**Lab 5 Purpose**

The Purpose of this lab is to be able to work with both Binary Search Trees and Hash tables which need to be able to function with two text files. Meaning that the user will need to choose what method they want to use which is BST or Hash and once they selected it, the program will need to read the files and create the BST and the Hash table with the information that they contain. In addition to creating the BST and Hash table, I need to be able to show the number of nodes, the height of the BST, the running time it took to build it. After that is done I need to use the second file that contains a list of words that are side by side and compare them and get the similarity value of both words which requires the use of the first text file and an equation. When focusing on the Hash table I need to demonstrate the initial and final size of the table, then get the load factor of the hash, find the percentage of the empty lists, and get the standard deviation of the length of the lists. Again, I need to compare both words from the second text file that is made by me and compare them. Once I compare all the words in the text file I need to get the running time of how long it took to build the hash table.

**Prompt the User to Choose a Table Implementation**

**Objective**

To begin this initial part of the program, I first need to create a code that asks for the user’s choice on what table implementation the want to use, with either of them being Binary Search Tree or Hash Tables.

**Proposed Solutions**

The proposed solution for this part is really simple which here I will depend on the basics of Computer Science. To begin I know that I need to start of by asking the user what table they want which will be a Binary Search Tree or a Hash table. Once the code asks them what they can choose, the code will give them the option to input 1 for BST or 2 for HT. After the user makes their input, the input they made will go into an if statement where if they chose 1 then the program calls the method to create the BST or if they chose 2 then the program calls the method to build the Hash Table.

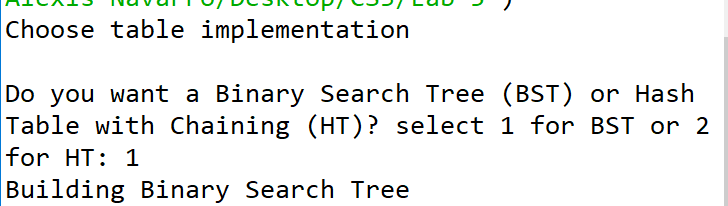
**Implementation**

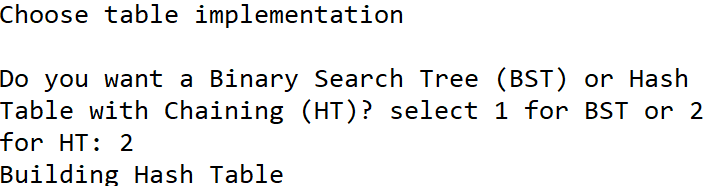
To begin creating this I first off started by printing “Choose table implementation” then after that line of code, I made the variable x to store the input of the user. In the same line where I store the input from the use, I print ‘Do you want a Binary search tree (BST) or Hash Table with chaining (HT)? select l for BST or 2 for HT:’. However, once the user inputs their choice I declare to variables called f1 and f2 which will open the two text files, but once I open them I need to type the name of the text files exactly by how its shown in the computer and to be able to completely read the file I needed to use the encoding variable and store ‘utf-8’.

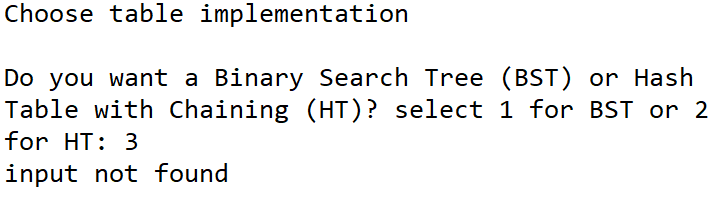
Now that everything is set up, I made an if statement where if x is equals to 1 then I print ‘Building binary search tree’ and call the method to build the BST which passes the two text files. However, if x is equals to 2 the I print ‘Building Hash Table’ and call the method to build the hash table which passes the two text files. Also, if the user inputs a number that its not one of the options then it prints ‘input not found’; at the end I need to close the two text files to be able to use it later on in my program.

**Trace**

When tracing this part of the program I will be the user the inputs the choice I want of my table. When the program begins I’m being asked if to choose a table implementation and then the program asks, ‘Do you want a Binary search tree (BST) or Hash Table with chaining (HT)? select l for BST or 2 for HT:’. After being shown the options, I choose 1 to use the Binary Search Tree and once the number was chosen, my input was stored in the computer and printed out Building Binary Search Tree which indicates to me that my choice was accepted by the program and will actually go into making the BST, but since I’m only focusing on the users input I’m not going to explain what happens at the BST method. However, if I choose 2 instead of 1 the same process would have but this time it will tell me that the program is Building the Hash table and would then go into making it. If my input is something else that is not 1 or 2 then the program will tell me that my input was not found, and it would instantly end the program.



****



Running time: O (1)

**Read the texts file, create a BST**

**Objective**

For this part of the code I need to be able to open the text files and to be able to read them in order to use the information they contain to create a complete BST, but when making the BST I need each node in the BST must consist of a list of size two, containing the word (a string) and the embedding (a numpy array of length 50).

**Proposed Solution**

For this part I know that I already have both text files open, which means that I now need to read only the first file since that’s what they ask for me initially. However, I need to store each line of information stored into my tree, but since the first text file mostly contains many numbers and words then I would need to Insert those two different data types in the correct way. This means that the best way to read the file is to iterate through it and every time I read a line of information, I then insert it into the tree. However, since the file is huge it takes a significant amount of time to finish reading the text file and insert it into the tree.

