CS2302 Data Structures

Fall 2019

Lab Report #5

Due: 1 November 2019

Professor: Olac Fuentes

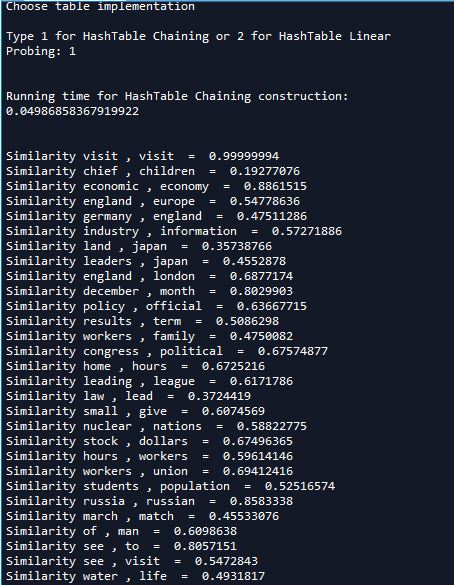
TA: Amindita Nath

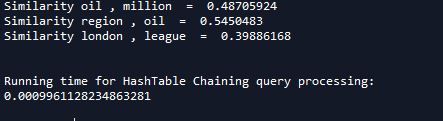
**Introduction**

The problem presented in this lab is relating to the similarity between words when AI process words, so given a large file of words with a list of floats that are embeddings for the word we have to implement a hash table with chaining and a hash table with linear probing. I implemented the hash tables and inserted nodes into each hash table through six different ways to hash where to input the nodes.

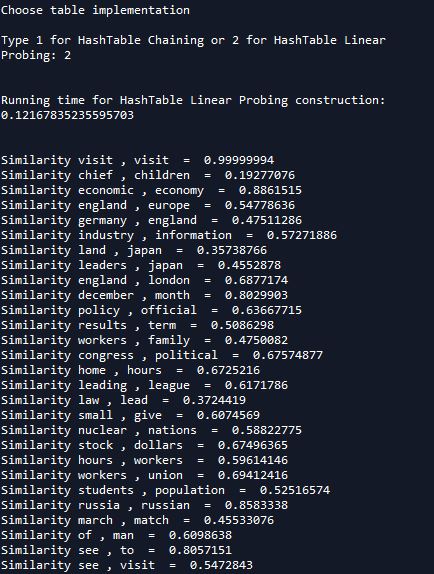
**Proposed Solution Design and Implementation**

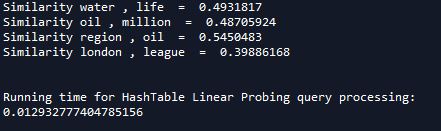
Design –

Hash Table Chaining Output



Hash Table Linear Probing Output





Operation #1: The first operation I completed was reading in the file which is where I read the whole file in and then split the file based on number because each word has 50 embedding floats, so I split them first into one attribute of the node and the next 50 into a different attribute of the node. Similarly, I read the pairs file and split the pair into different lists to use to test for similarity.

Operation#2: I implemented one hash table with chaining and one hash table with linear probing, I added separate class word embedding nodes into each of the hash tables depending on the hash function being used.

Operation#3: For the hash functions used to insert nodes most used the ascii value as the value to used to hash, the functions were the length of the string, ascii value of the first character, product of the ascii values of first and last characters in string, the sum of the ascii values of the characters in the string, a reclusive formulation, and the one hash I choose was the sum of the ascii value for the characters in the string plus the length of the string. Each function call is commented out in the insert and find functions for both hash tables

**Experimental Results**

The average running time for insert and search for the hash tables is O(1). Although since the set of words is so large there are a lot of nodes at certain indexes you have to iterate over the list in the bucket to find it and you have to iterate over the list in linear probing since there is a lot of collision.

