Lab 4 – BSTrees and BTrees

Jonatan M. Contreras

University of Texas at El Paso

**Introduction**

In lab four, Binary Search and B Trees were implemented to store close to 370,000 words and, using these data structures, calculate the similarities of the words given by their embeddings. The program asks the user to choose their data structure implementation (out of the BSTree and Btree options; i) of choice in order to store the words and retrieve the similarities of the words desired. If Btrees were chosen, a max\_data for the data list is also requested.

**Proposed Solution Design and Implementation**

*Part 1: BSTree Implementation:*

To implement the BSTree solution, I began by modifying the starter code of the BSTree so it can work with WordEmbedding objects. This meant modifying the Insert() and Search() functions for the BST so that the WordEmbedding objects’ words were being compared in the respective functions. Then, the program simply reads the glove.6B.50d.txt file, line by line. If the first character of the line that was read is not alphabetical, it moves onto the next line. Else, it creates a WordEmbedding object by passing the first item in the list created by reading the line as the word, and the rest of the list as the embedding attribute. Once this is done, it uses the Insert() function to insert the WordEmbedding object into a BST. The program does this until every line has been read, and thus, every word in the .txt file has been stored as a WordEmbedding object in the BST.

Once the words are stored in the BSTree, a .txt file containing pairs of words is read line by line. As the pairs of words are read, it creates a list out of the words on the line. The function then searches for the words in the BSTree and stores the node that they are in. Once the nodes are retrieved, the method calculates the similarities of the two words. It stores these similarities in a list and continues to the next line. Once it is done storing every similarity, it returns the list of similarities.

*Part 2: Btree Implementation:*

To implement the BTree solution, I also modified the starter code of the BTree so it can work with WordEmbedding objects. This meant modifying the FindChild(), InsertLeaf(), Insert(), and Search(), functions for the BTree so that the WordEmbedding objects’ words were being compared in the respective functions. For the InsertLeaf() function, it was modified so that the list was sorted based on the words of the WordEmbeddings object. The program then reads the glove.6B.50d.txt file, line by line. Like previously, If the first character of the line that was read is not alphabetical, it moves onto the next line. Else, it creates a WordEmbedding object by passing the first item in the list created by reading the line as the word, and the rest of the list as the embedding attribute. The program then uses the Btree’s Insert() function to to store it in the Btree. The program does this until every line has been read, and thus, every word in the .txt file has been stored as a WordEmbedding object in the Btree.

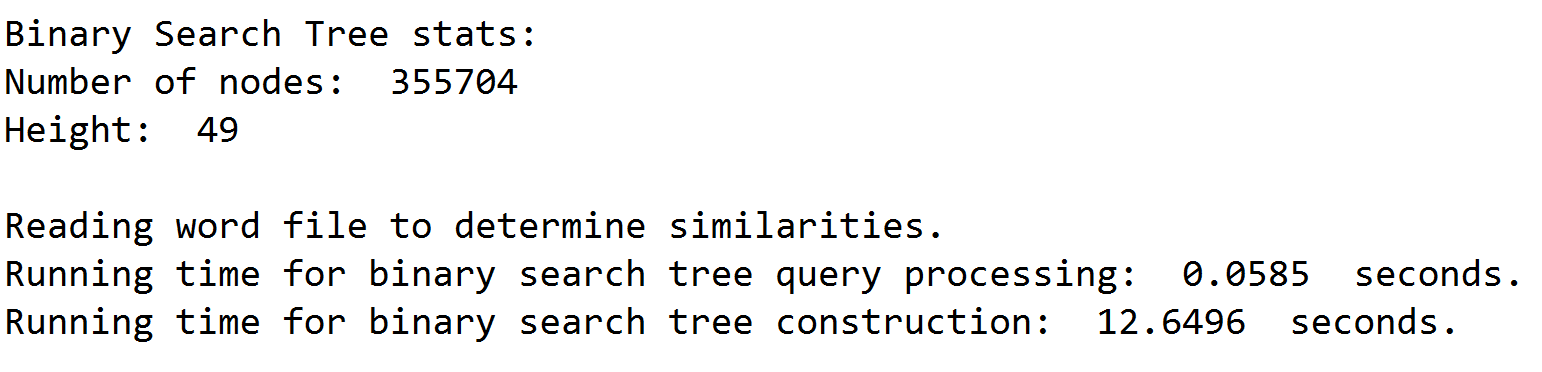
Once the words are stored in the BTree, a .txt file containing pairs of words is read line by line. As the pairs of words are read, it creates a list out of the words on the line. The function then searches for the words in the BTree and stores the node that they are in. Once the nodes are retrieved, the method calculates the similarities of the two words. It stores these similarities in a list and continues to the next line. Once it is done storing every similarity, it returns the list of similarities.

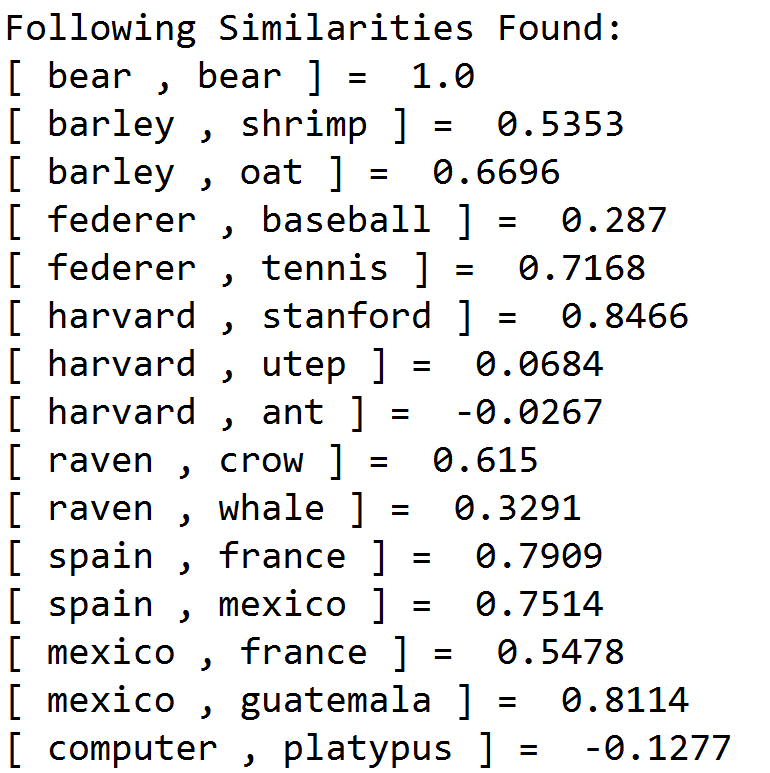
**Experimental Results**

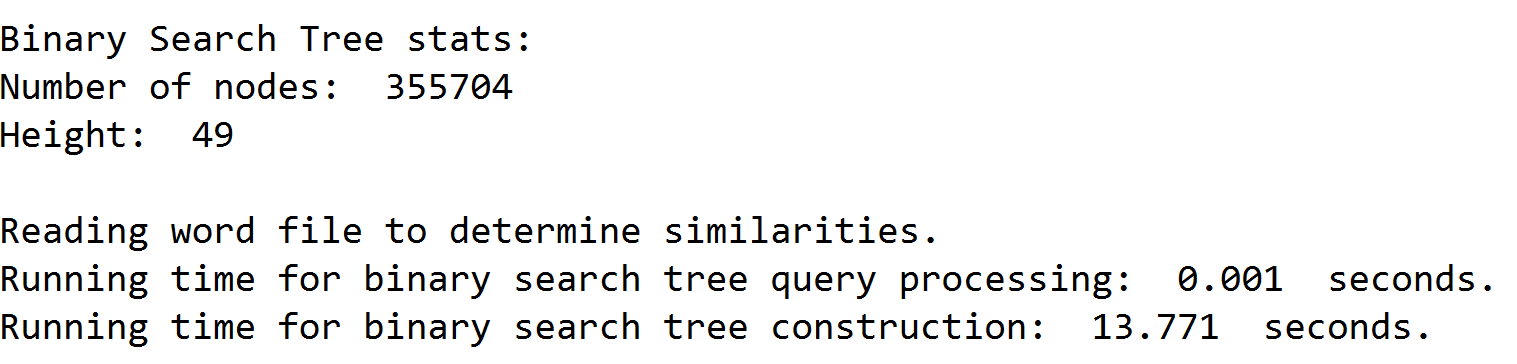
*Part 1: BSTree Implementation:*

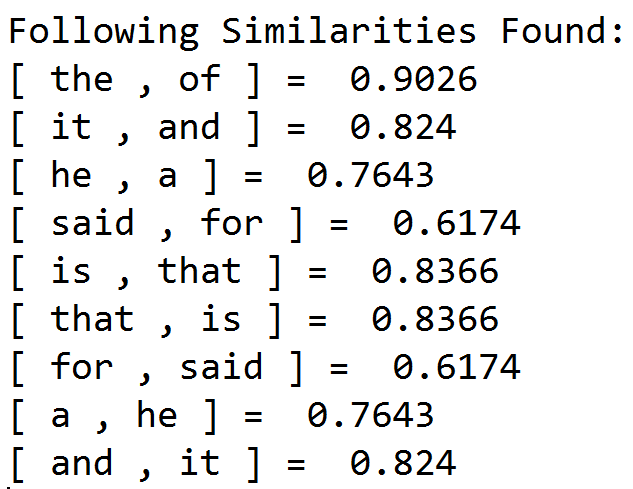
The following are screenshots of the output at varying sizes of compared words:

**Similarities for Example in Lab Handout:**

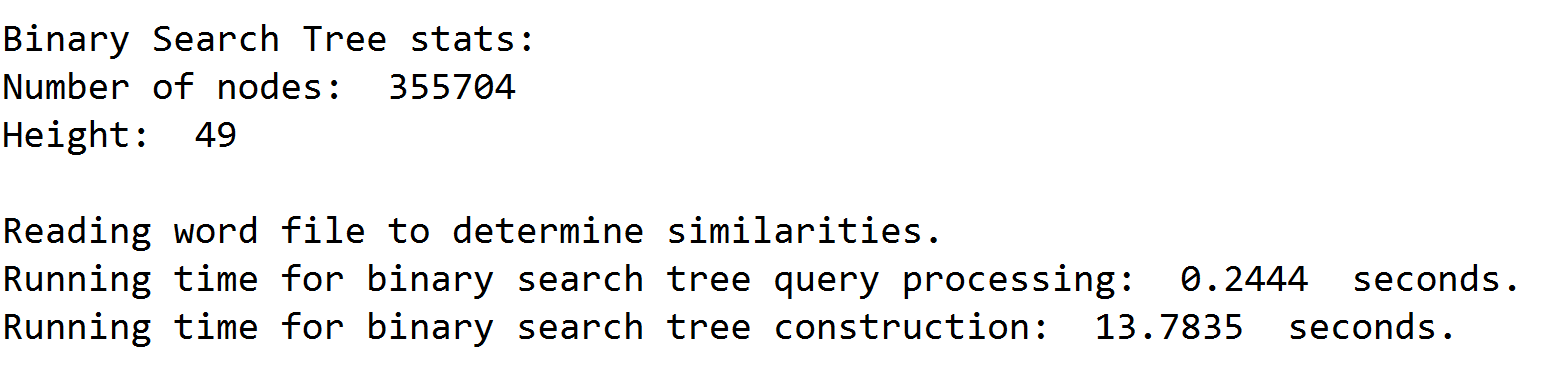
****

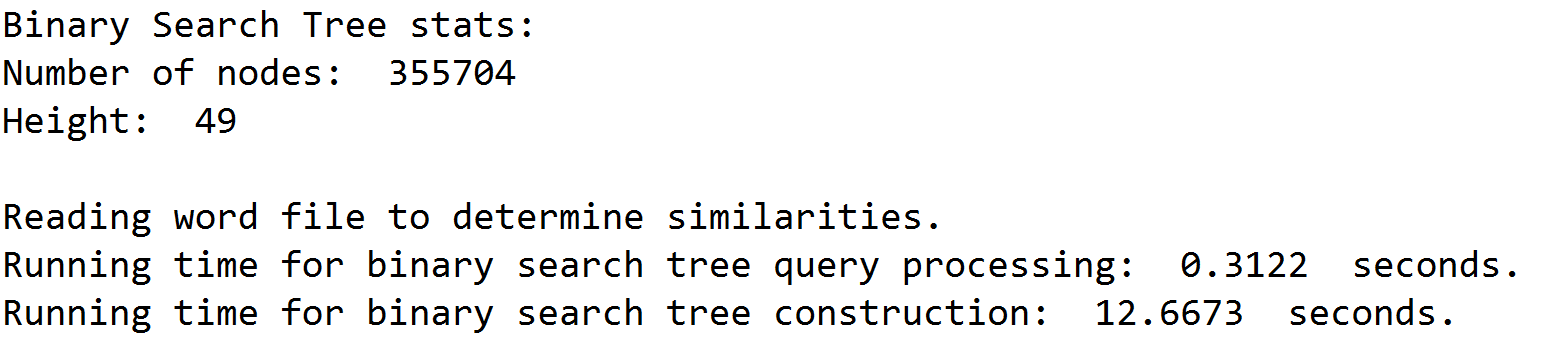
****

**List Size = 10 [Similarities only printed for this list size]:**  


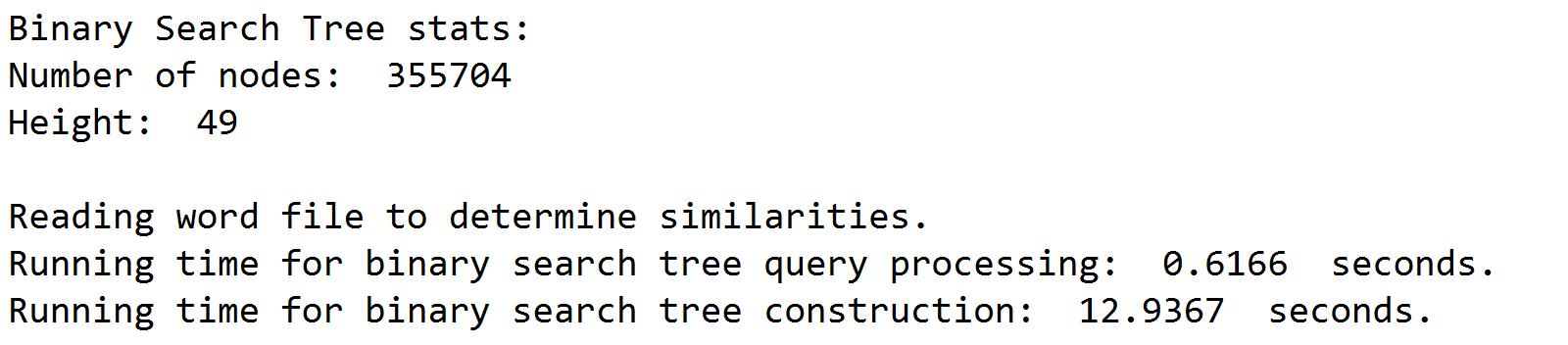


**List Size = 100:**

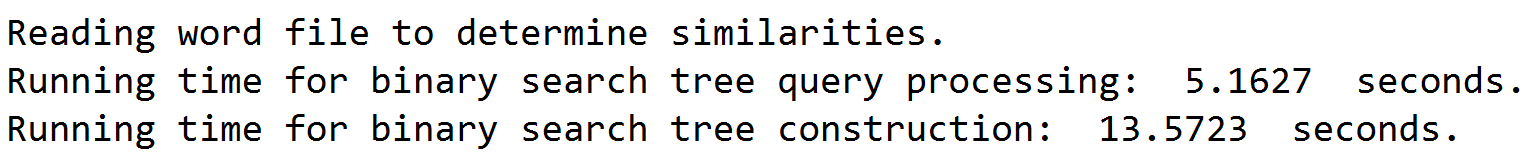


**List Size = 1,000:**

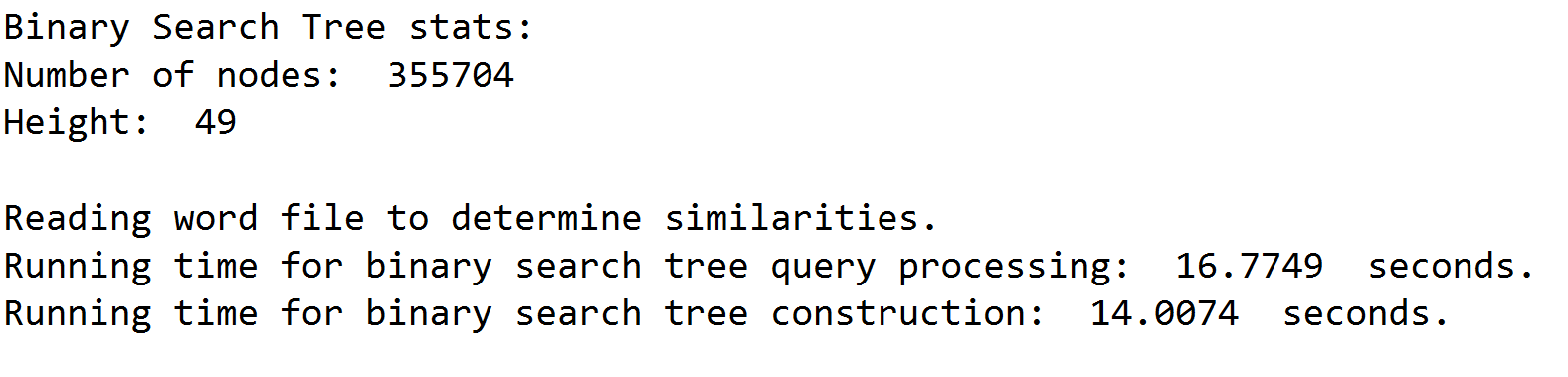