**Implementation**

To begin creating the method to build a BST I first need to make the signature of my method in which I call it Building\_BST and has the parameters of f1 and f2 which are both the files that the method will need, however we are only using f1 which is the file provided to me. Inside the method I make an empty tree in which all the information that comes from the file will be stored. Once I make my empty tree I then proceeded to creating a for loop that would traverse f1 at each index; inside the for loop I created a variable called info which stores the values at the split of a line where there is a space. After getting the info variable to store the values, I then need to use my empty tree and make it store the values that came from the info variable. The way I stored all the values into the BST was by using the insert method that is specific to the BST. When I call the Insert method I pass the variable info at the starting index of the text file, then I pass an array that starts as the index after the starting position and goes until it reaches the end; also to make sure I can obtain the number values I need to use the code astype and declare it as a float so I am able to insert them as an integer rather than a string which if it stayed as a string then I would get an error. Since I know that the text file is about 400000 lines of information, I know that if I print that then it would take a long time to print all the values or it would crash my computer/Spyder in the process of showing the tree.

**TRACING CONTINUES AT THE END OF COMPLETING THE REQUIREMENTS FOR THE BST**

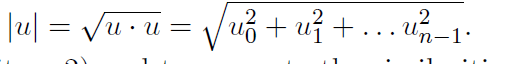
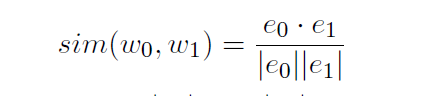
**Display BST Statistics and Similarities**

**Objective**

For this part of building the BST, I need to show all the statistics in the BST which consists of getting the number of nodes, the height of the tree, the running time to make the BST, the similarities from the second text file need to be shown, and the running time of the query processing.

**Proposed Solution**

Since I have many parts to this code I will separate them and explain what needs to be done.

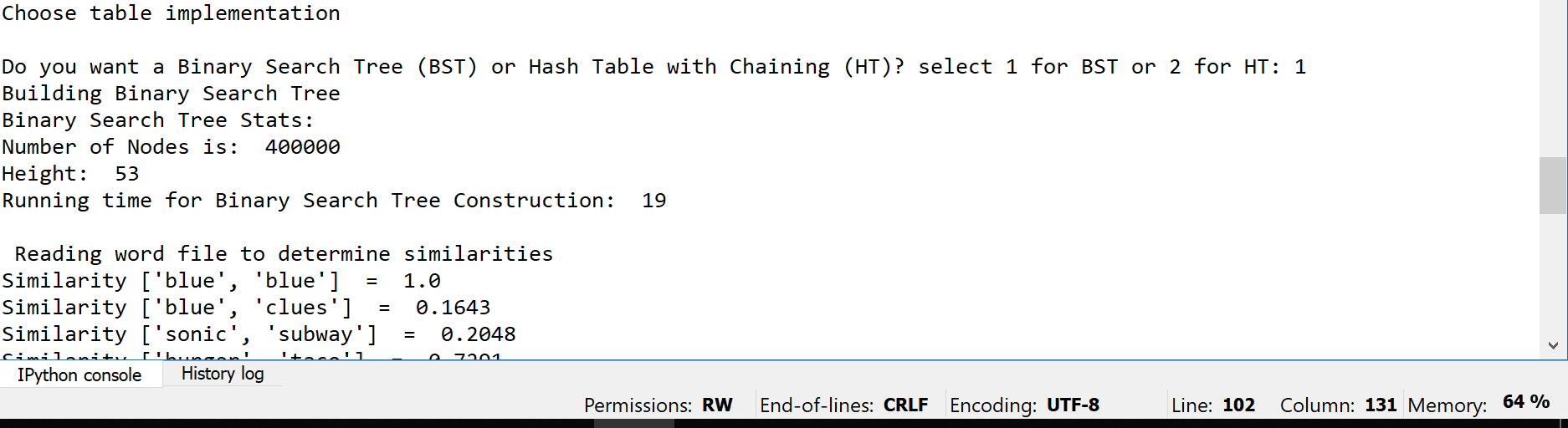
1. **Number of Nodes:** For this part of the of the BST, I want to create a method that counts all the nodes in the tree and to do that will be done by creating a new method that provides that function. In the new method, I need to use the tree that was made to store the values from the first value. When I have the tree, I would need to traverse the BST with either recursion or iterative, but here I prefer to use recursion. When traversing with recursion I need to check the left and right side of the trees and count how many nodes they have in total.
2. **Get Height:** Next, I need to be able to get the height of the binary search tree and to do that I would need another method that will allow me to get the height of the BST. When getting the height for the tree the first thing I noticed is that I can’t use the generic height method because of how different this BST. What I need is to use the tree and traverse the through the left and right since there might be a difference in height for the left and right, but since I’m looking for the total height, I need to return the count of the total height if the tree every time it traverses the nodes of the trees until it reaches the end.
3. **Running time of building the BST:** This part is a simple solution which is that I need to use the time function. Right before I make the for loop to insert the values from the text file into the tree, I need to start the time which will be counting in seconds and in order to stop counting, I need to insert another time function at the end of the for loop. Now that I have the starting time and ending time, I need to subtract the starting time from the ending time, which will give me the running time.
4. **Similarities:** For this method, I need to use another for loop because I need to read the second text file and store the values it has in a variable. However, since I only have two words in each line, I need to access text file and obtain the list of words from the index at 0 and the index at 1 since each line has only two strings in each line. After getting both the words, I now need to compare them with the use of an equation that was provided which is****. Once, I manage to create that equation in python, then now I can print the similarities and show the similarity between words.
5. **Running time for query BST processing:** For this final part, this is the same as when I got the running time to make the BST. What I need to do, is to get the starting time before the for loop and get the ending time of when the for loop ends for making the similarities; Once I have those two times I need to subtract the starting time from the ending time.

