Lab 5

For this lab we are comparing words and receiving the running time it takes to compare those two words, in two different types of methods. One of them being through a BST, and the other being through has tables with chaining.

For question one, we need to ask for an input from the user. From that input, we need to decide if we should use the BST or Hash Table to do the work of comparisons. I was not able to complete this as whatever I entered would be read, and I can call on it to print in later functions. However, when using it as a way to check comparison type, it always fails. I have tried converting it to type ‘int’ in the case that it was reading the input as a string, but it still faces the same problem. It can read the problem as 2, and print it, but it will never be able to compare it to anything else using == comparisons.

For question 2, we need to read the document and place them into the BST and Hash Table. For the BST I would sort the values by word size, but I need to figure out how to separate the String from the floats. So as of now the tree does not work as I can’t exactly place them without having them all only having a right branch. For the hash tables, I was able to fit them into the hash table correctly, HOWEVER the word and the floats are still stuck together. But I can still read the hash table. The way I solved this was by creating a list from the data in the file. The 2D list is made up of strings representing each set of word + floats. I still have yet to separate/distinguish them from the same string. From there using the InsertC() method to place them into the Hash . To properly deal with the load factor, I have the function that creates the empty hash table compare the size of the hash table to the length of the list. The problem asks for a load factor of no more than 1, so if the hash table size < the length of the list, I have the size multiply \*2, then add 1. I do this until the Hash table size is larger than the length of the list.

For question 3, we are asked to check statistics regarding the file we read. For the BST I have not been able to complete it. For the Hash Table, the biggest thing I had to do was display the load factor and percentage of the empty lists. To find the load factor I just divided the length of the list by that of the hash table. (n/k) To find the percentage I subtracted the load factor from 1, and multiplied it by 100 to get the percentage of the lists that were empty. However, there was one statistic I didn’t understand, and it asked for standard deviation of lengths of lists, and I wasn’t sure if it wanted me to compare string sizes or overall list size.

For question 4, we are asked to compare the strings and find their similarity. In order to do this, I need to be able to separate the break up the string into the word and floats, but I have yet to be able to do this.

Question 5, we are asked to find the running time for the operations in question 2, and question 4. I have not tackled this question yet as I am still working towards separating string from float.

In Conclusion, I’ve learned how to read files using python, and how to better interact and implement with hash tables. However, what I need to learn is how to better deal with data types in python, so that I can effectively use chunks of information. From this I’ve gotten a better idea of how to interact and compare words, but I still have quite a way to go before I learn it. Lastly, I’ve learned I need to talk/interact with people more when it comes to Lab works. I keep thinking that I can figure out how to do something on my own, but I end up spending too much time on the problem, unable to get it done, or not have much time left once I do get it done.

Appendix:

#Patrick Brannan - Last Edited 4/12/2019

# Implementation of hash tables with chaining using strings

class HashTableC(object):

# Builds a hash table of size 'size'

# Item is a list of (initially empty) lists

# Constructor

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

self.item = []

while num\_items > size:

size = (size\*2)+1

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): #helps determine b --- how/why?

r = 0

for c in s:

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

return r

#PROBLEM 2

def ReadFile1(F): #There are about 400,000 different words in the file

f = open('glove.6B.50d.txt',encoding='utf-8')

for line in f:

F.append(line)

return F

#PROBLEM 4

def ReadFile2(F2):

o = open('wordpairing.txt', "r")

for line in o:

F2.append(line)

return F2

#Problem 1

#print('Choose Table Implentation')

#print('Type 1 for BST, or 2 for Hash Table with Chaining')

#selection = input()

#work = selection

#int(selection, 10)

#print('test', work)

#print(work)

#if work == 1: #BST WORK

# print('WIP')

#PROBLEM 2

F = []

ReadFile1(F)

E = HashTableC(53, len(F)) #Load Factor is Number of entries/Number of buckets

for f in F:

InsertC(E, f, len(f))

#END OF NUMBER 2

#Number 3

print('Hash table stats:')

print('Initial Table Size:', 53)

print('Final Table Size:', len(E.item))

load\_factor = len(F)/len(E.item)

print('Load factor:', load\_factor)

print('Percentage of empty lists:', (1-load\_factor)\*100, '%')

print('Standard deviation of the lengths of the lists:') #What is being asked for exactly?

#START OF NUMBER 4

F2 = []

ReadFile2(F2)

print(F2[0])

print(F2[1])

***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 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.***