**List Size = 10,000:**



**List Size = 100,000:**



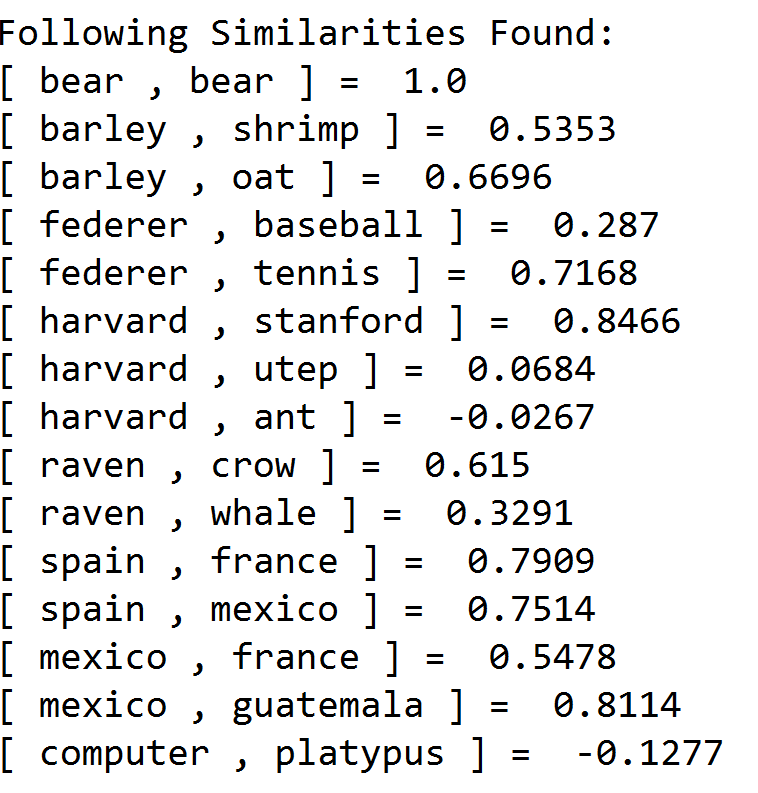
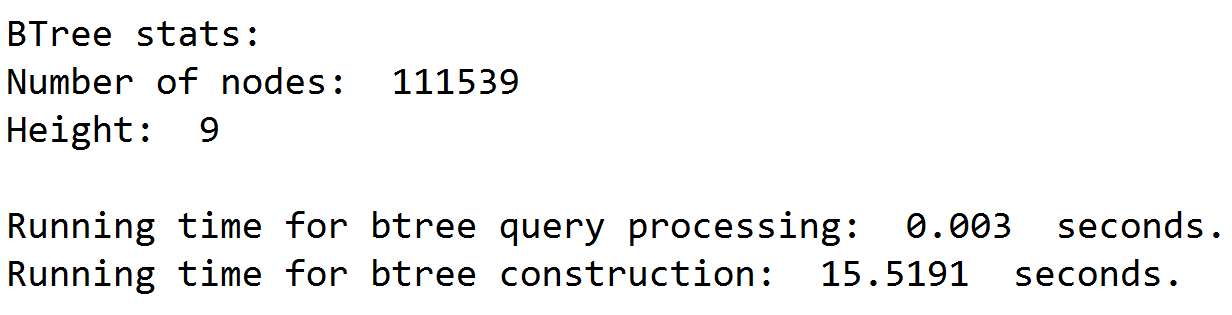
**List Size = (Close to 355,000):**

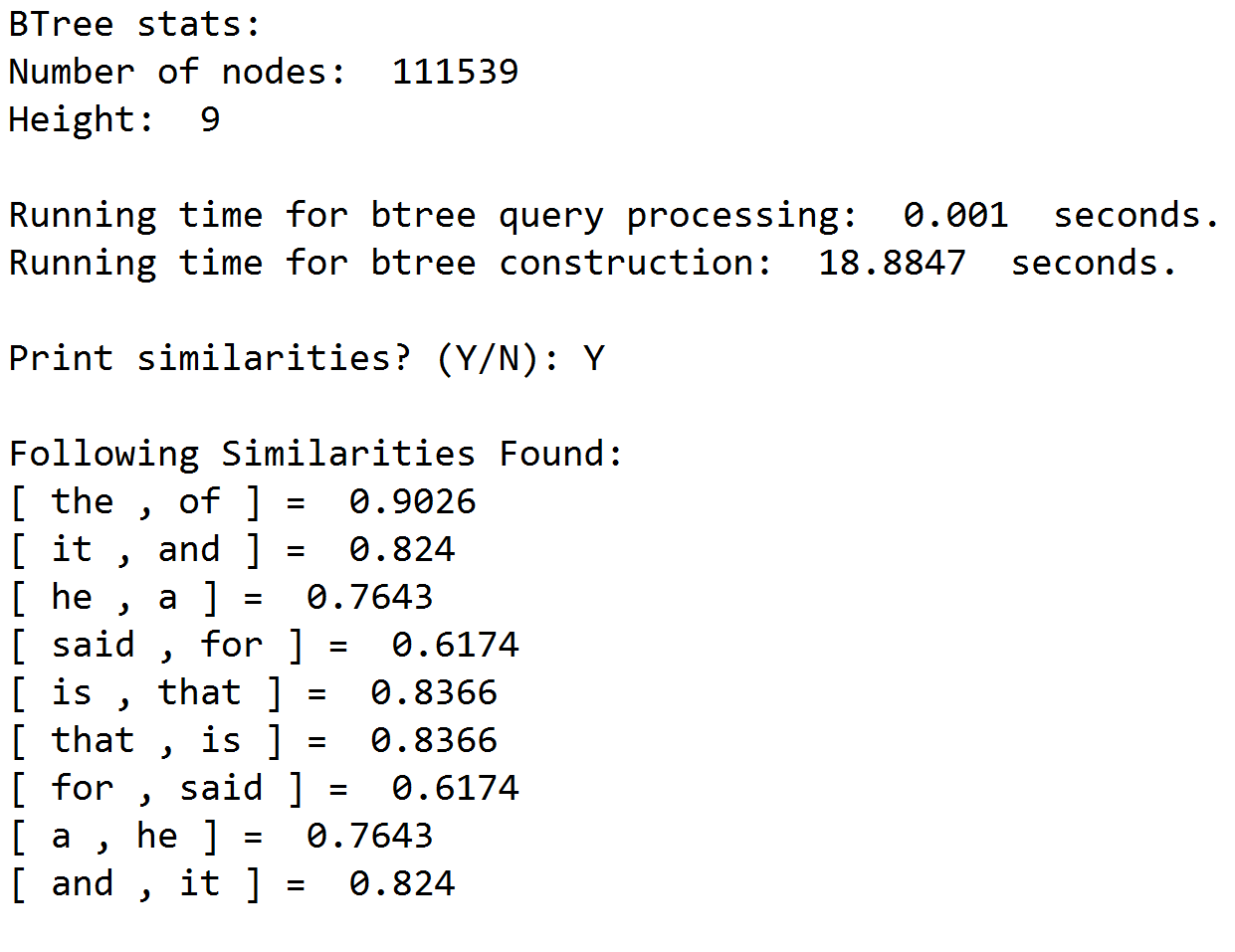


*Part 2: BTree Implementation:*

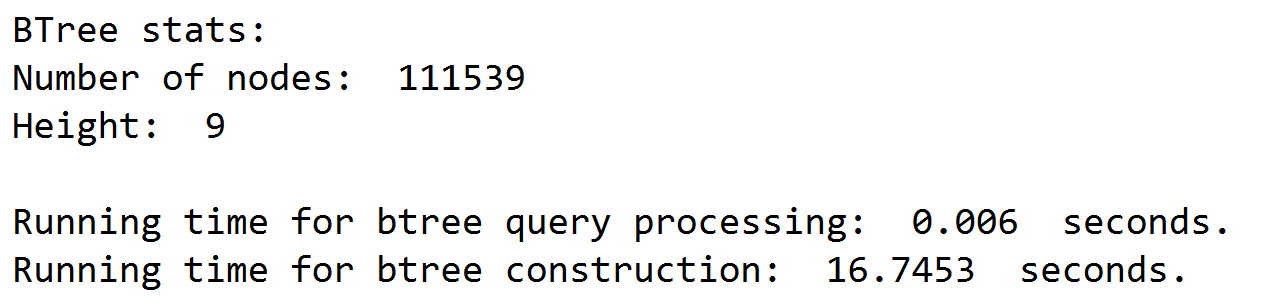
The following are screenshots of the output at varying sizes of compared words:

**Similarities for Example in Lab Handout:**

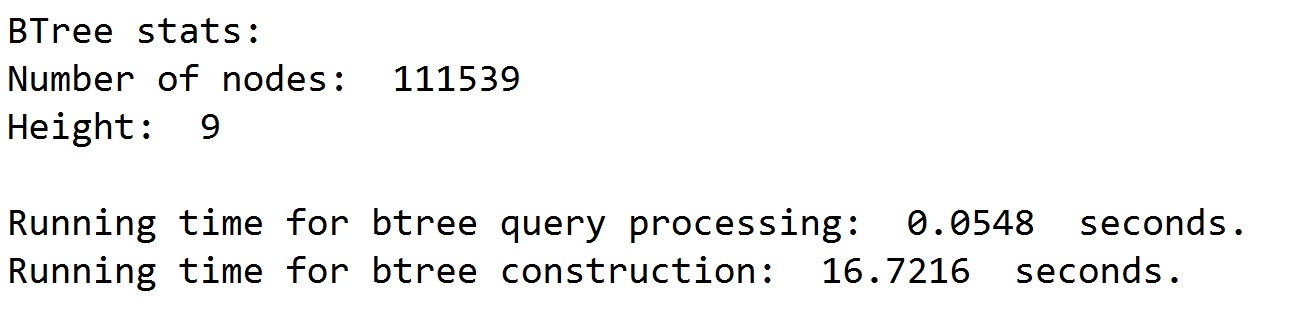
****

**List Size = 10 [Similarities only printed for this list size]:**  


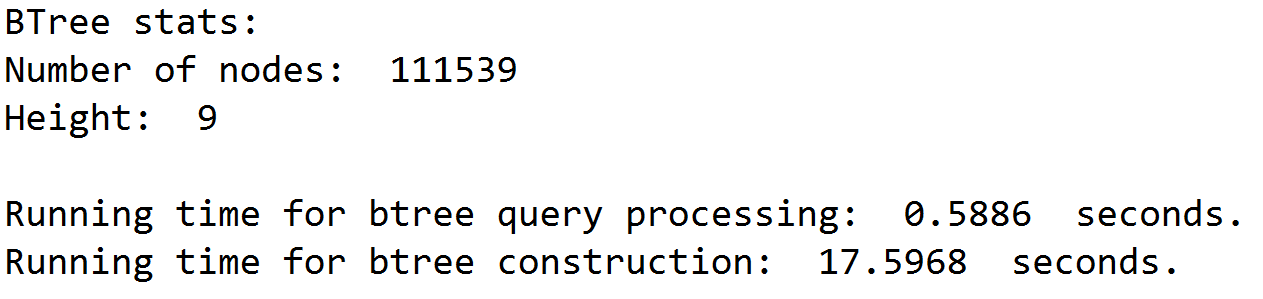
**List Size = 100:**



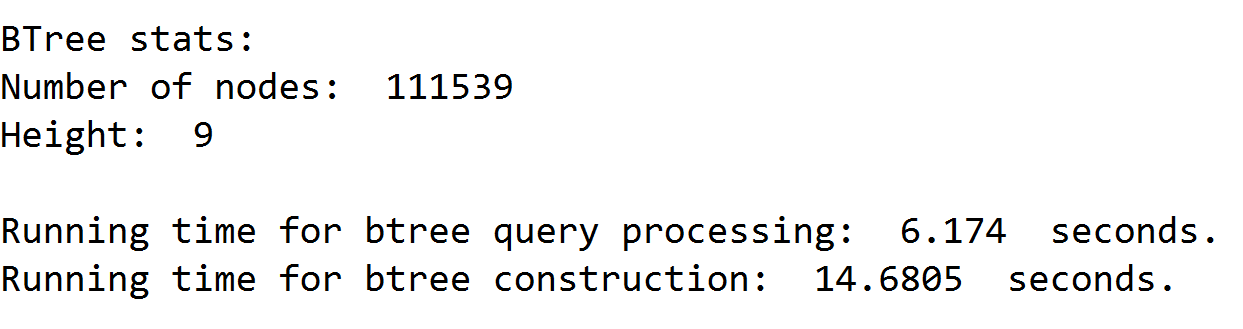
**List Size = 1,000:**

****

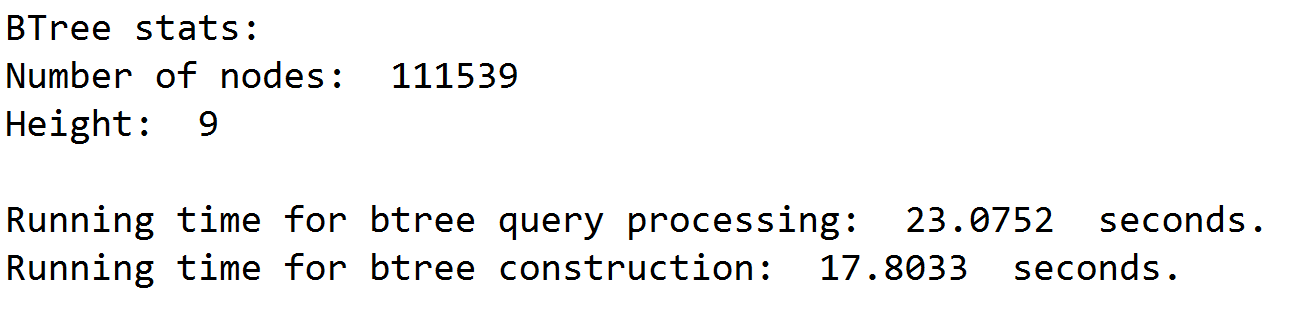
**List Size = 10,000:**

****

**List Size = 100,000:**

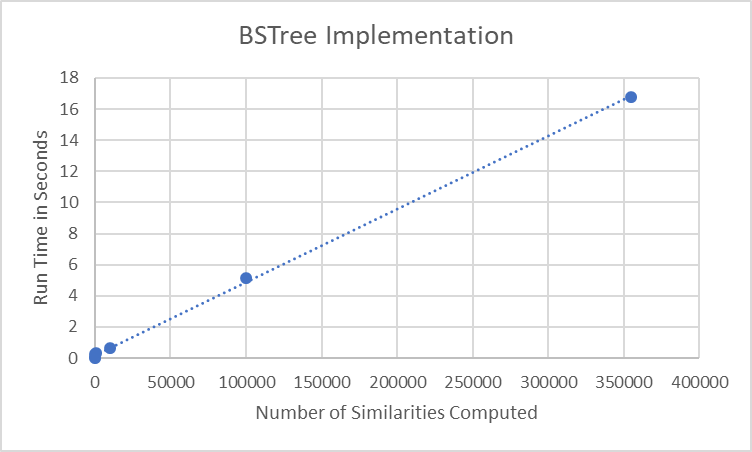


**List Size = (Close to 355,000):**

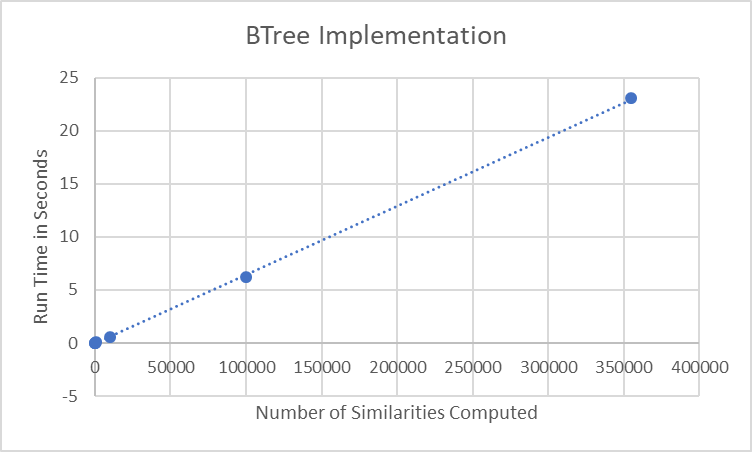


*Part 3: Implementation Graphs:*

*BSTree Implementation*



*Btree Implementation*

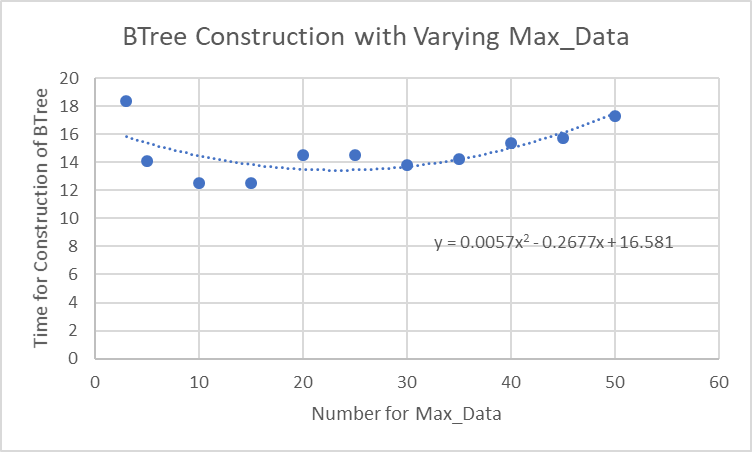


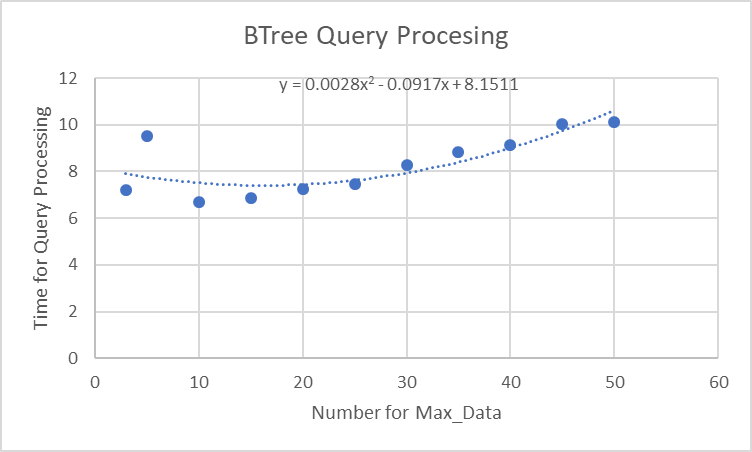
**Efficiency of Methods**

Both my BSTree and Btree methods seem to be performing comparably when it comes to building the trees. This is surprising because I believed the Btree should perform better than the BSTree. It seems as both my methods are running at O(N) for the query processing. This seems approppriate as it is performing two O(logN) algorithms and (logN)2 becomes N.

