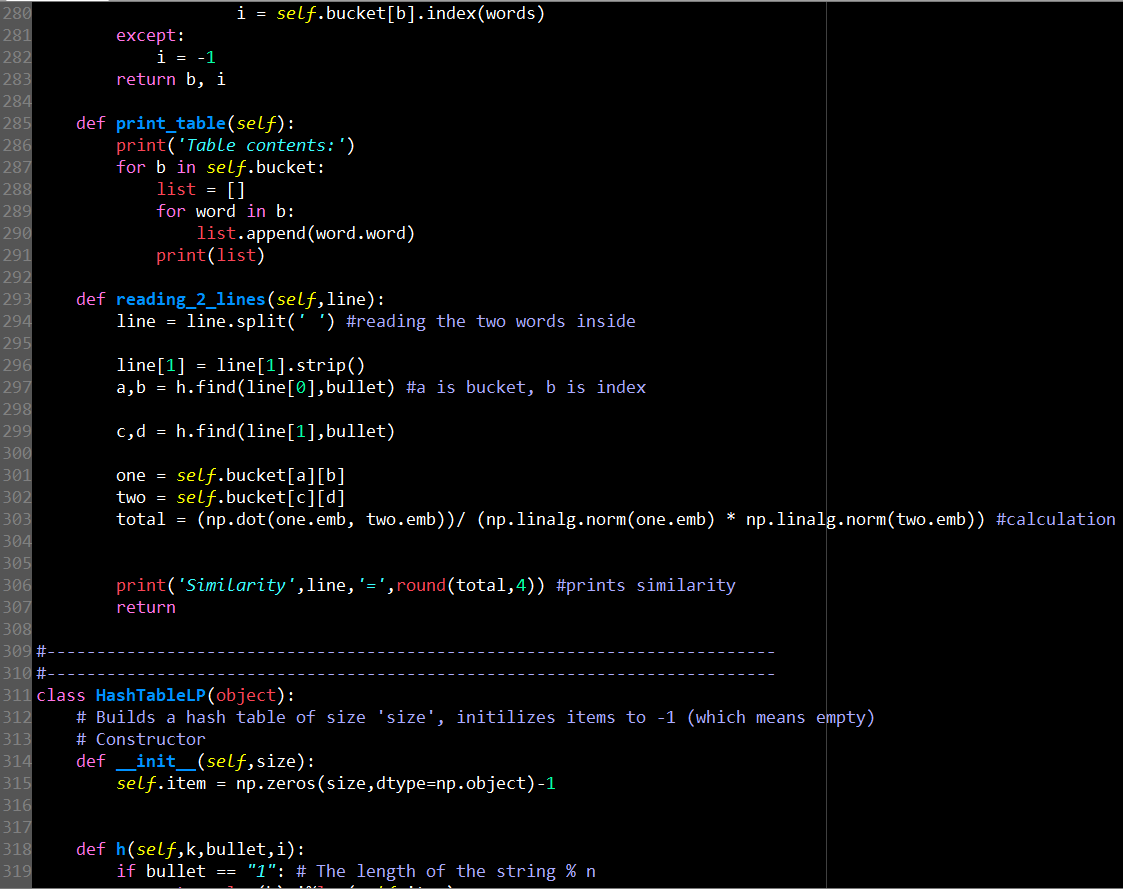
****

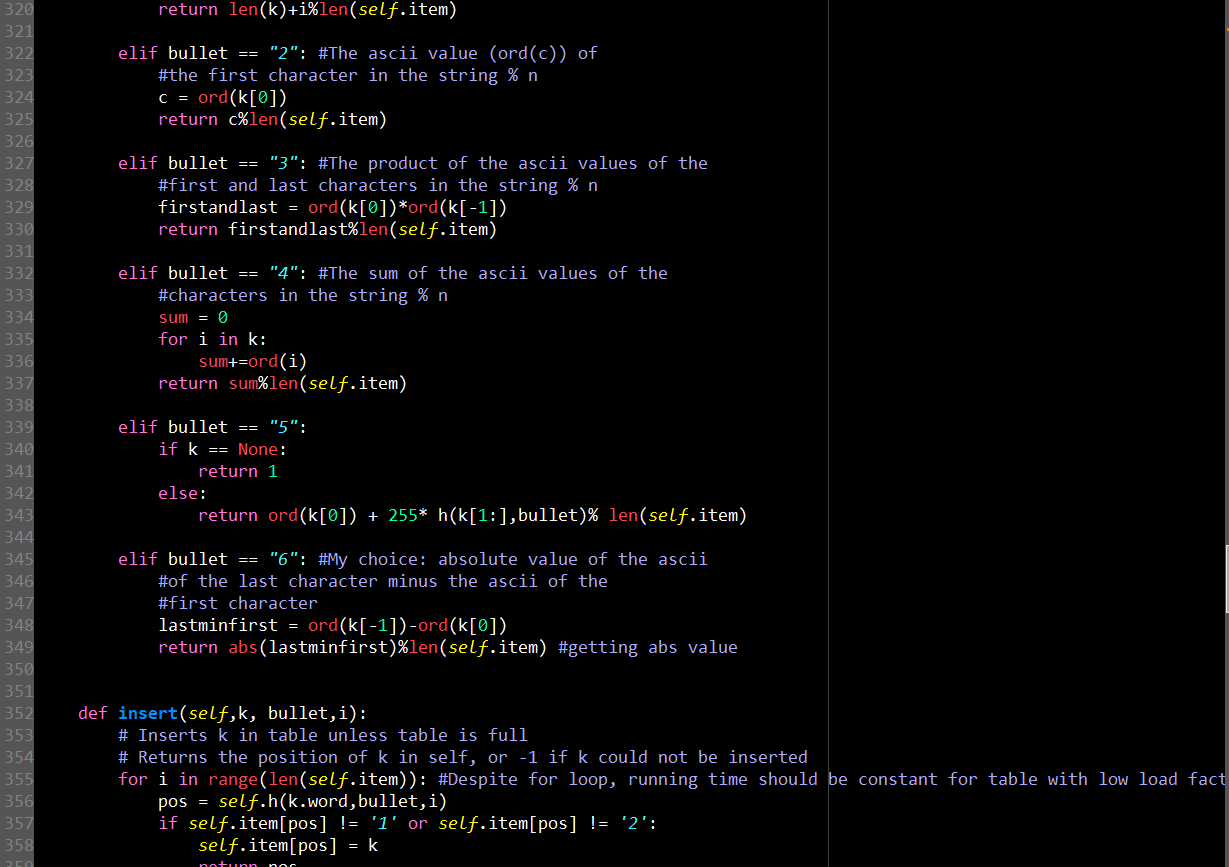