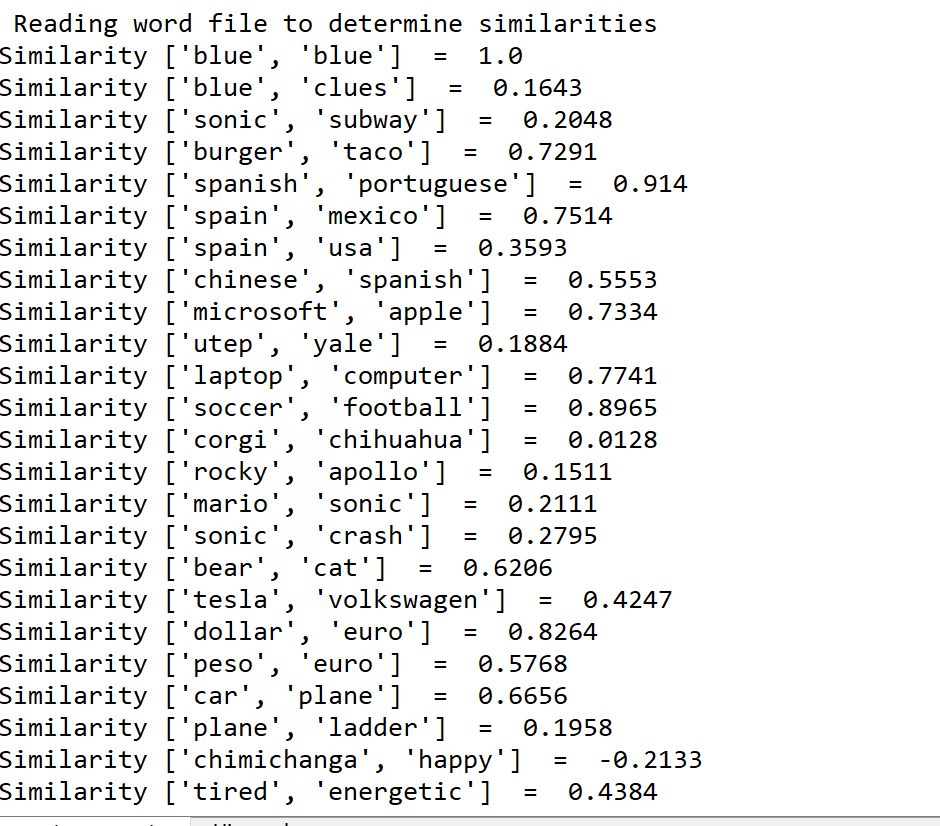
**Implementation**

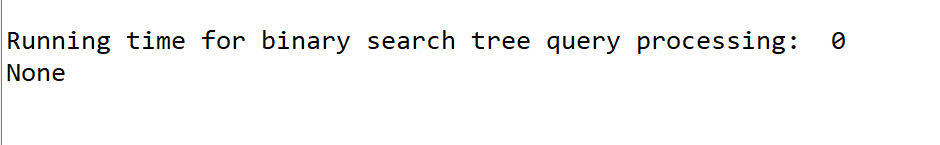
1. **Number of Nodes:** To implement my idea from before, I first need to create a new method that gets the BST that was just made from the first text file. When making the signature of the method the only parameters that I need is the tree. In the method, I will need to create a base case which is if my tree is none the I return 0. After the base case, I used an else to return 1 plus the recursive call of the method where it traverses to the left of the tree and I add again to the return the recursive call of the method where it traverses to the right of the tree. However, outside the if and else I return 0 in order to end the counting of the nodes.
2. **Get Height:** To implement my idea to this, I first need to make the signature of a new method where it takes in the parameters of the tree. Inside the method, I first make a base case that if my tree is none then I return 0. After the base case, I made two variables that store the height of the left side and the height of the right side of the tree, but the way it stores it is by using recursion to traverse onto the left and right side. After storing the height for that recursive call, I made an if statement that if my right height is less than the left height, then I return the left height + 1 and then I had an else statement that returns the right height + 1.
3. **Running time of building the BST:**  For this I didn’t require to make any new method, but instead I made a variable called start\_Time which will count the time it takes to build the BST, the time here is considered in seconds. The start\_Time variable will be inserted before the for loop where the tree is created, but after the tree is created, I have to make a variable called end\_Time which ends the counting of the start time and the end\_Time store the total seconds. When printing the running time, I need to use the equation end\_Time-start\_Time in order to get the difference among the times which gives us the total time.
4. **Similarities:** To get the similarities, between the words of the text file that I made, I first need to traverse the second file in a for loop, also the similarities will be done inside the building of the BST method. Inside the for loop I create the variable called data which will store the string of the 2nd text file every time it splits at a comma. Once I’m able to split the words, I need to make the variables e0 and e1 which will be used in the equation of similarities. The variable e0 will call the method findWord which passes the BST and the data at index 0; also, the variable e1 will also call the same method and passes the BST, but in this case, data is being passed at index 1. The findWord method is used to search the string in the tree and would return the same number if it’s in the BST. After, I return the words from the FindWord method I had to print the similarity between words and what I did is get the values of data from 0 to 2 position. Additionally I had to make the equation to get the similarity but by using python terminology, which what I did was to use the round function and in parenthesis I need to use np.sum while multiplying the values of e0 and e1, after multiplying I needed to divide the total of the multiplication and in the division I used the math.sqrt function and applied it to the np.sum of e0 times e0. Additionally, I need to use the math. sqrt function again and apply it to the multiplication of e1 times e1 while still using np.sum and at the very end I pass 4 in that print, and the 4 is to show us how many decimal numbers we want to give for the result of the similarity.
   1. **FindWord:**  When making the signature for this method I took the parameters of the T which is the BST and k which is data at 0 or 1 index. Inside the method I first made a temporary variable which will hold the BST in order to not affect the BST. After making the temporary I need a while loop to traverse the BST until its none. Inside the while loop I made an if statement that if the temp item at index 0 is the same as k then I return the temporary item at 1. If the statement does not apply, then I need to make an elif that if the temporary item at index 0 is greater than k then I traverse the temporary to the left. Then I had to create another elif, that if my temp item at index 0 is less than k, then I traverse to the right side of the temporary. If I can’t even enter the while loop meaning that my temporary is empty, then I return none.
5. **Running time for query BST processing:** For this I did the same thing as the first running time, but instead I made a variable called start\_Time2 which will count the time it takes to build the BST, the time here is considered in seconds. The start\_Time2 variable will be inserted before the for loop where the tree is created, but after the tree is created, I have to make a variable called end\_Time which ends the counting of the start time and the end\_Time2 store the total seconds. When printing the running time, I need to use the equation end\_Time2-start\_Time2 in order to get the difference among the times which gives us the total time.