**Effect of Different Max\_Data for Btree:**

As pictured below, it seems like the behavior for number of max\_data allowed in the Btree with the construction time as well as query processing time is an inverse parabola. As number for Max\_data increases, it reduces time needed for construction and query processing for B Tree. Then, as it continues to increase, the time needed for construction and query processing increases again.





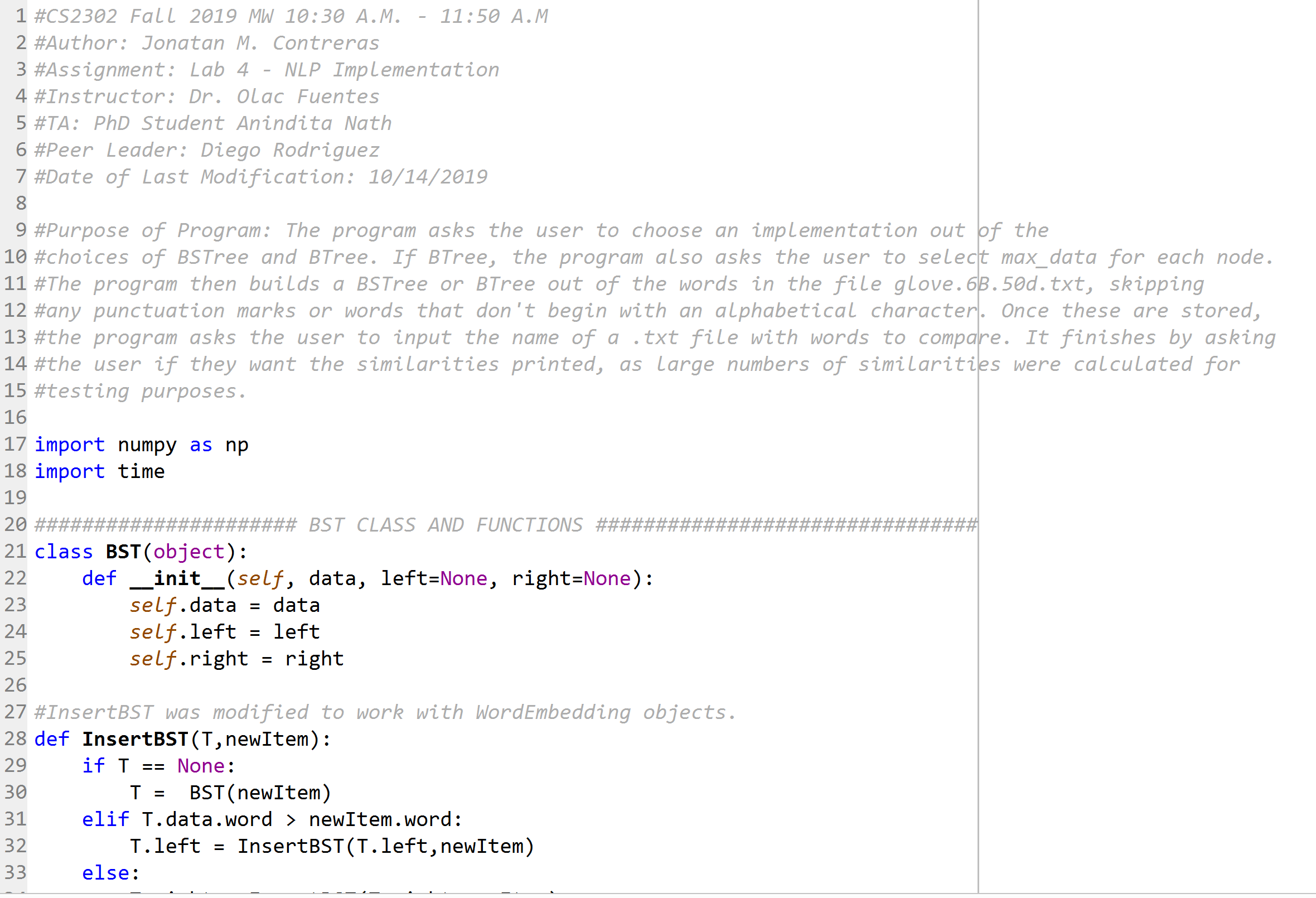
**On Word Embeddings**

This is an interesting way of representing words and there were interesting results calculated. For example, the similarities between Harvard and Stanford versus Harvard and UTEP were interesting. A Google Scholar search of word embeddings yields very interesting looking scholarly articles that I would like to learn more about.