****

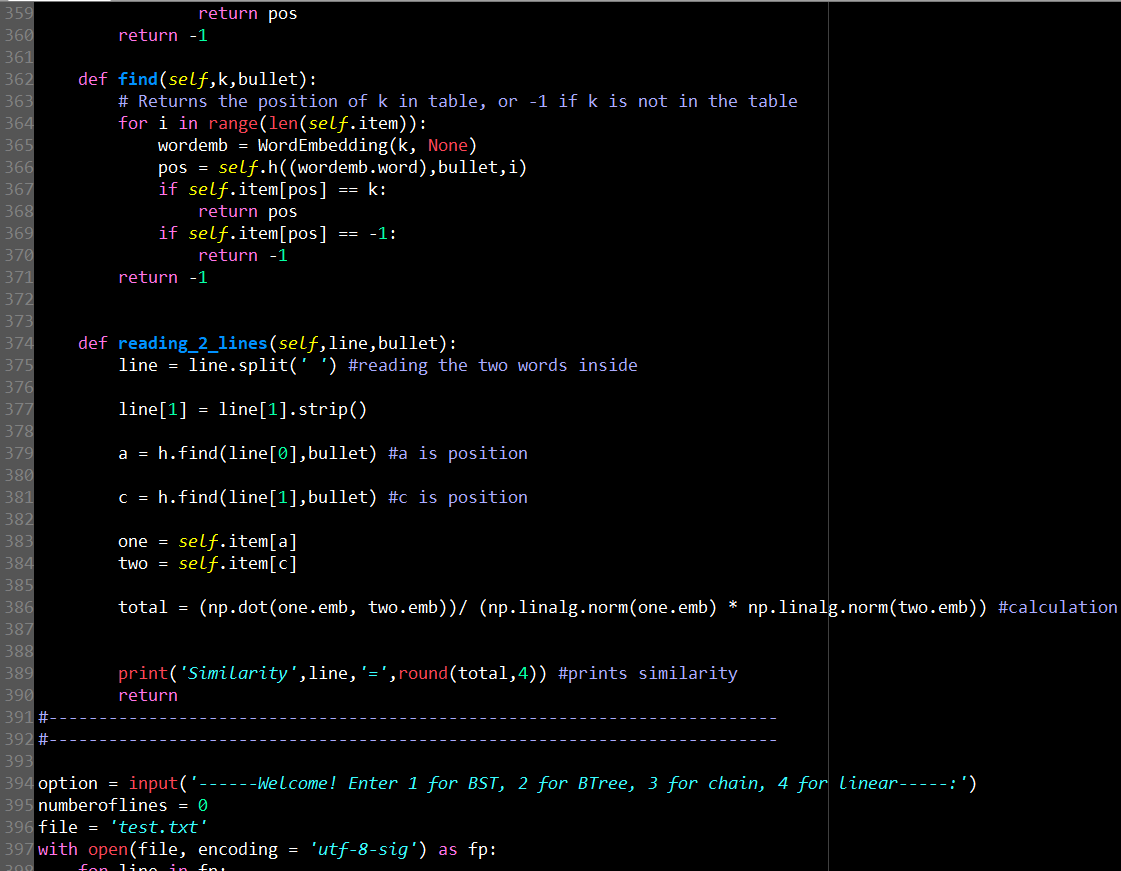
****

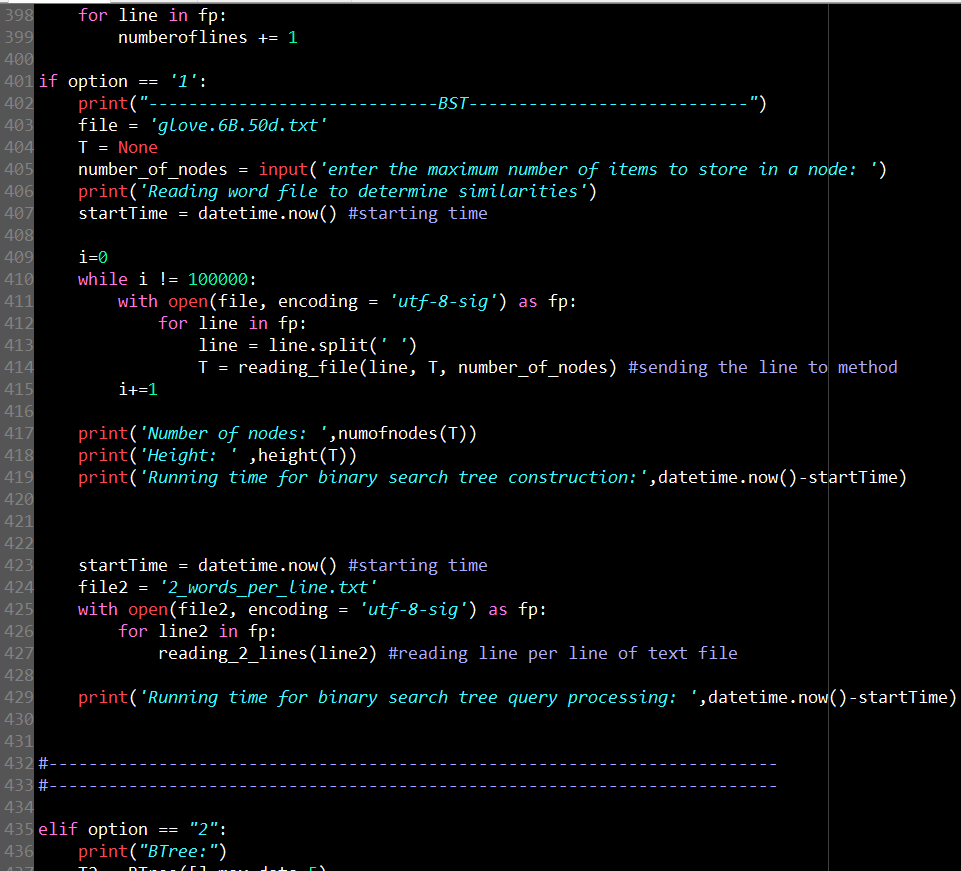
****