Hash Table Chaining Query Time:

(This is the query time from the hash function that uses the product of the ascii values of first and last character, fastest query time)



Hash Table Linear Probing Query Time:

(This is the query time from the hash function that uses the product of the ascii values of first and last character, fastest query time)



Binary Search Tree Query Time:

(Fastest query time BST)



BTree Query Time:

(Fastest query time BTree, max\_items = 3)

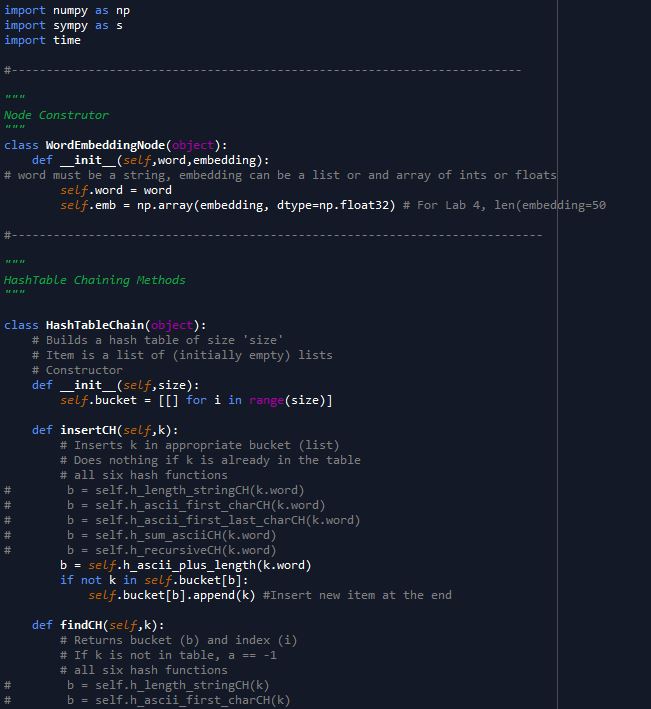


The fastest times shown above show that the Hash Table Chaining that uses the product of the ascii values of the first and last characters of the string with the Btree with the max amount of 3 being almost as fast as the Hash Table with chaining. The Hash Table with linear probing seems to be the slowest amongst the best results of these 4 data structures.

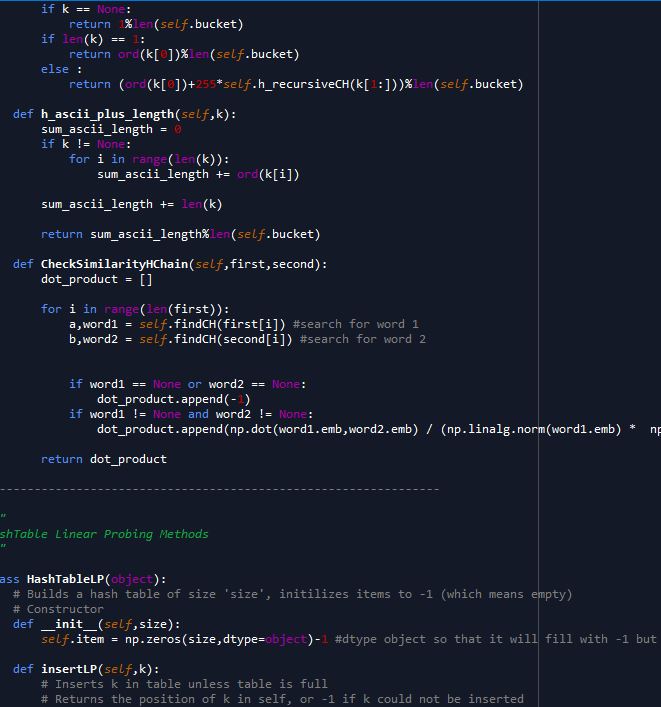
**Conclusion**

I learned that Hash Tables are very efficient when searching and inserting since you hash the index so that you know where each node is inserted. I have also learned that the HashTable chaining is faster than the other data structures that were tested for this lab.