**Tracing**

1. **Number of Nodes:**  For this method, the program that builds the BST calls the method numNodes which it takes in my BST that was made before. Before hand, I know that in the first text file there is 400000 lines of text which means that each line will be taken as a node in the BST which means that my output should be 400000. In the numNodes method it begins by checking if my tree is empty and since its not it doesn’t go into the base case, since it doesn’t apply to my tree the I return 1 plus the traversal of the left and right side in the tree, but its done recursively and it goes until the end of the tree which should recurse up to 400000 times and return 0 once its done. When running the program, the number of nodes is 400000 in the BST. (Look at the final image to see the output)
2. **Get Height:**  Inside the building of my BST, there is a print method that calls the method getHeight which takes my BST as a parameter. Inside the height method there is a base case that checks if my tree is empty then it returns 0 and since this is not the case for my tree yet, then I go onto storing the height into the left height and the right height variables that recurse through the method. Assuming the recursion call works properly, I then go through an if statement that it compares the height of my two sides and if the right side is less than the left then the height of the left is increased by one and if that doesn’t apply then the right height increases by 1. Also, since I know that there are 400000 nodes then this will be difficult to trace due to the size of the first text file. But assuming the method works, the method will return a height of 53. (Look at the final image to see the output)
3. **Running time of building the BST:**  when getting the running time of the building of the BST, I just have the start time to begin counting before I insert the values of the first text file into the BST. After the BST insertions I have the end time to get the final time where to code stopped inserting the values in the tree. Since this is only possible to be computed by the program, the program will count the time passed in seconds and the time is affected by the factors on the computer I’m using which is a Microsoft surface pro 6 with intel core i7 7th gen which should be quite quick to process the code considering since its modern and new. The output I get is 25 seconds. (Look at the final image to see the output)
4. **Similarities:** when getting the similarities of the two words in the text file, the program uses a for loop to traverse the 2nd text file and the first thing that happens in the text file is that the words in my text file are split by the comma they have in between. After the split the variable e0 and e1 is made to store the list of words at the 0 index for e0 and stores the list of words for e1 at index 1, but it has to call the method to find the word that is the same to the one in the BST. If the word that is from the second file, the same as the starting word from the first file then it returns the next item in the binary tree. If the word from the tree is greater (as in size) than the one from the second file then the tree traverses left, if the tree is less than the word from the second then it traverses to the right. Since the tree is large this entire part is difficult to trace the output. Back into the for loop, the similarity is printed out by using the equations that we we’re given, which the output depends on the words use however if they’re the same then they return 1 as the output.
5. **Running time for query BST processing:**  when getting the running time of the building of the similarities, I just have the start time to begin counting before I compare the words from the 2nd text file. After the comparisons I have the end time to get the final time where to code stopped comparing the values. Since this is only possible to be computed by the program, the program will count the time passed in seconds and the time is affected by the factors on the computer I’m using which is a Microsoft surface pro 6 with intel core i7 7th

****

****

****

**Read the texts file, create a Hash Table, double size, initial table size, load factor**

**Objective**

For this part of the code I need to be able to open the text files and to be able to read them in order to use the information they contain to create a complete Hash table but when making the hash choose a prime number for your initial table size and increase the size to twice the current size plus one every time the load factor reaches 1.

**Proposed Solution**

For this part I know that I already have both text files open, which means that I now need to read only the first file since that’s what they ask for me initially. However, I need to store each line of information stored into my hash, but since the first text file mostly contains many numbers and words then I would need to Insert those two different data types in the correct way. This means that the best way to read the file is to iterate through it and every time I read a line of information, I then insert it into the hash. However, since the file is huge it takes a significant amount of time to finish reading the text file and insert it into the hash.

**Implementation**

To be able to read the first file into my hash, I first have to completely rework the way I will do the hash, because when you try to compare it to BST there are many things that don’t work. To begin the hash, I used the code given to me where it uses hashing with chaining with extra fields. Since I’m using that source code, program will be changed a lot compared to the BST. To begin making the hash I made a hash table that creates a table of size 29 and its stored in the H variable. Next, I printed out the table size by getting the length of the hash table which the order I’m basing it off from the output file demonstrated by the instructions.