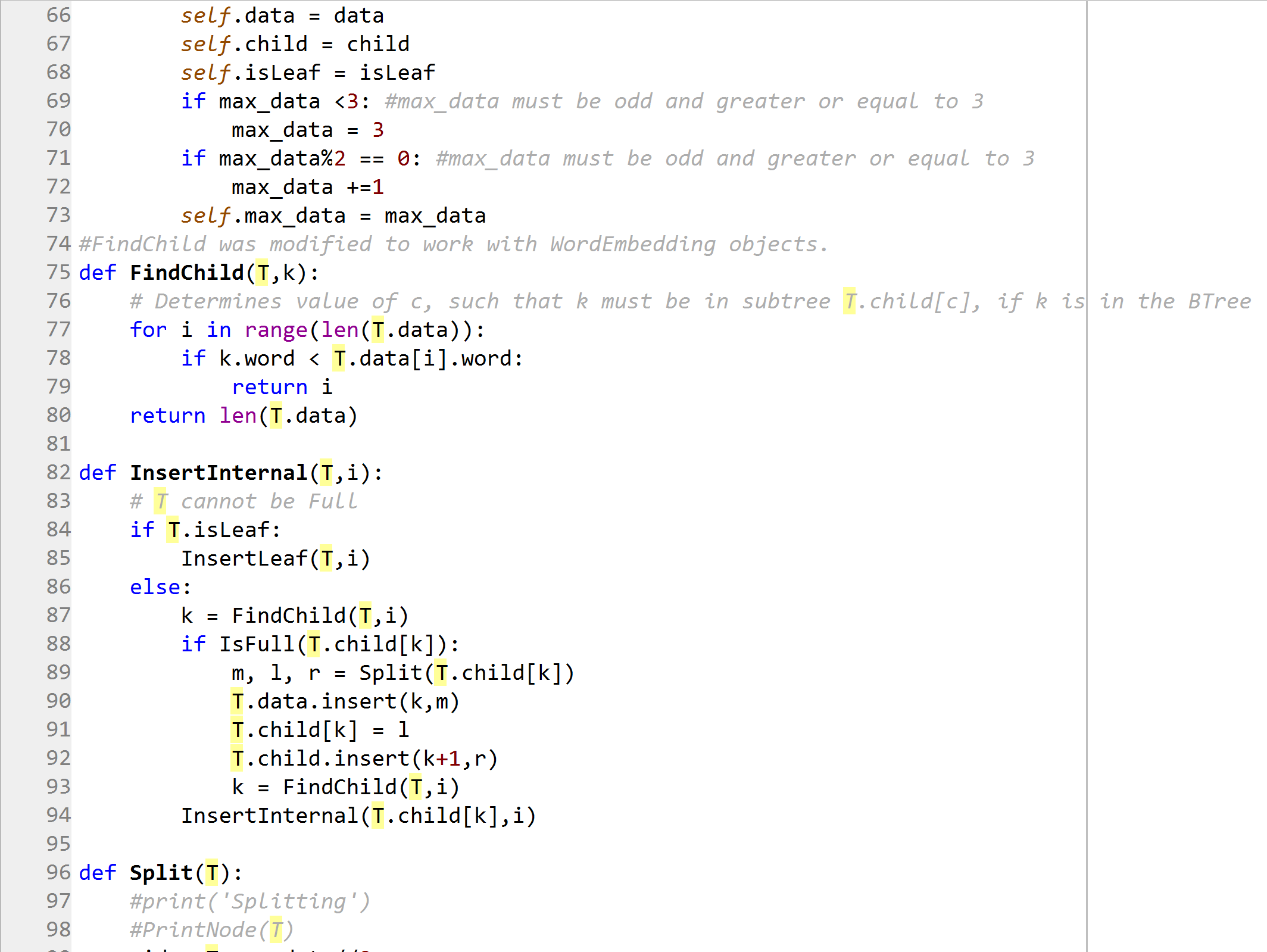
**Conclusion**

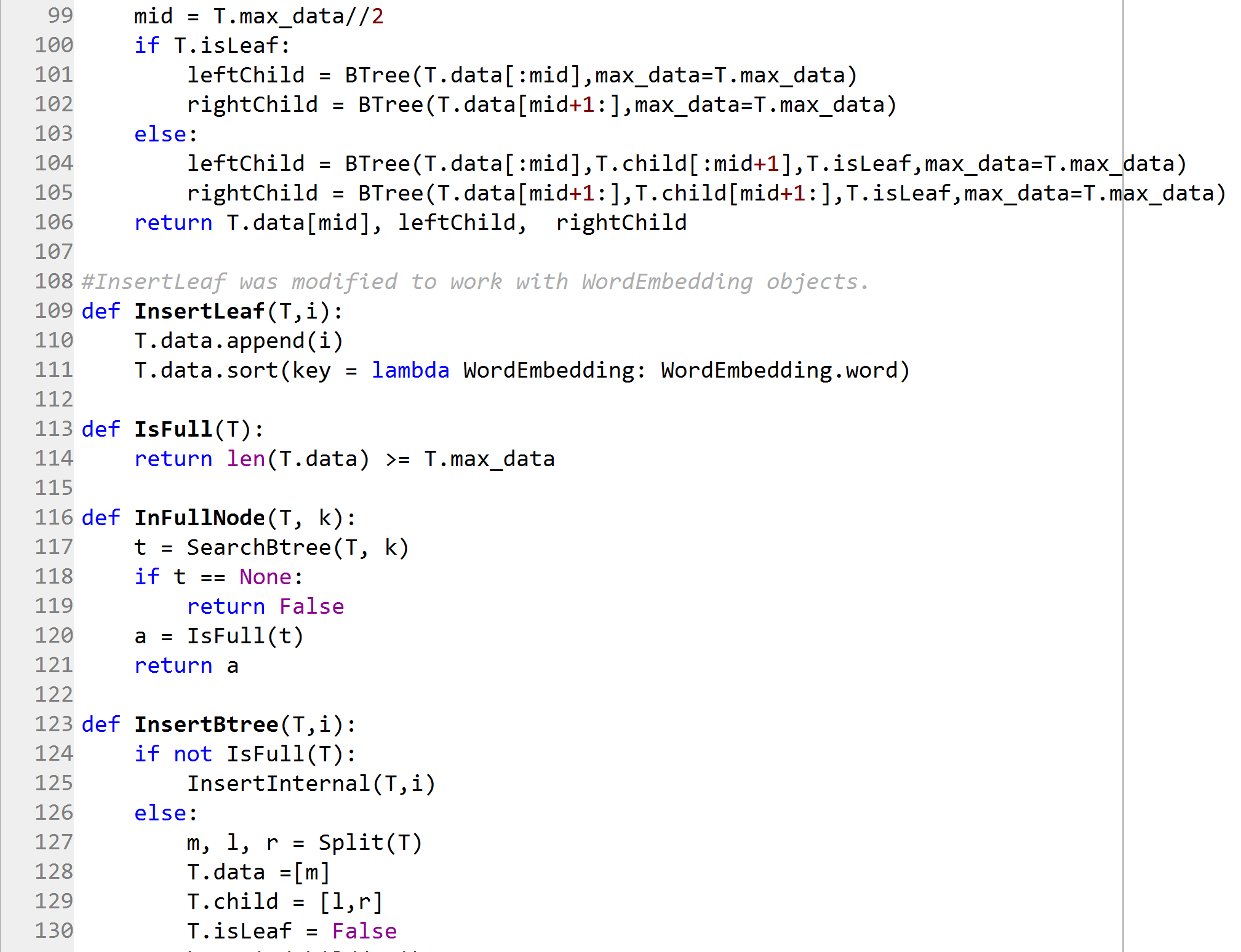
I learned how to implement BSTrees and Btrees to be used with different data types. I also learned a very small amount about natural language processing which is a field I am highly interested in. Finally, I also learned about the effects having different max\_data values has on the efficiency of the Btree.