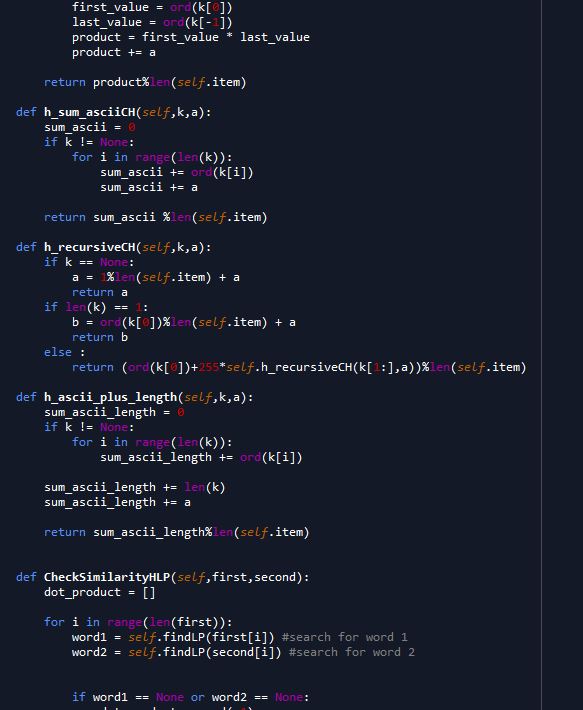
**Appendix**

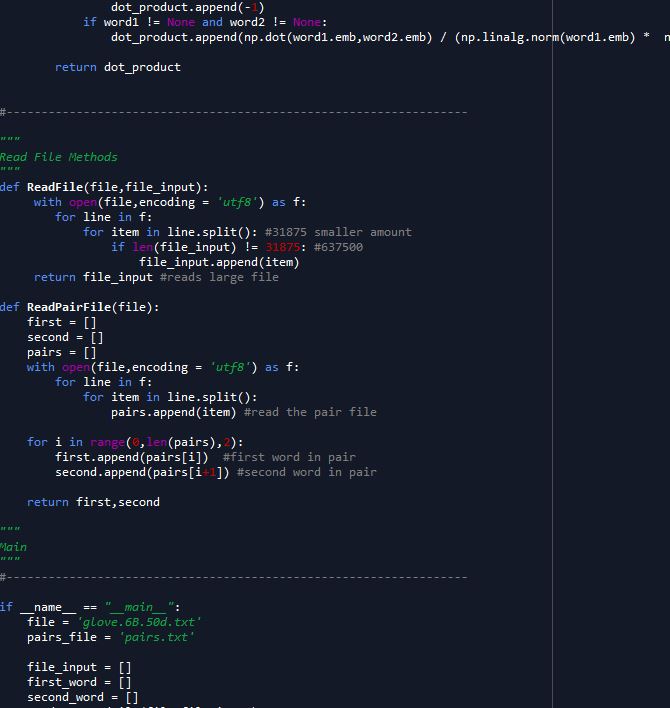
****

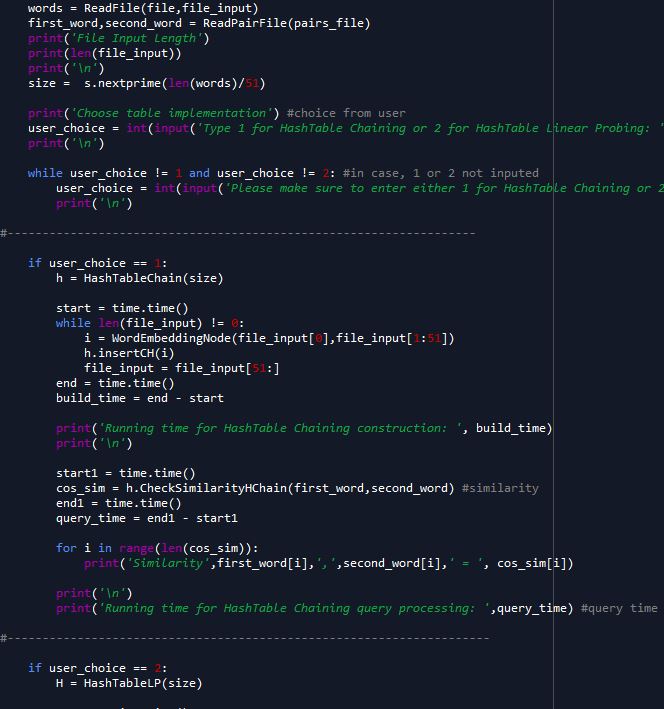
****

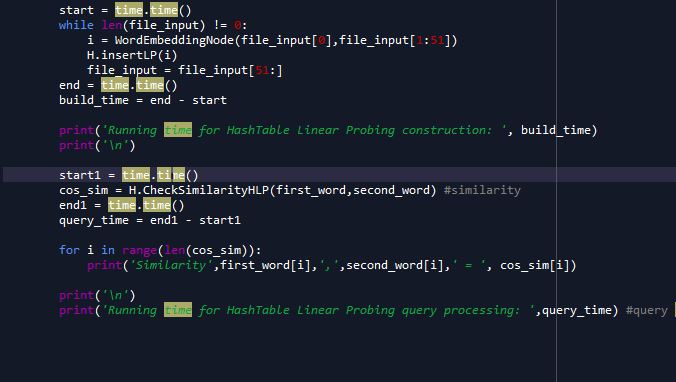
****

****

****

****

****

****

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