When I’m about to make the for loop to iterate through the first file, I first need to make a count variable that starts at 0 for later use. In my for loop I don’t split like before in the BST, but this time I split by using line.index which separates at a space. In order to store the word value, I need to store it in the variable word which stores the first value of the file until it reaches the separation (empty space). With that done I need to do the embedding where I store both of the word and numbers from the first file. The way I embed is to use np.fromstring in order to get the strings of the file into the hash and the strings have to be embedded starting from the separator to the value -1 or until there is no more. Also, I need to use the function dtype in order to declare the data type of the numbers which is a float, and the I have the function sep which separates the values at an empty space.

Now with being able to create the hash table, the next thing I need to do is to check if my load factor is 1 and to do that I use the load factor method which returns the item divided by the length of the hash. If the load factor returns 1 then I would go into the if statement where I double the size of the hash, insert the values from the original hash and my count increments by 1 each time I go through the method. In the if I first double the size of the method by first passing the hash table. In the double size method, I make the variable newHash where I would double the size of the hash and what I have to do is to call the method where I create a hash table and pass 2 times the length of h plus 1 in order to be actually able to double my table. When doing that I need 2 for loops where one allows me to iterate through the table and the other allows me to iterate through the items in the bucket of the hash. Inside both for loops I have an if else statement that if the item in my hash table at the current index is none then I print nothing. Else I call the insert method where I pass the newHash which is double the size of the original but empty, next I pass the original hash with the index for the bucket, items and 0 for the first position that contains the word. Then I pass the original hash again with the index for the bucket, items and 1 for the second values from the text.

Once the size is double I returned to making the hash table method and this time I call the insert method where I pass the new Hash that is doubled in size, the words from the text file, and embedding from the file. The insert method should insert the word and the embedding into the empty hash table where it doubled the size; After that is done then the count increments. If the load factor is not equal to 1 then the only thing the code does is to insert the word and embeddings into the hash table without it being doubled, but there is still an increment of 1 for the count.

**Tracing**

To trace this part, I know that my first text file contains 400000 lines of text which means that this trace will be hard to keep track for a person. When I pass both the files into the method, I initially create a hash table of size 29. The code begins to print the table size which is 29. Now more into the method, the code iterates through the for loop to read the first text file. In the for loop the loop iterates through the first file. In the loop the information of the text files is split and stored every time there is an empty space, after separating the values they are considered to be stored at an index. Next the strings in the text file get stored inside the word variable and after storing the strings; after storing the strings then the code embeds the numbers from the file. With the embedding done, there is an if statement that checks if the load factor is 1, but since the hash is empty and the count is zero we first go into the else statement and insert the word and embeddings into the empty hash and once the insert is done then the count increments by 1. If my load factor is equal to 1 after iterating through the loop then I would double the size of my hash, then insert the values into my new hash table and the count increments by 1 still.

**TRACING CONTINUES AT THE END OF COMPLETING THE REQUIREMENTS FOR THE HASH TABLE**

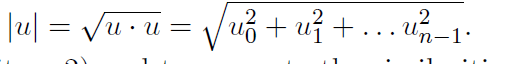
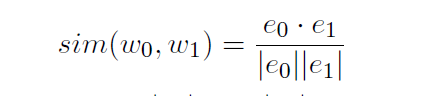
**Display Hash Table Statistics and Similarities**

**Objective**

For this part of building the Hash Table, I need to show all the statistics in the Hash which consists of getting the number of nodes, the height of the tree, the running time to make the Hash, the similarities from the second text file need to be shown, and the running time of the query processing.

**Proposed Solution**

Since I have many parts to this code I will separate them and explain what needs to be done.

