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.

Problem 1 – we prompt a choice for the use to use a Hash Table or BST. What I did is ask the user to input 1 or 2 for either BST or hash table. Any other input asks the user to re-run the program and enter valid choice. The running time of this is O(n)

Problem 2 – We are asked to read the file and store each word with its embeddings in the Tree/Table as list containing the word, and a list of embeddings. The way I approached this was by creating a list and storing the words/embeddings I read in there. As I read the file, I split up the word from the embeddings. For the BST I sorted it in alphabetical order by getting the int related to the first letter of each word, and I increased the size of the hash table by inputting the length of the list of words along with the base size, and I have the base size increase by (n\*2)+1 until it is greater than that of the list of words.

Problem 3 – we are asked to display data for information regarding the BST or Hash, so I pulled out the information it asked for. I needed methods for some of them though. For the BST I needed to make a method to find the height of the tree. I did this by creating a counter as I traversed the tree, and it stores that counter through recursion and returns it after the whole tree has been traversed.

Problem 4 – Here we need to read a file and compute the similarity between two words on display in the file. So, I read the file and store the words in a list. I then find the words with the BST and Hash table and I get a hold of the embeddings. The embeddings I take hold of are still classified as a list of strings, so I convert them into floats before I compute the embeddings. With the two embeddings I work on finding the dot product/magnitude of them and divide the dot product by the magnitude to compute similarity. And that’s it

Number 5 – We need to find the running times combined for problems 1 and 2, so I place the time.time() function before and after problem 2 and 4 and find the difference. From those two times, I then add them up to receive the total time spent on Problems 2 and 4.

In conclusion, I learned how to better manipulate the data of how to change lists to different data types, how to split them up, how to read files, and how to handle hash tables. The most difficult part of this was getting the information I had into information I needed. So, most of my time was spent on problems 2 and 4 trying to store the lists/information properly and from there using that information for what I needed. (I need to work on asking for help on problems when I need it)

Appendix:

#Patrick Brannan - Last Edited 4/18/2019

import numpy as np

import math

import time

# 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):

# Inserts k in appropriate bucket (list)

# Does nothing if k is already in the table

b = h(k[0],len(H.item))

H.item[b].append([k[0],k[1]])

def FindC(H,k):

# Returns the embeddings

# If k is not in table, returns -1

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

#print('TEST 2:', k)

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

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

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

print('NOT FOUND')

return -1

def h(s,n):

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:

temp = line.split(None, 1)

F.append(temp)

return F

#PROBLEM 4

def ReadFile2(F2):

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

for line in o: #Stores the file

temp = line.split(None)

F2.append(temp)

return F2

def compute\_similarity(F2, H):

for f in F2:

numerator = 0

w0 = f[0]

w1 = f[1]

word0 = FindC(H, w0)

word1 = FindC(H, w1)

word0 = word0.split(None)

word1 = word1.split(None)

word0 = list(map(float,word0))

word1 = list(map(float,word1))

#Solve for Numerator

for i in range(50):

dot = word0[i] \* word1[i]

numerator += dot

#print('The Numerator is:', numerator)

#Solve for Denominator

mag0 = 0

mag1 = 0

for i in range(50):

mag0 += (word0[i]\*word0[i])

mag1 += (word1[i]\*word1[i])

mag0 = math.sqrt(mag0)

mag1 = math.sqrt(mag1)

denominator = mag0\*mag1

#print('THE DENOMINATOR IS: ', denominator)

#Solve for similarity

similarity = numerator/denominator

print('Similarity [',w0,'|',w1,'] =', similarity)

#BST OPERATIONS FOR THE PROJECT

#sys.setrecursionlimit(400000) #---Breaks the console

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):

#Sorts by character order

search = ord(newItem[0][0])

if T is not None:

currentnode = ord(T.item[0][0])

if T == None:

T = BST(newItem)

elif currentnode > search:

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

else:

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

return T

def InOrder(T):

# Prints items in BST in ascending order

if T is not None:

InOrder(T.left)

print(T.item,end = ' ')

InOrder(T.right)

def InOrderD(T,space):

# Prints items and structure of BST

if T is not None:

InOrderD(T.right,space+' ')

print(space,T.item)

InOrderD(T.left,space+' ')

def Find(T,k):

# Returns the embeddings

if T is None or T.item == k:

return T.item[1]

if ord(T.item[0][0]) < ord(k[0]):

return Find(T.right,k)

return Find(T.left,k)

def FindAndPrint(T,k):

f = Find(T,k)

if f is not None:

print(f.item,'found')

else:

print(k,'not found')

def FindHeight(T, depth):

if T is not None:

a = FindHeight(T.left, depth+1)

b = FindHeight(T.right, depth+1)

if a > b:

return a

else:

return b

else:

return depth

def BST\_similarity(F2, B):

for f in F2:

numerator = 0

w0 = f[0]

w1 = f[1]

word0 = Find(B, w0)

word1 = Find(B, w1) #RECIEVE THE EMBEDDINGS

word0 = word0.split(None)

word1 = word1.split(None)

word0 = list(map(float,word0))

word1 = list(map(float,word1))

#Solve for Numerator

for i in range(50):

dot = word0[i] \* word1[i]

numerator += dot

#Solve for Denominator

mag0 = 0

mag1 = 0

for i in range(50):

mag0 += (word0[i]\*word0[i])

mag1 += (word1[i]\*word1[i])

mag0 = math.sqrt(mag0)

mag1 = math.sqrt(mag1)

denominator = mag0\*mag1

#Solve for similarity

similarity = numerator/denominator

print('Similarity [',w0,'|',w1,'] =', similarity)

#Problem 1

print('Choose Table Implentation')

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

selection = int(input('Choice: '))

#BST WORK

if selection == 1:

#Problem 2 - ORDERED BY LENGTH

start2 = time.time()

print('Building Binary Search Tree')

print()

F = []

ReadFile1(F)

B = None

for f in F:

B = Insert(B, f)

end2 = time.time

#PROBLEM 3

print('BINARY SEARCH TREE STATS:')

print('Number of Nodes:', len(F))

print('Height:', FindHeight(B, 0))

print('Running time for BST contruction:', (end2-start2))

#Problem 4

start4 = time.time()

print("Reading word file to determine similarities")

print()

F2 = []

ReadFile2(F2)

print('Word similarities found:')

BST\_similarity(F2, B)

end4 = time.time()

#PROBLEM 5

total2 = end2 - start2

total4 = end4 - start4

print('Running time for Binary Search Tree query processing:', (total2 + total4))

#HASH TABLE WORK

elif selection == 2:

#PROBLEM 2 - I also edited the InsertC and FindC

start2 = time.time()

print('Building Hash table with chaining')

print() #Just for reading space

F = []

ReadFile1(F)

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

for f in F:

InsertC(E, f)

end2 = time.time()

#Problem 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() #For extra space

#NUMBER 4

start4 = time.time()

print('Reading word file to determine similarties')

print()

F2 = []

ReadFile2(F2)

print('Word similarities found:')

compute\_similarity(F2, E)

end4 = time.time()

#NUMBER 5

total2 = end2 - start2

total4 = end4 - start4

print('Running time for the hash table query processing:', total2+total4)

else:

print('Please run again and choose either option 1 or 2.')

***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.***