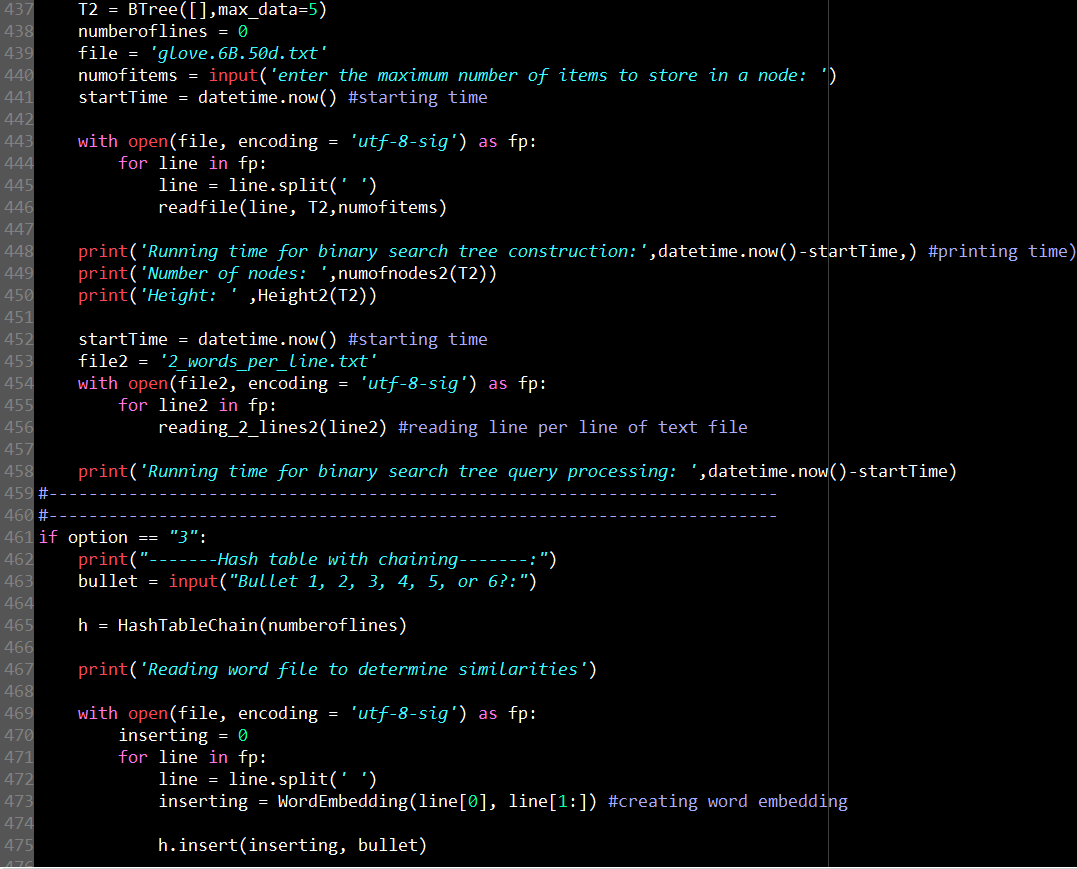
****

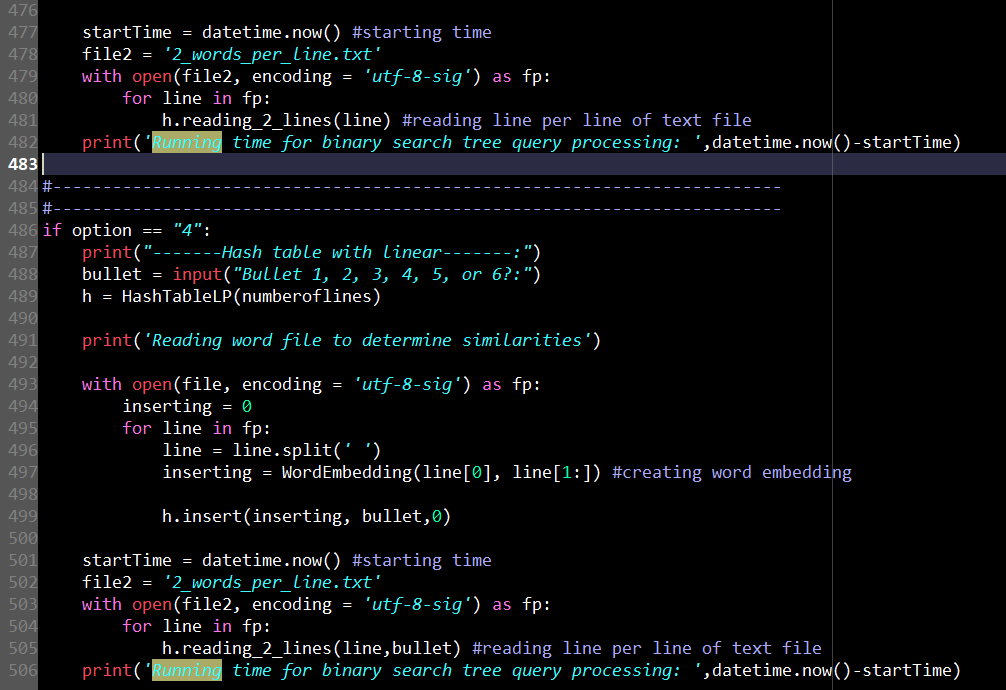
****

****

****

****

****

****

I certify that this project is entirely my own. 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.