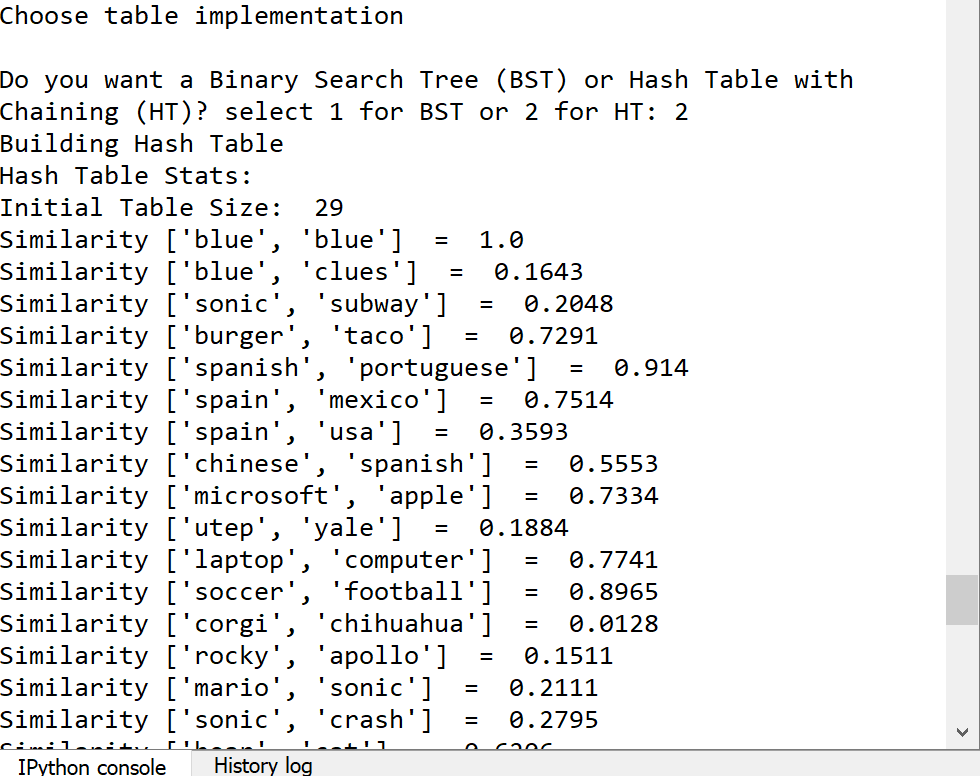
1. **Final Table size:** For this part of the of the Hash, I want to return the final size of the hash table and to do that I want to print the length of the table at the very end once I finish all the insertions for my current hash.
2. **Load factor:** For this, I plan on using the load factor that I was to in class where the load factor basically uses an integer and divides it by the length of the items of the Hash.
3. **Percentage of empty lists:** To get this, I need to create a method where I can get all the empty lists that was unused from the hash. To get the number of the empty lists I need to count the empty lists in another method and return the total. However, after getting the unused lists, I need to divide that total by the length of the hash.
4. **Similarities:** For this method, I need to use another for loop because I need to read the second text file and store the values it has in a variable. However, since I only have two words in each line, I need to access text file and obtain the list of words from the index at 0 and the index at 1 since each line has only two strings in each line. After getting both the words, I now need to compare them with the use of an equation that was provided which is****. Once, I manage to create that equation in python, then now I can print the similarities and show the similarity between words.
5. **Running time for query Hash Table processing:** For this final part, this is the same as when I got the running time to make the hash. What I need to do, is to get the starting time before the for loop and get the ending time of when the for loop ends for making the similarities; Once I have those two times I need to subtract the starting time from the ending time.