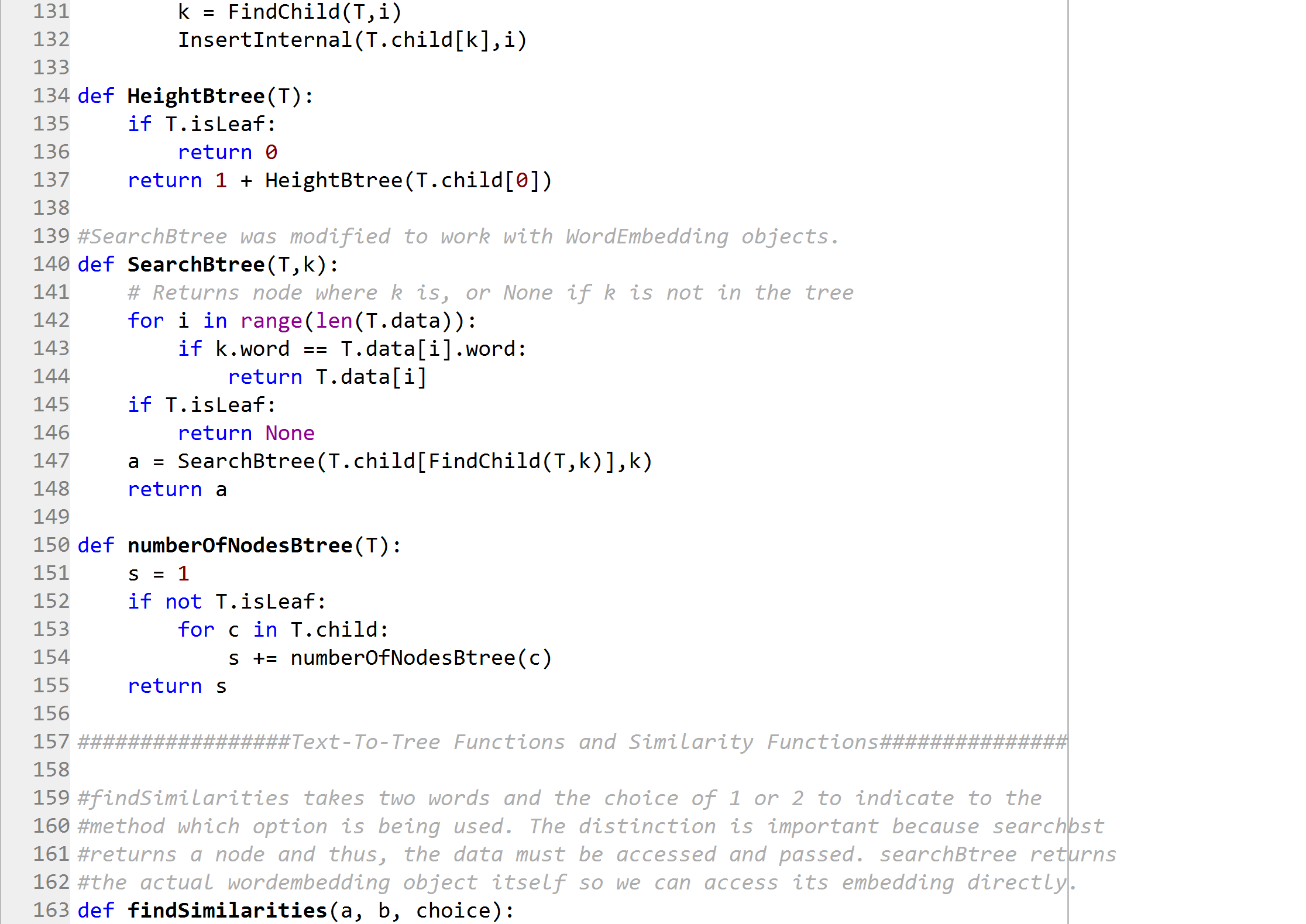
**Appendix – Source Code**

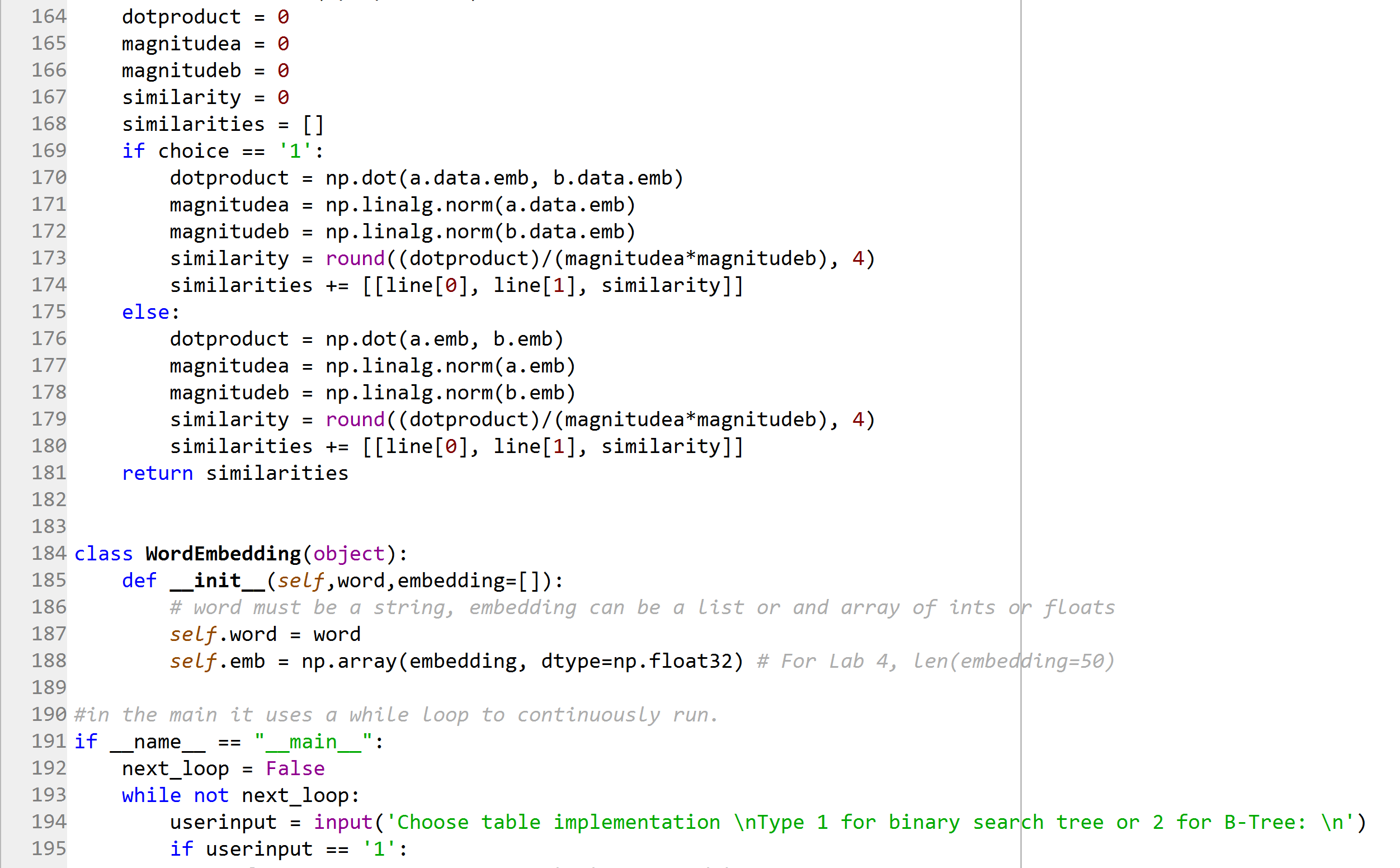
****

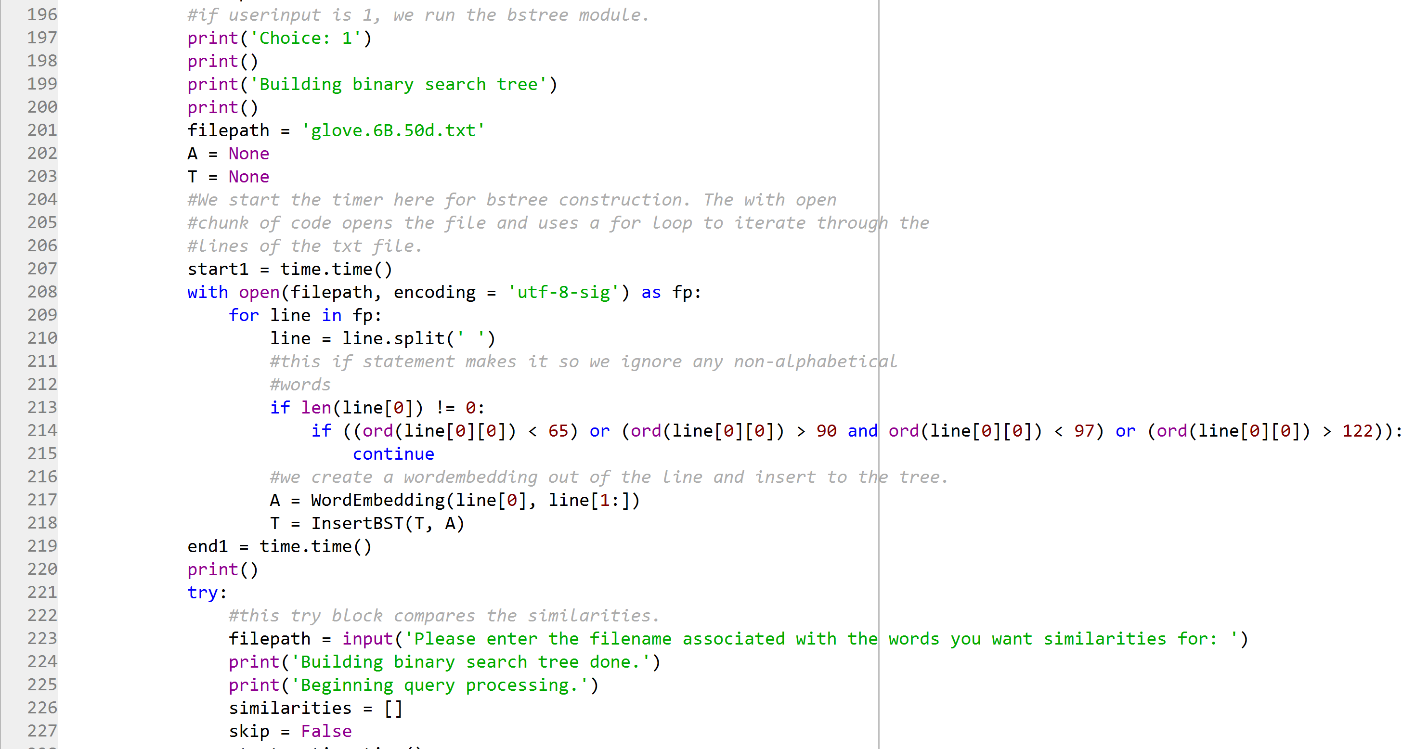
****

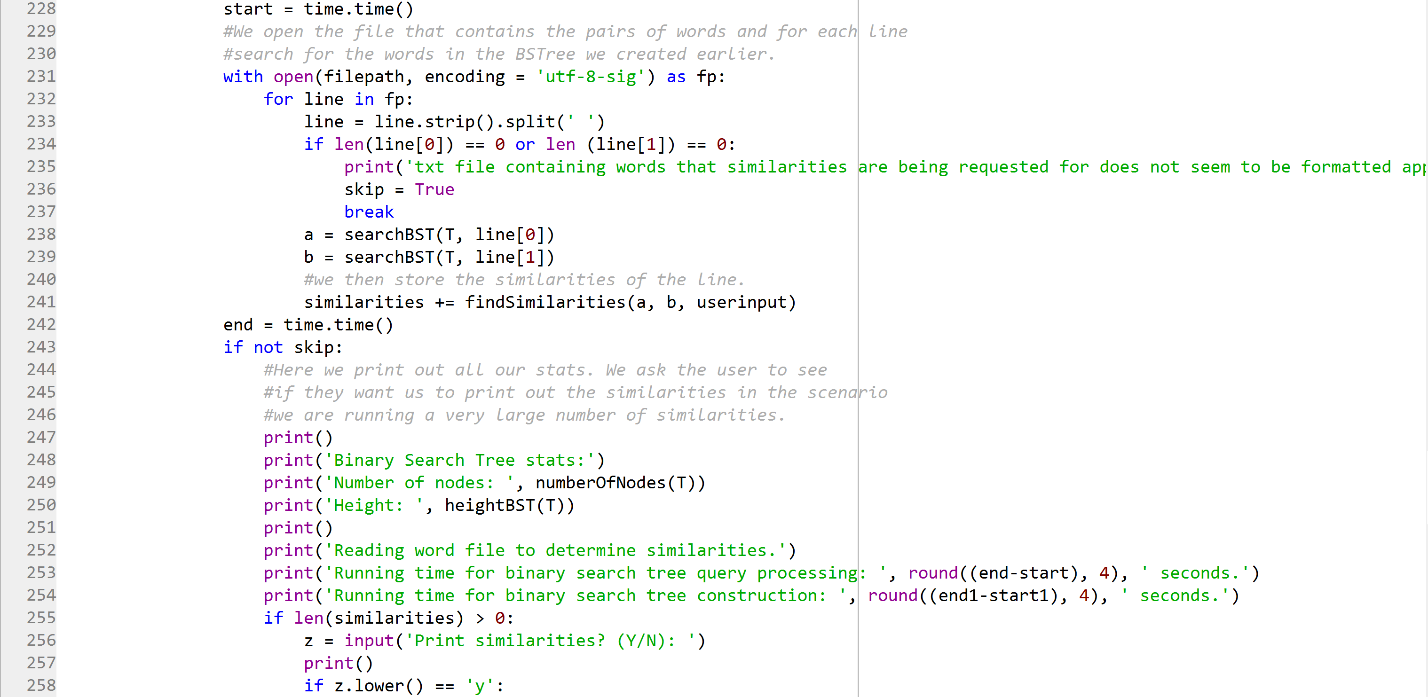
****

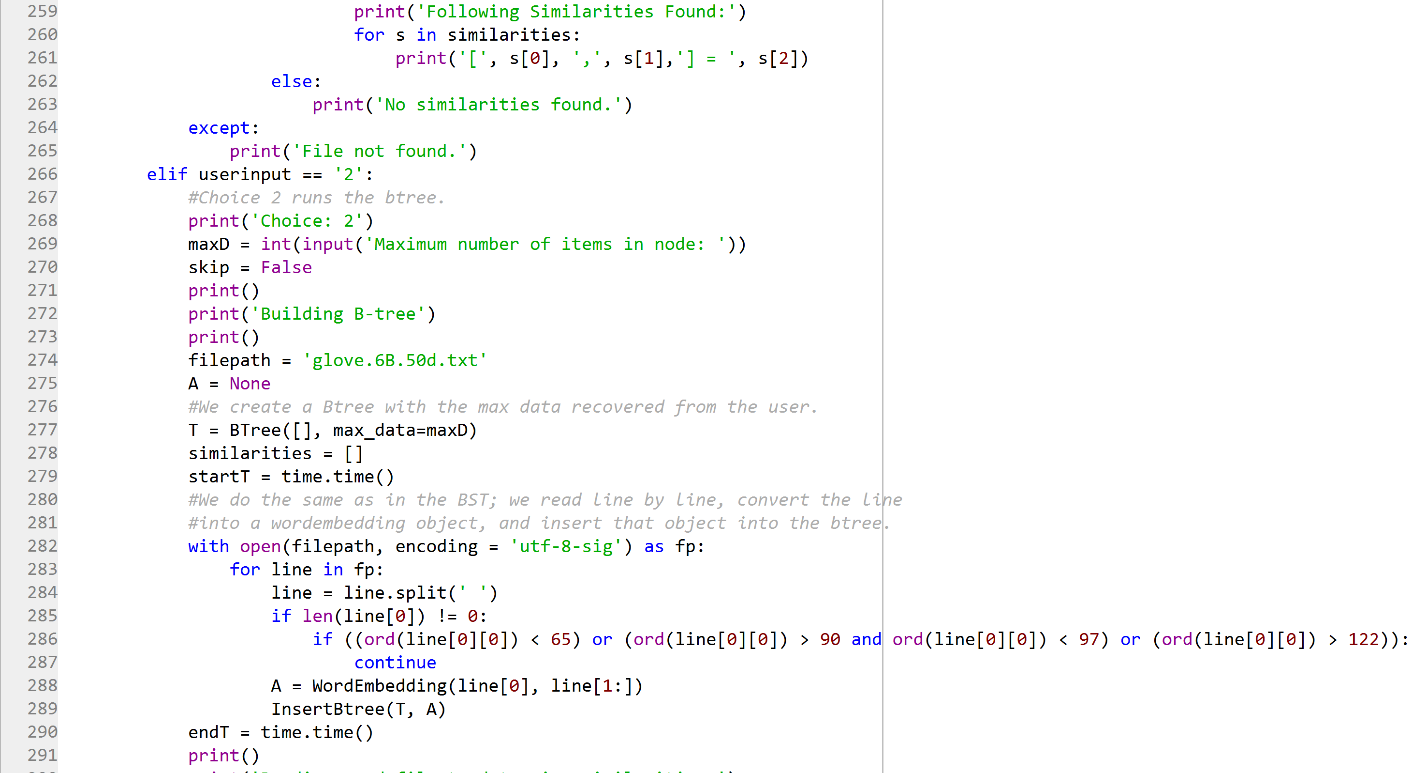