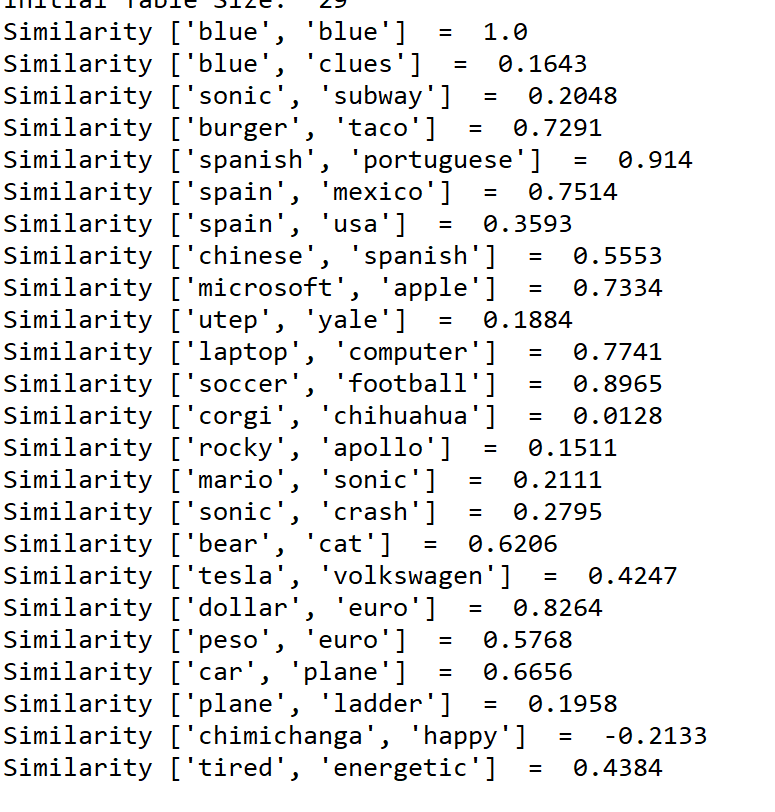
**Implementation**

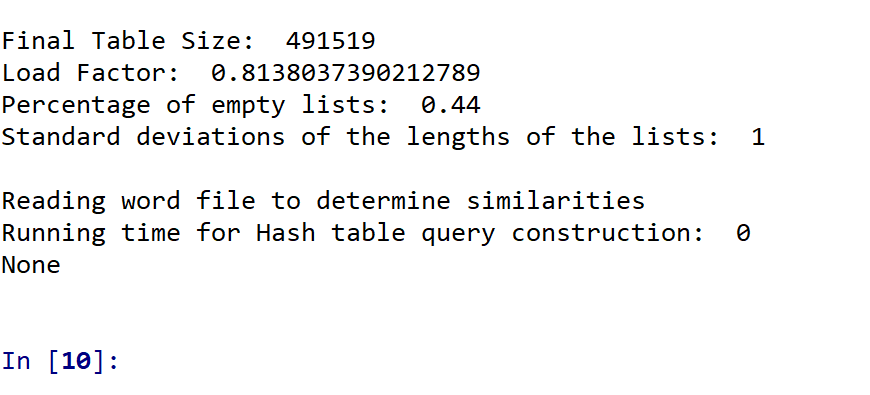
1. **Final Table size:** To implement my idea from before, the only thing I have to do is to print the length of the Hash table once I finish all the iterations to create the hash and once I finish finding the similarities.
2. **Load factor:** To implement this, I made a method that takes in the counter I and the hash table the we currently have. Inside the method I return ‘I’ divided by the length of h (i/len(H.item)
3. **Percentage of empty lists:**  For this I made a method where I can count the number of empty lists and in the method, I take in the hash table. In the method I make a count variable that starts at 0. Then I create a for loop to traverse the Hash table and what I traverse is the length of the hash. In the loop, I have an if statement that it checks if my hash is the size of 0. If the size is 0 then the count increases by 1, until the for loop reaches the end; Outside the loop I return the count. Then when getting the percentage, I need to divide my count by the length of the Hash and output the percentage in a 2-decimal format.
4. **Similarities:** To get the similarities, between the words of the text file that I made, I first need to traverse the second file in a for loop, also the similarities will be done inside the building of the hash method. Inside the for loop I create the variable called info which will store the string of the 2nd text file every time it splits at a comma. Once I’m able to split the words, I store the words into its own variable which the first variable stores the first word and the second variable stores the second word. Once I store the words I need to append them into an empty list since a hash table is essentially a list. Then I need to split my text file again and I need to split the file by each time there’s a comma. After that I need to make the variables e0 and e1 which will be used in the equation of similarities. The variable e0 will call the method find Hash which passes the hash and the data at index 0; also, the variable e1 will also call the same method and passes the hash, but in this case, data is being passed at index 1. The find\_Hash method is used to search the string in the tree and would return the same number if it’s in the Hash. After, I return the words from the Find\_Hash method I had to print the similarity between words and what I did is get the values of data from 0 to 2 position. Additionally I had to make the equation to get the similarity but by using python terminology, which what I did was to use the round function and in parenthesis I need to use np.sum while multiplying the values of e0 and e1, after multiplying I needed to divide the total of the multiplication and in the division I used the math.sqrt function and applied it to the np.sum of e0 times e0. Additionally, I need to use the math. sqrt function again and apply it to the multiplication of e1 times e1 while still using np.sum and at the very end I pass 4 in that print, and the 4 is to show us how many decimal numbers we want to give for the result of the similarity.
   1. **Find\_Hash:**  When making the signature for this method I took the parameters of the H which is the Hash and k which is data at 0 or 1 index. Inside the method I declared b to be equal to the h method that passes k and the length of the hash which will create the buckets. After making the bucket I need a for loop to traverse the hash until it reaches the end of the length of the hash. Inside the loop I made an if statement that if the hash at index b, I and at index 0 is the same as k then I return the hash item at 1. If the statement does not apply, then I return -1.
5. **Running time for query Hash Table processing:** For this I did the same thing as the first running time, but instead I made a variable called startTime which will count the time it takes to build the, the time here is considered in seconds. The startTime variable will be inserted before the for loop where the tree is created, but after the tree is created, I have to make a variable called endTime which ends the counting of the start time and the endTime store the total seconds. When printing the running time, I need to use the equation endTime-startTime in order to get the difference among the times which gives us the total time.

**Tracing**

1. **Final Table size:**  when running the code, the final table size will depend on the load factor and the 400000 lines of text. Assuming we run the complete code, and everything works perfectly then I should be able to get an output of more than 400000.
2. **Load factor:**  the load factor of this depends on the count so it the count starts at 0 and the length of my H is 29 then I divide 1 from 29 which gives the output of 0.03 as the initial load factor. this will be returned until it reaches the end of the hash and the output should be under 1.
3. **Percentage of empty lists:**  for this I call the empty method and pass the hash table and count the amount of empty lists which it should return me a value of empty lists. Once I get the amount of empty lists then I divide it by the length of the hash table and should get an output of a 2 decimal value which is under 1.
4. **Running time for query Hash Table processing:** when getting the running time of the building of the similarities, I just have the start time to begin counting before I compare the words from the 2nd text file. After the comparisons I have the end time to get the final time where to code stopped comparing the values. Since this is only possible to be computed by the program, the program will count the time passed in seconds and the time is affected by the factors on the computer I’m using which is a Microsoft surface pro 6 with intel core i7 7th

****

****

****

Academic dishonesty includes but is not limited to cheating, plagiarism and collusion. Cheating may involve copying from or providing information to another student, possessing unauthorized materials during a test, or falsifying data (for example program outputs) in laboratory reports. Plagiarism occurs when someone represents the work or ideas of another person as his/her own. Collusion involves collaborating with another person to commit an academically dishonest act. Professors are required to - and will - report academic dishonesty and any other violation of the Standards of Conduct to the Dean of Students.

# -\*- coding: utf-8 -\*-

"""

Created on Fri Mar 29 15:44:41 2019

@author: Alexis Navarro

CS2302

MW 1:30-2:50 PM

Professor:Olac Fuentes

Purpose of this code is be able to use binary search trees and hash tables with the use of a text file

with the use of a text file I need to find the similarities among words by using one of the files and using another file that contains the words that

will be used.

"""

import numpy as np

import time

import math

import statistics

class BST(object):

# Constructor

def \_\_init\_\_(self, item, left=None, right=None):

self.item = item

self.left = left

self.right = right

def Insert(T,newItem):

if T == None:

T = BST(newItem)

elif T.item > newItem:

T.left = Insert(T.left,newItem)

else:

T.right = Insert(T.right,newItem)

return T

#------------------------------------------------------------------------------

class HashTableC(object):

# Builds a hash table of size 'size'

# Item is a list of (initially empty) lists

# Constructor

def \_\_init\_\_(self,size):

self.item = []

for i in range(size):

self.item.append([])

def InsertC(H,k,l):

# Inserts k in appropriate bucket (list)

# Does nothing if k is already in the table

b = h(k,len(H.item))

H.item[b].append([k,l])

def FindC(H,k):

# Returns bucket (b) and index (i)

# If k is not in table, i == -1

b = h(k,len(H.item))

for i in range(len(H.item[b])):

if H.item[b][i][0] == k:

return b, i, H.item[b][i][1]

return b, -1, -1

def h(s,n):

r = 0

for c in s:

r = (r\*255 + ord(c))% n

return r

def loadFactor(H,i):

return i/len(H.item)

#------------------------------------------------------------------------------

#BINARY SEARCH TREE

#method to begin creating the binary search Tree

def Building\_BST(f1,f2):

T=None

start\_Time=int(time.time())#starting time

for line in f1:

info = line.split(' ')

T=Insert(T,[info[0],np.array(info[1:]).astype(np.float)]) #inserts the words and embeddings of the text file

end\_Time = int(time.time())#ending time

print('Binary Search Tree Stats:')

print('Number of Nodes is: ',numNodes(T))

print('Height: ', getHeight(T))

print('Running time for Binary Search Tree Construction: ',(end\_Time-start\_Time))

print('\n Reading word file to determine similarities')

start\_Time2=int(time.time())

for line2 in f2:

data = line2.split(',')

e0 = findWord(T,data[0])#returns the list when found

e1 = findWord(T,data[1])

print("Similarity", data[0:2], " = ", round(np.sum(e0 \* e1) / (math.sqrt(np.sum(e0 \* e0)) \* math.sqrt(np.sum(e1 \* e1))), 4)) # compute the similarity

end\_Time2=int(time.time())

print('\nRunning time for binary search tree query processing: ',(end\_Time2-start\_Time2))

def findWord(T,k):

t = T

while t is not None:

if t.item[0] == k:

#temp.item[1]

return t.item[1]

elif t.item[0] > k:

t= t.left

elif t.item[0]<k:

t = t.right

return None

#counts the number of nodes in the tree

def numNodes(T):

if T is None:

return 0

else:

return 1 + numNodes(T.left)+numNodes(T.right)

return 0

#get height of the tree

def getHeight(T):

if T is None:

return 0

leftH = getHeight(T.left)

rightH = getHeight(T.right)

if rightH<leftH:

return leftH+1

else:

return rightH+1

#------------------------------------------------------------------------------

#HASH TABLE

#Had to make the hash table differently in order to add the string fields into the table

def Building\_Hash(f1,f2):

H=HashTableC(29)

print('Hash Table Stats: ')

print('Initial Table Size: ',len(H.item))

count=0

for line in f1: # this for loop is only used for the text file provided to us

info = line.index(' ')# gets the index of the first character in the file

word=line[:info]

embedding = np.fromstring(line[info:-1],dtype=float,sep=' ') #makes the embedding

if loadFactor(H,count)==1: # if statement to check if the load factor is 1

H=doubleSize(H)

InsertC(H,word,embedding)

count+=1

else:

InsertC(H,word,embedding)

count+=1

List=list() #makes an empty list to store our information of the file

startTime=int(time.time())

for line in f2:#This for loop used the word file made by me for this program

info= line.index(',') # gets the index of the first character in the file

word=line[:info]#makes/gets the first word

word2=line[info+1:-1]#gets the second word

List.append([word,word2])

data=line.split(",")

e0=find\_Hash(H,data[0]) #returns the list when found

e1=find\_Hash(H,data[1])

print("Similarity", data[0:2], " = ", round(np.sum(e0 \* e1) / (math.sqrt(np.sum(e0 \* e0)) \* math.sqrt(np.sum(e1 \* e1))), 4)) # compute the similarity

endTime=int(time.time())

print('\nFinal Table Size: ', len(H.item))

print('Load Factor: ',loadFactor(H,num\_items(H)))

print('Percentage of empty lists: ',round((Empty(H)/len(H.item)),2))

print('Standard deviations of the lengths of the lists: ',round(statistics.stdev(len\_OfList(H))))

print('\nReading word file to determine similarities')

print('Running time for Hash table query construction: ',(endTime-startTime))

#method to find the number of items in the hash table

def num\_items(H):

Num=0

for i in range(len(H.item)):

Num+=len(H.item[i])

return Num

def Empty(H):

count=0

for i in range(len(H.item)):

if len(H.item[i])==0:

count+=1

return count

def find\_Hash(H,k):

b=h(k,len(H.item))

for i in range(len(H.item[b])):

if H.item[b][i][0]==k:

return H.item[b][i][1]

return -1

#method to be used to find the length of the standard diviation

def len\_OfList(H):

L=[]

for i in range(len(H.item)):

L.append(len(H.item[i]))

return L

#method to double the size of the table

def doubleSize(H):

newHash=HashTableC(2\*len(H.item)+1)

for i in range(len(H.item)):

for j in range(len(H.item[i])):

if H.item[i]==None:

print()

else:

InsertC(newHash,H.item[i][j][0],H.item[i][j][1])

return newHash

#------------------------------------------------------------------------------

#MAIN

print('Choose table implementation')

x=input('Do you want a Binary Search Tree (BST) or Hash Table with Chaining (HT)? select 1 for BST or 2 for HT: ')

f1 = open('glove.6B.50d.txt',encoding='utf-8') #uses the text provided to be read later on

f2 = open('List.txt',encoding='utf-8') #uses my own text file to be read later on

if int(x) == 1:

print('Building Binary Search Tree')

print(Building\_BST(f1,f2))

elif int(x) == 2 :

print('Building Hash Table')

print(Building\_Hash(f1,f2))

else:

print ('input not found')

f1.close()

f2.close()

print()