****

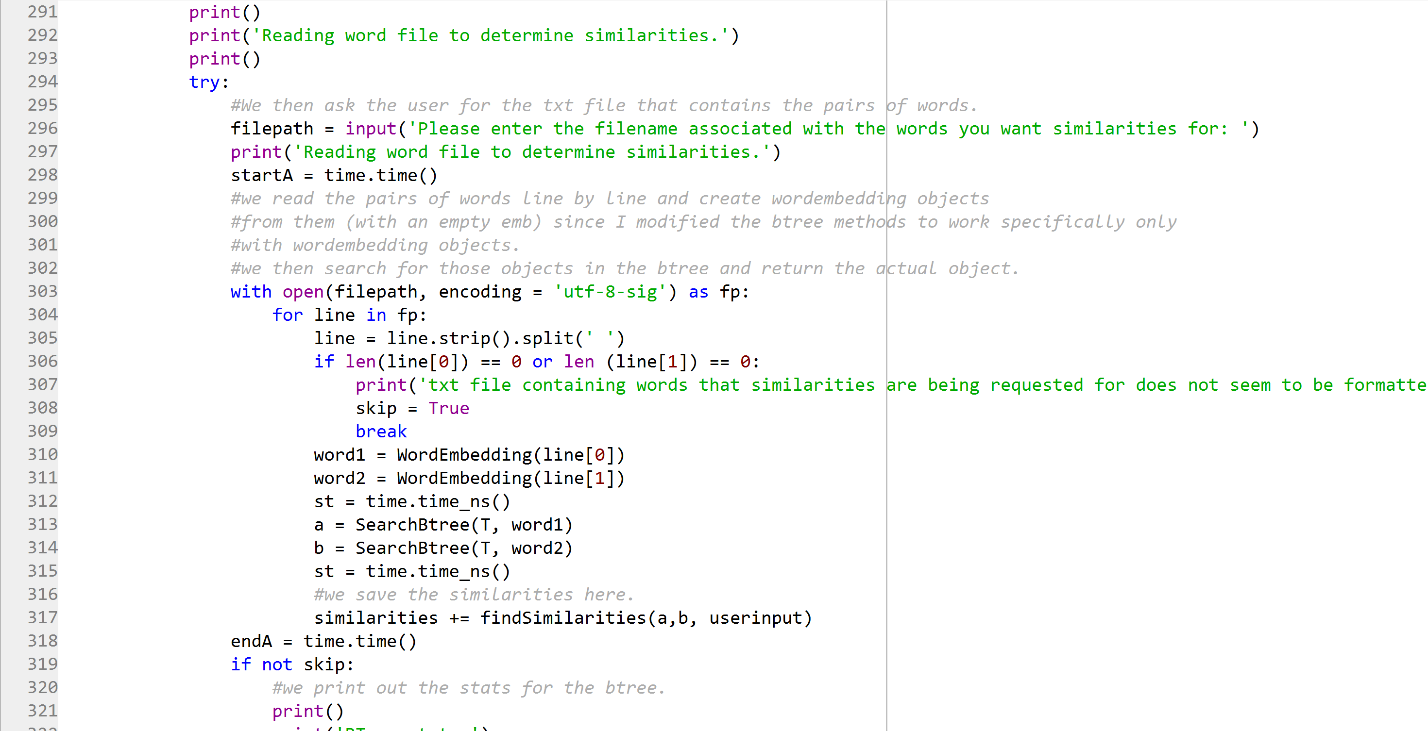
****

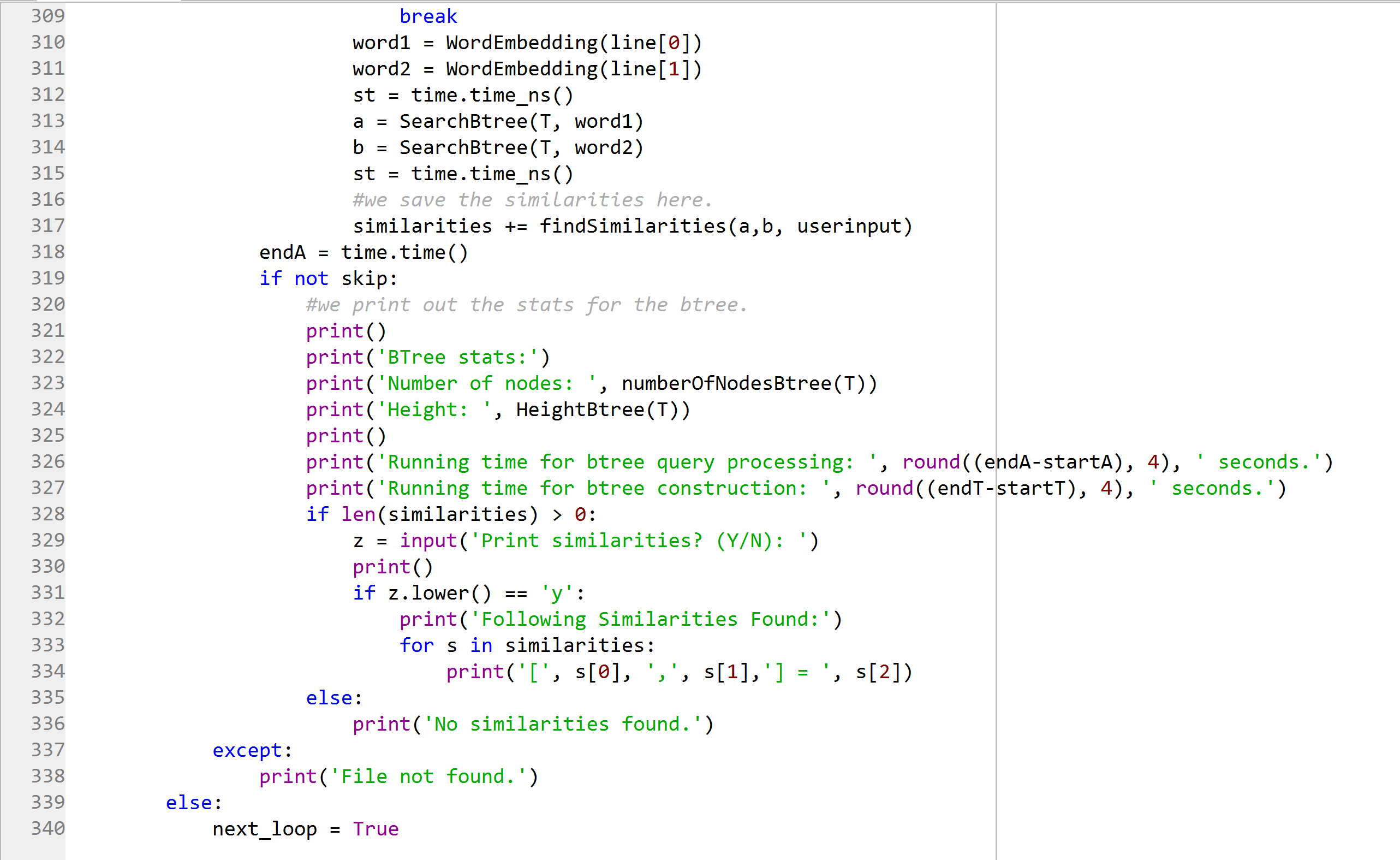
****

****

****

****

****

****

**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.

Jonatan Contreras